DOCTOR OF PHILOSOPHY

# An empirical study of stream-based techniques for text categorization

Thomas, Daniel

*Award date:*
2011

*Awarding institution:*
Bangor University

# An empirical study of stream-based techniques for text categorization

**Daniel Thomas**

# Abstract

### An empirical study of stream-based techniques for text categorization

**Daniel Thomas**

Due to the popularity of social networking sites such as Twitter, Facebook and blogs, the amount of electronic text is continuing to grow. There is a need to categorize these vast amounts of documents and it is no surprise that the field of text categorization is a popular one. The traditional approach to text categorization is a feature-based approach, normally processing features based on words. Stream based methods have been shown to perform well in some experimentations but there has been no thorough study of their performance on a number of major corpora and their results have not been thoroughly compared against the current state-of-the-art feature based techniques. This is an important problem as the techniques cannot be fully recognized until a thorough study has been performed.

The concept of protocols and how each affects categorization results has also not been studied thoroughly across a number of methods for several corpora. It is important to attempt to discover which stream based approaches perform best in which situations and how the choice of protocol affects their performance, if at all. It is hoped that it can be shown that for certain corpora or document lengths that certain approaches and protocols should be used. These findings could then drive the decision of which methods and protocols to use for future experiments.

An existing problem within the field of text categorization is that it is often difficult to recreate the exact experimentation conditions of previous studies. One reason for this is that the training and testing splits often differ and it was important that this study did not add to this existing problem, that the experimentations could be accurately recreated and that others would be fairly compared.

A toolkit has been developed that allows all of the methods and protocols to be compared in a consistent manner. The toolkit models the streams using suffix trees and all of the stream based methods have been implemented. In addition to the implementation of existing techniques, a number of new stream based methods have been detailed within the thesis and one of these new techniques, R-Ranges, has been shown to outperform all other methods for two of the corpora, including PPM (Prediction by Partial Matching) variants, state-of-the-art techniques that are mathematically well supported. The experimentation has also shown that the protocol (whether static or dynamic training models are used in addition to training documents of the same category being concatenated or not) does indeed affect the accuracies of each method. The concatenated dynamic protocol was found to outperform all others and performs consistently well across all methods, for all corpora. This study has now conclusively shown that the method used to categorize text must not be the only one, the selection of protocol is also just as important.

# Acknowledgements

I am sitting at home rewriting this section at three in the morning having actually already written this section within the first month of studying. I originally spoke of people who are no longer in my life and others who I have not seen in a long time. This has made me realise what a huge part of my life this study has become and how much my life has changed during this time.

I originally thanked my fiancée Amy Davies for giving me nothing but love and support no matter how tired I was, and the long nights hadn't even begun. I no longer have a fiancée named Amy Davies, thankfully she agreed to marry me on the 1st August 2010 and I would like to take this opportunity to thank her again as I see her as the greatest gift I have ever received. Amy was pregnant with our daughter Alisha when I was given the opportunity to study for a PhD, though we didn't know this at the time. I now have another daughter, Olivia, and she has ensured that I have had the opportunity of becoming the proudest father in the world, twice. There is no way I could mention my beautiful daughters without apologising for all the times I had to turn down the opportunity to play with them as I was working but I must also thank them for the unconditional love I received through this time.

I would like to thank my father, Dewi Thomas and my mother Susan Thomas who have both given me their full support throughout my life and have always encouraged me to be the best I can be. I would like to also thank my sister Michelle and brother Stephen as they have also supported me in various aspects.

I would like to thank all of my friends as they have been understanding of the lack of time I could spend with them and also for helping me move home no less than eleven times during this time. They may be thankful to hear that I am happily settled at our current home...for now at least. I would like to thank Leo Stammer who first captured my interest in the world of computing as from him I gained a thirst for knowledge and this is what led me to choose Computer Science as my course of study. I would also like to thank Dr. Robert Shepherd for helping me settle into my role as a postgraduate student during my first year.

Finally I would like to state how grateful I am to both the University Of Wales, Bangor and Dr. William J. Teahan for offering me the opportunity to study text categorization at the School Of Informatics, Bangor. Dr. Teahan has vast experience and knowledge within the field of text categorization and I am grateful to him for sharing his knowledge and experience with myself. Dr. Teahan was always available for guidance and I have come to see him as both a friend and a mentor.

# Contents

Abstract 1
Acknowledgements 2

**1 Overview 10**

1.1 Introduction 10
1.2 Background & Motivation 10
1.3 Objectives 11
1.4 Contributions 12
1.5 Thesis Outline 12

**2 Background**

**2.1 Introduction 14**

**2.2 Applications of text categorization techniques 15**
2.2.1 Authorship Attribution 16
2.2.2 Genre Categorization 16
2.2.3 Topic Categorization 17
2.2.4 Other types of classification 17
2.2.4.1 Language Identification 17
2.2.4.2 Dialect Identification 18
2.2.4.3 Style Classification 18
2.2.4.4 Document Indexing 19
2.2.4.5 A stage within Natural Language Processing Systems 19
2.2.4.6 Spam Filtering 19
2.2.4.7 Sentiment Classification 20
2.2.4.8 Gender Classification 21
2.2.4.9 Others 22

**2.3 Text pre-processing techniques 22**
2.3.1 Tokenization 22
2.3.2 Feature Selection and Extraction 22
2.3.3 Stop word removal 22
2.3.4 Stemming 23
2.3.5 Term Selection 23

**2.4 Data Sets 23**
2.4.1 Reuters-21578 23
2.4.2 Reuters-10 (R10) 24
2.4.3 RCV1-Author 24
2.4.4 20-Newsgroups 24
2.4.5 Gutenberg-10 (Gu-10) 24

**2.5 Evaluation Techniques 24**
2.5.1 Contingency Table 25
2.5.2 Precision 25
2.5.3 Recall 25

2.5.4 Accuracy 25
2.5.5 F1-Measure 25
2.5.6 Macro-averaging / Micro-averaging 25
2.5.7 The difficulty of comparing results 26

**2.6 Feature-based categorization 27**
2.6.1 Naive Bayes 27
2.6.2 N-Grams 27
2.6.3 SVM 28

**2.7 Stream-based Categorization 28**
2.7.1 C-Measure 28
2.7.2 R-Measure 29
2.7.3 PPM (Prediction By Partial Matching) 30

**2.8 Protocols 31**

**3 Extensions for stream based models 34**

**3.1 Extensions of R-Measure 34**
3.1.1 R-Ranges 37

**3.2 Extensions of C-Measure 37**

**3.3 Modifications to PPM 38**

**3.4 Complexity considerations 40**

**4 Implementation of stream-based models using Suffix Trees 42**

**4.1 Suffix Trees 42**

**4.2 Implementation 45**
4.2.1 Static C-Measure 46
4.2.2 Dynamic C-Measure 48
4.2.3 PPM Without Full Exclusions 52
4.2.4 PPM With Full Exclusions 53
4.2.5 Dynamic PPMC 55

**5 A Java based framework for implementing stream based models 57**

**5.1 Overview 57**

**5.2 Tools 58**
5.2.1 Splitting the corpora 59
5.2.2 Concatenating categories 60
5.2.3 Suffix Tree representation 61
5.2.4 Extracting suffixes 61

5.2.5 Optimisation note 62
5.2.6 Trimming concatenated models 64
5.2.7 Building the tree 64
5.2.8 Checking the counts within the suffix tree 67

**5.3 Base classes 67**
5.3.1 Comparison class 68
5.3.2 Test Collection class 68
5.3.3 Extending Test Collection class 69
5.3.4 Collection class 69

**5.4 Implementation of the algorithms 71**
5.4.1 C-Measure 71
5.4.1.1 Static case 71
5.4.1.2 Dynamic case 72
5.4.2 R-Measure 74
5.4.2.1 $r^{max}$ 74
5.4.2.2 $R_{\leq q}$ 74
5.4.2.3 $R_{\geq q}$ 74
5.4.2.4 R-Ranges 75
5.4.3 PPM 75
5.4.4 Using the toolkit 76

**6 Experimental results 79**

**6.1 Experimental setup 79**
6.1.1 Corpora setup 79
6.1.1.1 Reuters-10 79
6.1.1.2 RCV1-Author 81
6.1.1.3 20Newsgroups 82
6.1.1.4 Gutenberg 82
6.1.2 Hardware details 83

**6.2 Results 84**
6.2.1 C-Measure 84
6.2.2 PPM 92
6.2.3 R-Measure 99

**6.3 Execution times 117**

**7 Conclusions & Future Work 127**

**7.1 Discussion 127**

**7.2 Summary of chapters 127**

**7.3 Contributions 129**

**7.4 Review of aims & objectives 129**

7.5 Future work 130

8 References 132

# List of figures

2.1 Example process of text categorization 15
4.1 Suffix tree representation of string 'This is a threat•' 44
4.2 Suffix tree representation of string 'abrabra•' 45
4.3 Suffix tree representation of the string "abracadabra•" 45
4.4 Suffix trees of training file "abrabra•" and test stream "br•" 46
4.5 Dynamic suffix tree of training file "abrabra•" once • has been processed 48
4.6 Dynamic suffix tree of training file "abrabra•" once 'b' from within suffix br• has been processed 49
4.7 Dynamic suffix tree of training file "abrabra•" once 'br' from within suffix br• has been processed 49
4.8 Dynamic suffix tree of training file "abrabra•" once the suffix br• has been processed 50
4.9 Dynamic suffix tree of training file "abrabra•" once 'r' from within suffix r• has been processed 51
4.10 Dynamic suffix tree of training file "abrabra•" once the suffix r• has been processed 51
4.11 Dynamic suffix tree of training file "abrabra•" once the testing stream xbrx• has been processed 52
5.1 High level overview of jSCat 58
5.2 Example output of split parent directories 59
5.3 Example directory listing found within each split 59
5.4 Output of the concatenated files parent directory 60
5.5 Example output of concatenated training files 60
5.6 Suffix tree representation classes 61
5.7 Original tree before adding node which matches all characters within the current node 66
5.8 Tree shown in 4.10 after inserted the next node 67
5.9 Example extension of the base classes 68
5.10 Example of testing string being concatenated onto training string for dynamic cases 73
6.1 20Newsgroups C-Measure 86
6.2 Gutenberg C-Measure 88
6.3 RCV1-Author C-Measure 90
6.4 RCV1-10 C-Measure 92
6.5 20Newsgroups PPMC 93
6.6 20Newsgroups PPMD 93
6.7 Gutenberg PPMC 94
6.8 Gutenberg PPMD 95
6.9 RCV1-Author PPMC 96
6.10 RCV1-Author PPMD 97
6.11 Reuters-10 PPMC 98
6.12 Reuters-10 PPMD 98

# List of tables

2.1 Contingency Table 25
2.2 Protocols for stream-based text categorization 32
3.1 PPMC model after processing the string abracadabra with maximum order of 2 39
4.1 List of pointers within context list after processing the symbols ‘ab’ 51
4.2 Possible context list at order 1 without exclusions 52
4.3 Possible context list at order 1 with full exclusions 52
4.4 Context list example after processing the string ‘abr’ 53
5.1 Example subset of suffix model information, from which we construct a suffix tree 61
5.2 Parameter information for C-Measure setCounts method 69
6.1 The categories of Reuters 10 (R10) 80
6.2 Authors within Reuters-Author (R9) 81
6.3 The categories of 20-Newsgroups 82
6.4 The categories of Gutenberg 83
6.5 Corpora Summary 83
6.6 20Newsgroups C-Measure 85
6.7 Gutenberg C-Measure 87
6.8 Reuters-Author C-Measure 89
6.9 Reuters-10 C-Measure 91
6.10 20Newsgroups PPMC 92
6.11 20Newsgroups PPMD 93
6.12 Gutenberg PPMC 94
6.13 Gutenberg PPMD 95
6.14 Reuters-Author PPMC 96
6.15 Reuters-Author PPMD 96
6.16 Reuters-10 PPMC 97
6.17 Reuters-10 PPMD 98
6.18 20Newsgroups $R_{\leq q}$-Measure 100
6.19 20Newsgroups $R_{\geq q}$-Measure 101
6.20 R-Range average accuracies for 20Newsgroups, Concatenated Dynamic 102
6.21 R-Range average accuracies for 20Newsgroups, Concatenated Static 102
6.22 R-Range average accuracies for 20Newsgroups, Non-concatenated Dynamic 103
6.23 R-Range average accuracies for 20Newsgroups, Non-concatenated Static 103
6.24 $r^{max}$ average accuracies for 20Newsgroups 103
6.25 Gutenberg $R_{\leq q}$-Measure 105
6.26 Gutenberg $R_{\geq q}$-Measure 106
6.27 R-Range average accuracies for Gutenberg, Concatenated Dynamic 107
6.28 R-Range average accuracies for Gutenberg, Concatenated Static 107
6.29 R-Range average accuracies for Gutenberg, Non-concatenated Static 108
6.30 R-Range average accuracies for Gutenberg, Non-concatenated Dynamic 108
6.31 $r^{max}$ average accuracies for Gutenberg 108
6.32 Reuters-Author $R_{\leq q}$-Measure 110
6.33 Reuters-Author $R_{\geq q}$-Measure 111
6.34 R-Range average accuracies for Author, Concatenated Dynamic 112
6.35 R-Range average accuracies for Author, Concatenated Static 112
6.36 $r^{max}$ average accuracies for Author 112
6.37 Reuters-10 $R_{\leq q}$-Measure 113
6.38 Reuters-10 $R_{\geq q}$-Measure 114
6.39 R-Range average accuracies for Reuters-10, Concatenated Dynamic 115

6.40 R-Range average accuracies for Reuters-10, Concatenated Static 115
6.41 R-Range average accuracies for Reuters-10, Non-concatenated Static 116
6.42 R-Range average accuracies for Reuters-10, Non-concatenated Dynamic 116
6.43 $r^{max}$ average accuracies for Reuters-10 116
6.44 Average timings in seconds for C-Measure 20Newsgroups 117
6.45 Average Timings in seconds for PPMC 20Newsgroups 117
6.46 Average Timings in seconds for PPMD 20Newsgroups 118
6.47 Results for each method, for each protocol against each corpus 119
6.48 Best Results from other text categorization methods 120
6.49 C-Measure results for each of the corpora for the concatenated dynamic protocol 122
6.50 $R_{\geq q}$-Measure results for each of the corpora for the concatenated dynamic protocol 123
6.51 $R_{\leq q}$-Measure results for each of the corpora for the concatenated dynamic protocol 124
6.52 $\mathrm{r}^{\mathrm{max}}$ results for each of the corpora for the concatenated dynamic protocol 124
6.53 PPMC and PPMD results for each of the corpora for the concatenated dynamic protocol 125

# List of code samples

5.1 Inserting the next node into our tree 65
5.2 Base processing of non-concatenated comparisons 70
5.3 Base processing of concatenated comparisons 70
5.4 Coded Normalised R-Measure Value 74
5.5 Coded method for encoding all symbols for PPM 75
5.6 jSCat's main entry point 77

Chapter 1

# Overview

## 1.1 Introduction

The amount of electronic text is continuing to grow due to the overwhelming amounts of information and users on the Internet today. There is a need to categorize these vast amounts of documents and it is no surprise that the field of text categorization is a popular one. Users are becoming accustomed to having search engines retrieve the information they want in an instance with minimal effort. It is important to be able to classify information, no matter what the format, in order to ensure that the relevant information is returned.

The traditional approach to text categorization is a feature-based approach, normally processing features based on words. Hunnisett & Teahan (2004) defined a simple frequency-based measure for text categorization called the "C-Measure" which regardless of its simplicity has been proven to outperform a number of state of the art techniques. Although the effectiveness of the algorithm has been proven in a small study, no thorough study has been performed which measures the effectiveness of this approach, or indeed any other of the alternative stream based approaches in order to rigorously compare them against feature based approaches. The aim of this thesis has been to confirm that stream based approaches perform as well as the current leading feature based approaches and that these approaches should be considered in all future comparatives study within the field of text categorization.

## 1.2 Background & Motivation

It is the presence of unknowns and gaps in research that have formed the motivation behind this research. Small experiments have shown that stream based approaches achieve results that are competitive to traditional feature based approaches but there is a need for a thorough study to be performed. Hunnisett & Teahan (2004) discuss difficulties in processing substring lengths of considerable length and it is unknown if these yet to be researched lengths would further improve performance and surpass the already high accuracies that have been achieved using their technique. There is also a requirement to investigate the performance of these algorithms on several corpora in order to determine if their good performance is consistent. Part of the motivation behind the experiments in this thesis is to determine experimentally which measure performs better or whether different measures perform better in different domains.

The emphasis of the study is on models based on streams of character sequences (hence the term "stream-based" text categorization which will be mentioned numerous times throughout this thesis), but feature-based approaches shall also be reviewed for comparison, though in less detail. Compression-based approaches, usually based on the well-performed compression scheme PPM (Cleary and Witten 1984) have shown that models based on character streams are better than word models (Teahan, 1998); and we can avoid issues such as: word segmentation; normalisation e.g. stemming (reducing morphological variants to the root word); word sense disambiguation; and hapex legomena (words occurring only once within the text). The commonly held assumption that data compression is a "good" method for text categorization based on the fact that it is theoretically well founded creates a motivation to further investigate this assumption.

The methodology of how the stream-based categorization is performed based on whether static or dynamic models are used, and whether the training documents of the same category are concatenated or non-concatenated shall be termed as "protocols". The experimental performance of the newest protocol described in Hunnisett (2010), and of the other three protocols, are explained more fully by examining how these protocols are used to perform uni-label classification for text categorization, how both the protocols and methods can be implemented using suffix trees and the performance of each.

This thesis also explores the use of suffix trees as a universal data structure for storing the model representations. This data structure allows multiple similarity measures to be calculated using a single pass through the training and test sequences. Khmelev (2000) used suffix arrays to estimate probabilities for Markov models in authorship ascription studies; Khmelev & Teahan (2003), also used suffix arrays to implement R-Measure described later; but these implementations can be simplified when using suffix trees as cumulative counts can be associated with each node of the tree (Teahan, 1998; Bratko et al., 2006).

The number of protocols and algorithms being investigated brought with it a requirement for a common toolkit to be designed and implemented in order to facilitate the text categorization experiments. A toolkit has been developed in Java and its purpose is to handle all stages of the experimental process including preparation of the input data, splitting the data for cross validation, performing all experiments in a single pass and outputting the results for each experiment to allow simple comparison of each of the algorithms and procedures.

## 1.3 Objectives

The objectives of this research are as follows:

- to further investigate and perform a comparative study of stream based approaches;
- to discover which stream based approaches perform best in which situations. It was hoped to show that for certain corpora or document lengths that certain approaches and protocols should be used;
- to show that a single data structure, a suffix tree can be used to implement each of the stream based algorithms.

There is a need for results to be calculated in a consistent manner and a toolkit needed to be designed and developed to aid this. This single toolset would allow us to prepare the data and compute results before comparing them against previous examinations of other techniques. It is hoped that future studies may implement their algorithms within the toolkit so that the collection of classes and algorithms may grow and make comparing results easier and also less misleading.

## 1.4 Contributions

Though stream based approaches have been shown to perform well in small studies, there has been no complete and comparative study on their performance. This thesis has compared PPM, C-Measure and the closely linked algorithm R-Measure (Khemelev & Teahan, 2003). Variants of these algorithms, new implementations and their examination across a number of corpora and for longer suffix lengths than has been done in previous studies is novel work. The "protocols" of how stream-based categorization is performed, based on whether static or dynamic models perform best, and whether the training documents of categories should be concatenated or not, is described in detail.

A toolkit has been designed and implemented in order to facilitate the text categorization experiments. The toolkit, named jSCat, has been developed in Java and its purpose is to handle all stages of the experimental process including preparation of the input data, splitting the data for cross validation and also to perform all experiments in a single pass before outputting the results for each process to allow a simple comparison of each algorithm and procedure.

## 1.5 Thesis Outline

Chapter 2 offers a background to research within the field of text categorization and also describes a number of its applications. The chapter discusses the different approaches and techniques used within the field as well as their differences. The chapter also discusses the performance of each technique within different application domains and lists results to support this. The most popular corpora used within classification experiments are listed as well as the most popular techniques for evaluating experimental results.

Chapter 3 explains the new techniques which have been explored during the time of the study and also details all new work and improvements relating to C-Measure, R-Measure and PPM.

Chapter 4 shows how the different protocols for all models have been implemented using suffix trees.

Chapter 5 details an overview of the toolkit that has been created to aid in the calculation and comparison of the many different techniques. This chapter explains the components that exist within the toolkit and explains how the toolkit allows the introduction of categorization techniques through the extension of base classes. The implementation and also its usage are explained through discussion, figures and code samples.

Chapter 6 describes the experimental setup and methodology followed by a discussion of the results. Results will compare all algorithms within each dataset in order to discover the best performing within each corpus.

Chapter 7 summarizes all of the work included within the thesis and performs a review of the aims and objectives before concluding and identifying any future work.

Chapter 2

# Background

## Chapter Summary

The purpose of this chapter is to describe the background of research within the field of text categorization as well as describing a number of its applications. The chapter discusses the different approaches and techniques used within the field as well as their differences. The concept of Protocols, the four different variations and how each would be conducted are explained. The chapter also discusses the most popular corpora used within classification experiments as well as the most popular techniques for evaluating experimental results.

## Summary of each section

Section 2.1 offers an introduction to the field of text categorization by describing some background to the research and an abstract view of the typical steps involved within the process. Section 2.2 describes a number of its applications and describes some well known research examples. Section 2.3 discusses a number of text pre-processing techniques and how they may improve classification results. Section 2.4 describes a number of well known corpora, also known as datasets, in detail by examining the number of texts, how the texts are divided and also the differences in the size of the documents. The section also describes some examples of research that have used each of the datasets. Section 2.5 lists a number of techniques used to evaluate the performance of the text categorization including precision, recall, accuracy, F1-Measure and also the distinction between macro-averaging and micro-averaging the F-Measure. Section 2.6 offers a brief overview of feature based classification and details a couple of well known approaches. Section 2.7 discusses current stream-based algorithms including examples of how each is performed. Section 2.8 explains the four protocols and how they have been used in research to date.

## 2.1 Introduction

There is an overwhelming amount of electronic text available today and there is a need to categorize these vast amounts of documents. It is therefore no surprise that the field of text categorization is a popular one. It is important to be able to classify information, no matter what the format, in order to ensure that the relevant information is returned. People generally have little difficulty in recognising document and object categories (Watt, S. 2009). However, the speed at which users expect results to be returned, in addition to the amount of information through which to search means that indexing performed by humans has not been viable for many years. Although machines are achieving high rates of classification quickly, it could be said that human categorization will always be more accurate in some situations.

Text categorization in the past has concentrated on static situations, however, we now live in a digital era where we communicate and retrieve information from digital sources. This means that modern classifiers must now be dynamic enough to retrieve the uncategorized text as a stream, possibly directly from social networking applications such as Facebook or Twitter, or perhaps from blogs.

As far back as the 1960's, it seemed obvious that a growing amount of information was being submitted via electronic format and there was a need for these documents to be routed to the proper users (Maron, M. E. 1961). It may have been impossible to imagine back then the number of uses we have today for the application of text categorization and the number shall continue to grow so long as new technologies and ideas are developed. Due to its many applications, varied approaches and growing amounts of text, text categorization has indeed become an important research area within Information Retrieval (IR).

More formally, text categorization, also known as text classification, document categorization or document classification, is the task of automatically sorting a set of documents into predefined categories based on their content. This is a supervised learning approach as there exists documents already categorized to be used as training data which effectively define the categories. The training data is used to build a model that can be used to classify new documents, known as test data. Text categorization is not to be confused with text clustering, an unsupervised approach of which there exists no predefined categories. There is no training data and the classification is learnt from the data; similar documents are simply grouped to form a cluster.

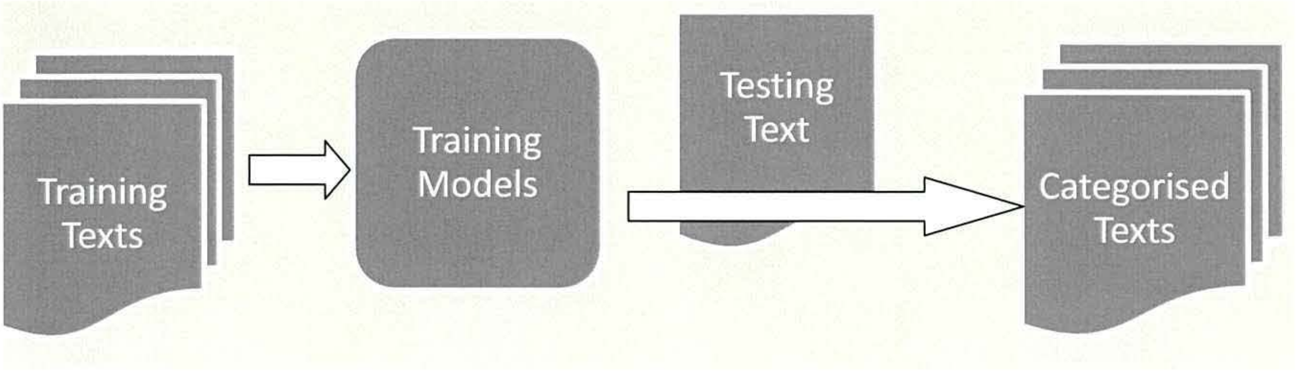


**Figure 2.1 Example process of text categorization.**

## 2.2 Applications of text categorization techniques

A number of popular applications for text categorization are detailed within this section, some of which are researched within this thesis, others are not but are detailed for completeness.

### 2.2.1 Authorship Attribution

Authorship attribution has a number of applications ranging from discovering the author of a novel to identifying the sender of an anonymous letter. Authorship attribution is more challenging than language/dialect identification as the differences among the authors' writing style is much more subtle than among different languages/dialects. Though this is the case, style and statistical properties may be noticeably different for different authors (Boggess et al., 1991). It is fair to say that there are great differences between two authors from different times with different writing style, e.g. Shakespeare and J.K. Rowling. However, within the corpora used for most text categorization experiments, which shall be discussed later, this is not the case.

A famous example of authorship attribution is the case of the Federalist Papers, of which twelve instances are claimed to have been written both by Alexander Hamilton and James Madison. Statistical analysis has been undertaken on a number of occasions to try to decide the authorship of the disputed documents based on word frequencies and writing styles, with nearly all of the statistical studies showing that all twelve disputed papers were in fact written by Madison (Mosteller & Wallace, 1984; Holmes & Forsyth, 1995; Fung, 2003).

An author may write about a number of topics and this means it is unlikely that topic-based features will perform well at discriminating among authors, e.g. a selection of features/words would not be suitable. Rather, stylistic features are the most appropriate choice; for instance, vocabulary richness (i.e. ratio between number of distinct words and total number of words), average word length, average sentence length, are important, in the sense that it is these features that tend "to give an author away" (Sebastiani, 2005).

This area of research has become more difficult with trends towards many shorter communications rather than fewer longer communications, such as the move from traditional multi-page handwritten letters to shorter, more focused emails. More recently, Twitter and other short message based web services are extremely popular and methods need to enable authorship to be determined for documents of 140 characters or less (Layton et al., 2010).

### 2.2.2 Genre Categorization

Genre classification is an important application in information retrieval (Biber, 1988; Kessler et al., 1997; Lee & Myaeng, 2004; Stamatatos et al., 2000) and more recently, Finn & Kushmerick (2006), as well as ongoing work by Santini (2007a and 2007b), and involves identifying the subject domain of a document. Examples of genres are: political, mystery and sport. A number of studies have investigated this problem usually by adapting methods found suitable for the related problem of topic categorization.

One use of genre classification could be to enable users to sort search results according to their immediate interests. Whilst visiting a bookstore people are not usually simply looking for information about a particular topic, they can often have requirements of genre as well i.e. they may be looking for novels about discoveries, Italian recipes or scientific articles etc. Genre may therefore be seen as a subject area.

A growing area of research is the categorization of single tweets and Sriram et al. (2010) define five generic classes of tweets (deals, events, opinions, news, and private messages) in order to improve information filtering. The authors represent tweets using a small set of language-dependent features to classify tweets written in English. The use of these features outperforms the BOW (bag-of-words) approach in the classification of tweets according to the typology.

There appears to be no consensus of what genre is. Though everyone believes they know what 'genre' is, unfortunately many people have different understandings of its meaning as discussed by Crowston & Williams (2000), Kwasnik & Crowston (2005), and Rosso (2005). Researchers must be careful not to confuse genre with topic as stated by Karlgren and Cutting (1994) yet some researchers (Lee and Myaeng, 2004; Stamatatos et al., 2000) seem unable to distinguish between the two and instead interpreted genre as the style of text, to be discussed later.

### 2.2.3 Topic Categorization

The task of topic categorization is a heavily researched text categorization problem (Dumais et al., 1998; Lewis, 1992; McCallum & Nigam, 1998; Teahan & Harper, 2001; Yang, 1999; Sebastiani, 2002) and concerns the problem of assigning one or more categories to a document from a list of pre-defined categories where the categories reflect the topics or subject the document is concerned with. The categories are likely to be more fine-grained than the broad categories for genre classification.

### 2.2.4 Other types of classification

#### 2.2.4.1 Language Identification

Language identification concerns the problem of identifying the language used to produce a document. It is a useful pre-processing step in information retrieval, but the task is deemed "too easy" as there are significant differences between all of the major languages, even when they are based on the same character set, as shown by experiments displaying perfect discrimination between a number of languages i.e. (House and Neuberg, 1977).

Though language identification is an easy and much studied task, it does still play an important role in a number of modern applications. Language identification is one of the most basic pre-processing stages of tasks such as summarization, question answering and translation as it is imperative to know the language of a text in order to process it. With the growing number of Internet users it is also becoming more useful to have texts processed written in a number of different languages. This is more crucial within bilingual or multilingual applications (news providers, question answering and information retrieval applications) that want to offer their services to each customer in a different language.

Other applications include travel services, translation services, national security applications and also emergency situations, as people in stressed conditions will tend to speak in their native tongue, even if they have some knowledge of the local language (Lamel and Gauvain, 1994).

### 2.2.4.2 Dialect Identification

Dialect identification is a problem closely related to language identification and it would be reasonable to argue that every person has their own dialect and that a dialect is a language in itself (Nagy et al., 2005). It is a popular categorization problem that has had much research on its subject (Nagy et al., 2005; Huang & Hansen, 2007; Nerbonne et al., 1999; Branner, 2000; Chiang et al., 2006).

In Europe, linguistic differences sharpened as the language of each nation-state was standardized. In China, standardization of spoken dialects was weaker, and mostly due to cultural influences (Branner. 2000). The variance in China's provinces where dialects are spoken can be compared with that in the Arab World. The standard written language is the same throughout the Arab world: Modern Standard Arabic (MSA). MSA is not a native language of any Arabic speaking people, i.e., children do not learn it from their parents but in school. Most native speakers of Arabic are unable to produce sustained spontaneous MSA. Dialects vary not only along a geographical continuum but also with other sociolinguistic variables such as the urban/rural/Bedouin dimension (Chiang et al. 2006).

### 2.2.4.3 Style Classification

Stylistic text categorization is another useful tool with which we can categorize documents, it is a vital tool within online libraries e.g. ERPAePRINTS (2009) and search engines. Style classification may also be known as the "type of text" or misunderstood as its genre (Lee & Myaeng, 2004; Kim & Ross, 2007). Examples of style are novels, poems, minutes, curriculum vitae and blogs etc.

As mentioned earlier, with the presence of such large amounts of digital text available today it is important to sort and manage this information in the most convenient way to the user whilst still being manageable. The ability to search media by its style as well as its topic and/or genre would allow for more relevant information being returned to the users without any additional pruning of the returned results. An example would be a user searching for the term "bread" whilst looking for a recipe, of course a number of resources including the history of bread, recipes and stores selling the product will undoubtedly be returned as well as a number of others. However if the user had additionally searched for the type of text i.e. style of document he/she required i.e. "bread recipe", the user should then only be returned documents relevant to the making of bread.

Lee and Myaeng (2004) proved that knowing the style (though they use the term genre) of a document helps to classify it based on its subject/topic more correctly, given that a classifier has been built for documents belonging to the same style. This is important and shows that we must ensure we build classifiers that not only represent the subject domain but also the style in which it was constructed.

#### 2.2.4.4 Document Indexing

A primary application of text categorization techniques is to support information retrieval systems by assigning subject categories to documents or to aid human indexers in assigning such categories (Biebricher et al., 1988; Hayes & Weinstein, 1990). Several keywords are taken from a controlled vocabulary such as a thesaurus and are assigned to a document in order to describe its subject. This transformation from a text document into a representation of text is known as indexing the document.

#### 2.2.4.5 A stage within Natural Language Processing Systems

Text categorization components are also seeing increasing use in natural language processing systems for data extraction. Categorization may be used to filter out documents or parts of documents that are unlikely to contain extractable data, without incurring the cost of more expensive natural language processing (Dahlgren et al., 1991; Grishman et al., 1991; Hobbs & Jerry, 1991).

#### 2.2.4.6 Spam Filtering

In the 1980s the term Spam was adopted to describe certain abusive users on Bulletin Board Systems who would repeat “SPAM” a huge number of times to scroll other users' text off the screen. In early Chat rooms services like PeopleLink and the early days of AOL, they actually flooded the screen with quotes from the Monty Python Spam sketch[1]. This was used as a tactic by insiders of a group that wanted to drive newcomers out of the room so the usual conversation could continue. This act, previously called flooding or trashing, came to be known as spamming. The term was soon applied to a large amount of text broadcasted by many users. It later came to be used on Usenet to mean excessive multiple posting, the repeated posting of the same message. The unwanted message would appear in many if not all newsgroups, just as SPAM appeared in all the menu items in the Monty Python sketch (Wikipedia, 2009), but is now also used to refer to unsolicited e-mail messages that are posted a large number of times.

In 2004, an estimated 62% of all email was attributed to spam, according to the anti-spam outfit Brightmail (2004). It costs money for ISPs and online services to transmit spam, and these costs are transmitted directly to subscribers (Scott Hazen Mueller, 2009). The European Union's Internal Market Commission estimated in 2001 that "junk e-mail" cost Internet users

[1] It is widely believed the term spam is derived from the 1970 SPAM sketch of the BBC television comedy series “Monty Python's Flying Circus”. The sketch is set in a cafe where nearly every item on the menu includes SPAM luncheon meat. As the waiter recites the SPAM-filled menu, a chorus of Viking patrons “SPAM, SPAM, SPAM, SPAM... lovely SPAM, wonderful SPAM”, hence “SPAMming” the dialogue. The excessive amount of SPAM mentioned in the sketch is a reference to British rationing during World War II. SPAM was one of the few meat products that avoided rationing, and hence was widely available.

€10 billion per year worldwide (Europa press release, 2001). The California legislature also found that spam cost United States organizations alone more than $13 billion in 2007, including lost productivity and the additional equipment, software, and manpower needed to combat the problem (Spam Laws, 2003).

Spammers have been documented as stealing other site's domain names via forgery, both Reply.Net and Concentric Networks have been hit this way. Indeed, Outernet, Inc. was actually attacked by one such spammer (Scott Hazen Mueller, 2009). Spam can also be used to spread computer viruses, Trojan horses or other malicious software and all of these factors have forced changes within legislation around the world. In 2003, the UK made spam a criminal offence to try to stop the flood of unsolicited messages. Under the new law, spammers could be fined £5,000 in a magistrate's court or an unlimited penalty from a jury. However the British measures are not as drastic as other anti-spam laws. Italy have imposed tough regulations to fine spammers up to 90,000 Euros and impose a maximum prison term of three years and in Australia spammers may be fined up to $1.1 million a day. On May 31, 2007, one of the world's most prolific spammers, Robert Alan Soloway, was arrested by U.S. authorities. Described as one of the top ten spammers in the world, Soloway was charged with 35 criminal counts, including mail fraud, wire fraud, e-mail fraud, aggravated identity theft and money laundering. Prosecutors allege that Soloway used millions of computers to distribute spam during 2003. This is the first case in which U.S. prosecutors used identity theft laws to prosecute a spammer for taking over someone else's Internet domain name (Wikipedia 2009).

Andrej Bratko is well known within the field of text categorization for his research on spam filtering whether it be for using compression models such as PPM (2005a, 2006a, 2006b, 2006c) or character-level Markov Models (2005b). As within this thesis, Bratko (2006) dynamically updates the training models when processing the testing text and he has also found that in the case of spam detection, pre-processing steps are often exploited by spammers in order to evade filtering.

### 2.2.4.7 Sentiment Classification

Sentiment classification is the process of computationally determining whether a document is labelled as a positive or negative evaluation of a target object. The target object may be a film, book, album etc as long as the author has a positive or negative view on the subject. An opinion may also be neutral but these are generally uncovered by this area of research. There is not a great deal of evidence of research within this field when compared to others such as topic, gender and style classification, however, this area of research has become popular in this decade. This is due to the rapid growth in on-line discussion groups and review sites and possibly also because it seems to be a challenging area of research (Pang et al., 2002) with studies not achieving the high accuracies that can be found within the other areas of text categorization.

Important current applications of this area include data and Web mining, analysis of blogs or market trends and consumer opinions (Dave et al., 2003) and the automatic filtering of abusive messages (Spertus, 1997). Other possible uses may be for politicians to track public opinion, reporters to track public response to current events and for stock traders to track financial opinions (Turney, 2002). Many review sites allow the option to include a rating as well as your written opinion (Amazon, Rotten Tomatoes etc), this allows researchers to easily generate a corpora with which to work with by for example assigning the number of stars given as a rating for the body of text.

The research within this area has so far fallen into two categories, the sentiment orientation of the document by comparing the number of positive words or sentences against the number of negative ones (Turney, 2002; Kennedy & Inkpen, 2005; and more recently Miyoshi & Nakagami, 2007); and the second is using machine learning techniques (Mullen & Collier, 2004; Pang et al., 2002). Gamon & Aue (2005) improved the results of a sentiment orientation classifier by combining it with the bootstrapping approach described by Nigam et al. (2000). Read (2005) demonstrated that in order to get reasonable results, the training and testing data must not only be relevant with regards to topic, but the time-period and domain are also important. He also investigated the use of emotional symbols (i.e. smilies) as they have the potential to be independent of domain, topic and time.

An interested note which also demonstrates the difficulty of the task follows a statement by Pang et al. (2002) that it is essential to also distinguish which sentences within the document are relevant to the item being reviewed. As an example "I hate the leading actor in this film, I think he is boring. He has no talent and normally stars in boring films of which I have hated them all. Yet I love this film!" has a majority of negative words and sentences yet a human can easily tell that the review of the film is a positive one. This is because the majority of the text is not relevant to the movie but to the actor himself.

#### 2.2.4.8 Gender Classification

Linguists have attempted to identify differences in linguistic styles between males and females for decades (Trudgill, 1972; Lakoff, 1975; Labov, 1990; Biber, 1995; Schiffman, 2002). Differences were originally found within speech but researchers have since also investigated the possibility of applying these findings to determine differences within written text. This has indeed brought researchers to test these theories within the field of text categorization, to see if it is possible to determine whether the author of a document is male or female.

Biber (1995) termed females writing style as "involved", they are more likely to specify relationships among the people and things within their text. The writing style of males is termed as being "informative", they are primarily concerned with specifying the properties of objects as well as using a greater use of swearing (Rayson et al., 1997). These findings have since been supported by a number of other researchers (Mulac et al., 2001; Pennebaker et al., 2003; Groom & Pennebaker, 2005). It is clear to see that there are indeed a number of applications of text categorization techniques and the exact techniques and successes of each shall be highlighted within later sections.

#### 2.2.4.9 Others

Another application of text categorization is within text understanding systems. Categorization may be used to filter out documents or parts of documents that are unlikely to contain extractable data, without incurring costs of more complex natural language processing, Dahlgren et al. (1991). Finally, the categorization itself may be of direct interest to a human user, as in judging whether a threatening letter against a government official signifies real danger, Hardt (1988).

## 2.3 Text pre-processing techniques

Pre-processing steps can reduce the storage space required, memory requirements and improve classification time, but at what cost? It has been shown that performing pre-processing steps on the documents may harm classification (Yu, B. 2008 and Bratko, A. 2006). Bratko explained that in the case of spam detection, pre-processing steps are often exploited by spammers in order to evade filtering.

Often it is the case that after pre-processing steps have been applied, unless the steps were thoroughly explained, it can be impossible to reproduce the same experiment at a later date for comparison or verification. This problem is reduced in the case of stream-based methods as the original data is often unmodified. The pre-processing steps often used within feature-based techniques which are omitted from stream-based and text compression techniques are discussed here for completeness:

### 2.3.1 Tokenization

The goal of tokenization is to separate text into individual words, i.e. "We're going to be late." becomes "We ' re going to be late .". The word splitter (Word Splitter, 2009) is a simple script that reads plain text and outputs the words with spaces between every word and punctuation mark, and this format is needed by tools such as POS (Part of speech) taggers.

### 2.3.2 Feature Selection and Extraction

Feature selection chooses which features should be used in classification. In text categorization, features are often the frequency of words appearing in a document. By reducing the feature space, it is not only known to increase the efficiency of the training and test processes, but can also reduce the risk of over fitting the model to data. Feature extraction computes the chosen features from an input document. In statistical classification, features are represented in a numerical vector, which is then used by the classifiers. Feature selection involves stop word removal, stemming and term selection (Toman et al. 2006).

### 2.3.3 Stop word removal

Words used in text indexing and retrieval are called terms. According to the term discrimination model (Salton, G. 1975), moderate frequency terms discriminate the best.

High frequency words, which are called stop words, have low information content, and therefore have weak discriminating power. Example words are as ‘a’, ‘the’, ‘I’, ‘he’, ‘she’, ‘is’, ‘are’, etc. and are removed according to a list of common stop words such as the one by Van Rijsbergen (1979).

### 2.3.4 Stemming

Stemming reduces morphological variants to the root word. For example, “removes”, “removed”, and “removing” are all reduced to “remove” after stemming. This relates the same word in different morphological forms and reduces the number of distinctive words. The Porter stemmer (Porter, 1997) is a commonly used stemmer as used by Frakes (1992) and its implementation in many different programming languages can be found at Martin Porter (2006).

### 2.3.5 Term Selection

Even after the removal of stop words and stemming, the number of distinct words in a document set may still be too large, and most of them appear only occasionally. In addition to removing high frequency words, the term discrimination model suggests that low frequency words are hard to learn about and therefore do not help much. They should be removed to reduce the dimensions of the vector space as well.

## 2.4 Data Sets

The availability of datasets allow standard benchmarks and encourages research by providing a setting in which different research algorithms could be compared against each other, and in which the best methods and algorithms could stand out. As in other tasks, there are several common data sets in text categorization. In this section a number of these that shall be used within our later experiments are described, and though there are many more, the following are widely used and more suitable for comparing results. More detailed information regarding the distribution of classes and file sizes can be found in Chapter 6.

### 2.4.1 Reuters-21578

Reuters-21578 is the most widely used data set for text categorization. All the texts in this data set were collected from the Reuters newswire in 1987. The original dataset contained 22,173 documents, however, 595 were later found to be exact duplicates and so these were removed. The formatted version submitted by David Lewis therefore contained 21,578 documents. Although the original data set contains 21,578 texts, researchers use a data-splitting method to extract a training set and a test set. The most popular partition (Sebastiani , 2002) is the ModApte split (available at The UCI KDD Archive, 1999) which contains 12,902 documents with a fixed splitting between test and training data, 9603 training texts and 3299 test texts. This is the most used version as confirmed by Sebastiani (2002).

There are a couple of variants of this version used. One set contains 115 categories, known as Reuters 115 (R115), and according to Sebastiani (2002) are the categories with at least one training document (Alessandro Moschitti, 2008). The other, known as Reuters 90 (R90) (also available from Alessandro Moschitti, 2008), contains 90 categories. According to Joachims (1997), they are the categories containing at least one training and one testing document and now contains 9,598 documents. The majority of excluded documents are assigned to more than a single category and is therefore not useful for our study as we are only concerned with single label classification as mentioned earlier.

### 2.4.2 Reuters-10 (R10)

In order to obtain the Reuters 10 categories split (known as R10), we simply select the ten largest categories from the remaining documents, i.e. Earnings, Acquisition, Money-fx, Grain, Crude, Trade, Interest, Ship, Wheat and Corn.

### 2.4.3 RCV1-Author

RCV1 texts are short and these small samples per author can offer a greater challenge. The RCV1 corpus has already been used in author identification experiments, Hunnisett & Teahan (2004) selected the top 50 authors (with respect to total size of articles) and the same subset is used within our experiments.

### 2.4.4 20-Newsgroups

20-Newsgroups is also a common data set used for text categorization. Although 20-newsgroup is less popular than Reuters-21578, it has been used by many researchers (e.g. Baker and McCallum (1998), McCallum and Nigam (1998), Joachims (1997)). This data set consists of Usenet articles collected by Ken Lang from 20 different newsgroups. The collection consists of 19974 non-empty documents evenly distributed across 20 categories. The version used in experiments reported in this dissertation is J. Rennie's version in which duplicate postings were removed. This subset contains 18828 documents.

The articles in this data set are postings to some newsgroups, unlike Reuters-21578 are taken from newswire. The categories also do not have multiple category labels as with Reuters 21578. In addition, the category set has a hierarchical structure within confusable clusters (e.g. "sci.crypt", "sci.electronics", "sci.med" and "sci.space" are subcategories of "sci (science)").

### 2.4.5 Gutenberg-10 (Gu-10)

This dataset, used in experiments by Thaper (2001) and Marton et al. (2005) consists of 40 documents, 4 works of each of 10 well known authors, all of which have been taken from the Gutenberg Project. The works are from the following authors, Charles Dickens, Daniel Defoe, Emerson, Jane Austen, Kiplking, Shakespeare, Shaw, Twain, Wells and Wilde.

## 2.5 Evaluation Techniques

Evaluation is of fundamental importance to IR research. It is important to be able to measure the success of the research and be able to compare the results against past research. It is also just as important to evaluate in a uniform way, as it is becomes difficult to compare results unless the research being compared is measured in the same way. The most common evaluation techniques are discussed in this section.

### 2.5.1 Contingency Table

Consider a system that is required to make $n$ binary decisions, each of which has exactly one correct answer, namely yes or no. The result of $n$ such decisions can be summarized by a contingency table, as shown in table 2.1. Each entry in the table specifies the number of decisions of the specified type. For instance, $a$ is the number of times the system decided true, and true was in fact the correct answer. Common metrics for text categorization evaluation are calculated based on the following contingency table and are discussed here.

| | **True is Correct** | **False is correct** |
|---|---|---|
| **Assigned True** | *a* | *b* |
| **Assigned False** | *c* | *d* |

**Table 2.1 Contingency Table.**

### 2.5.2 Precision

Precision is the proportion of items assigned to a category which are true members of that category. It is a measure of the number of true positives and is defined as $a/(a+b)$.

### 2.5.3 Recall

Recall is the proportion of correctly classified examples of a category. It is defined as $a/(a+c)$.

### 2.5.4 Accuracy

This measures the proportion of all decisions that were correct decisions. It is defined as $(a+d)/(a+b+c+d)$.

### 2.5.5 F1-Measure

It is possible to modify the classifiers to obtain either a higher recall or precision and the F1-measure combines both precisions. It is defined as $2rp/(r+p)$ where $r$ and $p$ are recall and precision respectively.

### 2.5.6 Macro-averaging / Micro-averaging

As F-measure is computed for each category, in order to evaluate its performance across all categories, the F-measures must be averaged. There are two conventional methods, namely macro-averaging and micro-averaging (Lewis, D., 1991). Macro-averaged performance scores are computed by first computing the scores for the per-category contingency table and then averaging these per-category scores to compute the global means. Micro-averaged performance scores are computed by first creating a global contingency table whose cell values are the sums of the corresponding cells in the per-category contingency table, and then use this global contingency table to compute the micro-averaged performance scores.

There is an important distinction between macro-averaging and micro-averaging. Micro-averaging performance scores give equal weight to every document, and is therefore considered a per-document average. Likewise, macro-average performance scores give equal weight to every category, regardless of its frequency, and is therefore a per-category average. The number of documents in each category within the datasets used for the experimental results contained in this thesis varies considerably. Because of this, micro-averaging, a per-document averaging is more suitable for the results in this thesis.

### 2.5.7 The difficulty of comparing results

It is worth mentioning the importance of releasing accurate data as incorrect data leads to difficulties when attempting to compare results with that of previous experiments. The lack of standard data collections is a problem that has been discussed by Yang (1999) and is still a problem to this day as it is possible for experiments to use the same corpora but results can differ greatly when different training and testing splits are used. Similar problems have occurred with published research within the sub-field of stream-based categorization. Teahan and Harper (2001) used a different set of categories from 20Newsgroups based on the size of the training data, but this was misinterpreted by Marton (2005), who then used these categories as though it was a known subset. The files contained within each split of all experiments are listed in the attached DVD so that all experiments can be accurately repeated.

It is important to note that inseparability on some Reuters categories is often due to dubious documents or obvious misclassifications of the human indexers. An important discovery is that within all 155 categories, 984 contained little more than the words "Blah blah blah". The same was also true for 719 of the files when tested on only the top ten categories.

A simple experiment on this dataset showed that there are still many duplicates located within the Reuters dataset and supports findings by Khmelev and Teahan (2003). Within the collection of all 115 categories, a total of 4381 duplicates were found, over 32% of the total number of files. 1183 of these were testing files and 3198 were training. Duplicates can also be found once all but the top ten categories have been removed from the collection. In fact over 19% of the remaining files are still duplicates, and these are found only by comparing against the other categories within the top ten. 475 of these are testing files and 1447 are training files.

The Newsgroups corpus is also not without problems as the files within the corpora do contain a significant amount of redundant data, i.e. text representations of attached files such as images and archives. Ideally this information should be removed, however, as no mention of this has been found previously it has been decided to not alter the contents of the files so that the experimental setup can be as correct as possible with regards to mirroring previous experiments.

If we are to effectively evaluate the performance of techniques in the future, duplicates should be removed, and files containing redundant data i.e. not much more than "blah blah blah" or file representations of attachments should also be removed. It would also be beneficial to have the 'cleansed' corpora available in a central location with the number of files and the sizes of each listed so that these values are static. This would allow for more effective comparison between research techniques and would remove ambiguity when attempting to reproduce past experiments by others. In a truly ideal situation, the results of all experiments would also be held in one place with a full description of any modifications or preprocessing that was performed as this would solve the issues raised by Yang (1999).

## 2.6 Feature-based Categorization

Feature based classifiers act upon the occurrence of words or character sequences. This approach often relies upon extracting these sequences from within the text and pre-processing steps such as those mentioned in 2.3 are used in order to reduce the complexity of the search space. Feature-based approaches, although the predominant approach in the literature, are not the focus of this dissertation and shall therefore be discussed in less detail than stream-based approaches.

### 2.6.1 Naive Bayes

Naive Bayes classifiers have long been used for text categorization tasks. A Bayes classifier is a simple probabilistic classifier based on applying Bayes' theorem and makes strong assumptions that features are independent given the class. Although more sophisticated models outperform Bayesian ones, these models are popular due to their low computational costs. The effectiveness of the models have been studied by Sahami(1996); Lewis (1998); McCallum and Nigam (1998) and Yang and Liu (1999).

### 2.6.2 N-Grams

An n-gram in the context of natural language processing can refer to either a contiguous segment of n-words or character strings of a fixed length. A document may be categorised on by its n-gram frequency list, a list of n-grams ordered by the number of occurrences in the given document. Character n-grams have been proved to be quite effective for author identification problems (Kjell et al., 1994; Peng et al., 2003; Juola, 2004; Marton et al., 2005) and as tokenization is not needed when extracting character n-grams, the approach is also language independent. They can, however, require much more computing power and time than word based approaches if attempting to calculate for multiple lengths, and n-grams of fixed length are often used in order to prevent this.

### 2.6.3 SVM

Support Vector Machines (SVMs) are learning systems that analyze data and recognize patterns and was first introduced by Boser et al. (1992). In the area of text classification SVMs separates categories within a hypothesis space and any unclassified texts that are placed within the space are categorised as belonging to the category to which it is closest. This approach has been shown to outperform many other systems in a variety of Machine Learning applications and is popular due to its efficient performance estimation (Joachims, 2002).

## 2.7 Stream-based categorization

In comparison to tokenization/feature based classification methods, a stream-based approach is similar to text compression methods in that they operate directly on the entire text sequence. Stream-based text categorization, as with compression methods, considers the text being categorized as a stream of symbols, which differs from the traditional feature-based approach which relies on extracting features from the text (Thomas and Teahan, 2007). It is also able to omit pre-processing steps such as tokenization, stopword removal and stemming altogether.

A common step between both methods is data collection. In order to objectively compare different text categorization methods, a standard data collection should be used in the evaluation experiments. However, this appears to be a serious problem. There are several different collections, and even when the same collection is chosen, there are many alternative ways that the data in the collection are used for training and testing.

The remainder of this chapter will describe existing stream-based methods that have previously been described in the literature. These will be used in the experimental results detailed in subsequent chapters.

### 2.7.1 C-Measure

Hunnisett & Teahan (2004) defined a simple frequency-based measure for text categorization called the "C-Measure" that uses the sum of the number of common substrings (or "contexts") of a fixed length between the training and test documents represented as text strings. Regardless of its simplicity, the technique has been proven to outperform a number of state of the art techniques (Hunnisett & Teahan, 2004). The results found in Hunnisett & Teahan (2004) suggest that the classification performance of context-based classifiers increases with a higher order character context. Hunnisett & Teahan (2004) did express the need to investigate this claim for higher orders but were unable due to the memory constraints of their software.

Formally, let the set of symbols in the testing text $T$ be $x_1 \dots x_N$ and $k$ be the order of the model (i.e. the fixed context length used for the model). Let $d_i(X) = 1$ if context $X$ is present in both the training text $S$ and testing text $T$, 0 otherwise. Then the C-Measure is defined as:

$$c_k(T|S) = \sum_{i=k}^{|T|} d_i(x_{i-k+1}, \dots, x_i). \qquad (2.1)$$

Here, for the definition of $c_k(T|S)$, the standard notation from probability theory is being used to indicate that the C-Measure for a given testing document $T$ is being calculated with respect to training document $S$ - i.e. $(T|S)$.

In order to try to determine the correct class of text $T$ among $m$ classes represented by texts $S_1, \dots, S_m$, Hunnisett & Teahan (2004) suggested that the source be guessed using the following estimate:

$$\hat{\theta}(T|S_i) = arg\ max_i c_k(T|S_i). \qquad (2.2)$$

**Example 1**

Consider the training string $S$ = "abracadabra•" and testing string $T$ = "abrabra•". The count C$_4$ for substrings of length 4 is 3 as the testing substring "abra" appears twice within the training string and the substring "bra•" appears once.

The $c_k$ counts are then normalized to obtain the C-Measure, with minimum and maximum values between 0 and 1, as follows:

$$C_k(T|S) = c_k(T|S)/(|T| - k + 1).$$

**Example 2**

The normalized C-Measure for substrings of length 4 using the previous example is obtained as follows:

$$C_4(T|S) = \frac{3}{(12 - 4 + 1)} = 3/9 \approx 0.33333.$$

### 2.7.2 R-Measure

Khemelev & Teahan (2003) defined the R-Measure as a number between 0 and 1 characterising the repetitiveness of the document. The R-Measure can be found by normalising the sum of the lengths of all substrings that appear in both the training files and test files. Suppose that the collection consists of $m$ documents, each document being a string $S_i = S_i[1..|S_i|]$, where $S_i[i..|S|]$ is the $i$th suffix of document $S$. A squared $R^2$-measure of document $T$ with respect to documents $S_1, \dots, S_m$ is defined as:

$$R^2(T|S_1, \dots, S_m) = \frac{2}{l(l+1)} \sum_{i=1}^{l} Q\,(T[i \dots l]|S_1, \dots, S_m). \qquad (2.3)$$

where $l = |T|$ is the length of document $T$, T$[i \dots l]$ is the $i$th suffix of document $T$ and $Q(T|S_1, \dots, S_m)$ is the length of the longest prefix of $S$, repeated in one of documents $S_1, \dots, S_m$. For example, let us take $T$ = "cat sat on" with $T_1$ = "the cat on a mat" and $T_2$ = "the cat sat". Then:

$$R^2\left(T|T_1, T_2\right) = \frac{2}{10 \times (10+1)}\left((7+6+5+4+3)+(5+4+3+2+1)\right) \approx 0.727272$$

with $(T|T_1, T_2) = \sqrt{R^2(T|T_1, T_2)} \approx 0.852802$ . Notice in the above formula that the sum consists of two parts, (7+6+5+4+3) from the repetition of "cat sat" $= T[1 \dots 7]$ and (5+4+3+2+1) from "at on" $= T[6 \dots 10]$.

The measure was originally designed to detect plagiarism and duplicates within a text collection; however, Khemelev & Teahan also used the measure to see whether or not test documents had been correctly categorised.

### 2.7.3 PPM (Prediction By Partial Matching)

The PPM algorithm was first published by Cleary and Witten (1984) and though PPM is best known for text compression, it is also a highly effective technique when used for text categorization .PPM is a well performed compression algorithm that effectively uses a language model to estimate the probabilities of each symbol in the text (Teahan, 1998). It does this by blending the probability estimates for different length contexts by a back-off technique known as the escape mechanism. Bratko and Filipic (2005) were able to show that the PPM compression model is able to outperform word-based spam filtering methods and did so using adaptive models as shall also be investigated. They share the common goal of attempting to devise a strategy which would automatically determine the order of the PPM model that optimizes classification performance and found that an order-6 model performed best typically but that there was a need to prune the model (as has been found and is discussed in section 4.3.3).

Two well performed adaptive PPM models shall be used during this thesis, namely PPMC and PPMD (these use escape method C and D respectively (Teahan, 1998)). These models blend different order models by using an escape mechanism. These variants of Cleary and Witten's original design are based on improvements described by Moffat (1990), with PPMC now being the model of choice in most cases. A technique known as exclusions removes the counts for symbols already predicted at higher orders (i.e. context lengths). The model is adaptive as it dynamically updates the counts used by the language model as the text is processed sequentially. An alternative static variation primes the model from some training text, and then suspends updating of the model when processing the testing text. Formally, given a document $T$ of length $n$ symbols and a model $\rho_L$ for a particular category $L$, then the cross entropy is calculated as follows:

$$
\begin{aligned}
H(T|S) &= -\frac{1}{n} log_2\rho_L(T) \qquad (2.4) \\
&= \frac{1}{n}\sum_{i=1}^{n} -log_2\rho_L(X_i|X_1 \dots X_{n-1}).
\end{aligned}
$$

i.e. the average number of bits to encode the document using the model. $X_i|X_1 \dots X_{n-1}$ denotes the probability of symbol $X_i$being encoded for each context. The approach taken by Teahan (1998) is based on this calculation – stated simply, each testing text is compressed against the category models, and the category is chosen from the one used to train the model that achieves the best compression. This has proven to be a highly effective technique often achieving accuracy results competitive with other text categorization techniques. In practice, PPM uses a Markov approximation i.e. assumes a fixed order context; order 5 has been found to be competitive on most texts:

$$
H(T|S) = \frac{1}{n}\sum_{i=1}^{n} -log_2\rho_L(X_i|X_{i-5} \dots X_{i-1}). \qquad (2.5)
$$

By using frequency counts the model is able to estimate probabilities for each context and these counts are updated adaptively as the text is processed sequentially, with the occurring symbol being encoded using the prediction value of the encoding model. Should the model discover an unseen symbol, the model encodes that this event has occurred and then escapes to a lower order model and continues, attempting to encode the current symbol at a lower order. Should the symbol then be matched, the context length may again grow until either the maximum context length is reached, the end of the stream is reached or another unseen symbol is discovered, forcing us to again escape to a lower context length. A detailed example of how PPM is used to perform encoding, prediction and classification of character streams is provided in Chapter 4.

## 2.8 Protocols

Marton et al. (2005) provide an overview of three compression-based approaches in the literature to text categorization which they called SMDL (for standard minimum description length), AMDL (for approximate MDL) and BCN (for best-compression neighbour). They characterized many of the prior compression-based approaches under these three labels. We seek to re-characterize these approaches (which we call "protocols") in the following way, as shown in Table 2.2. AMDL and BCN both dynamically update the model as they employ the "off-the-shelf" technique to calculate cross-entropy. The approaches adopted by Bratko et al. (2006) for PPM spam filtering also dynamically update the training model when processing the testing text. On the other hand, SMDL and AMDL concatenates all the training data for each class, significantly reducing the number of calculations required compared to BCN which produces calculations for each training document separately.

| | Static Model | Dynamic Model |
|---|---|---|
| Concatenation of training documents in the same class | Protocol I<br>(SMDL) | Protocol II<br>(AMDL) |
| Non-concatenation of training documents in the same class | Protocol III | Protocol IV<br>(BCN) |

**Table 2.2: Protocols for stream-based text categorization and contained within brackets are where each approach used within Marton et al. (2005) resides.**

If we tabulate these two features – static versus dynamic models (see section 4 for examples of these models being implemented); and concatenation of training documents in the same class versus non-concatenation – it is quite clear that a fourth protocol presents itself (labelled as Protocol III in table 2.2). This protocol has been partially examined by Hunnisett (2010) with inconclusive results.

It is not clear which of these protocols is the most appropriate for text categorization and that was a major motivation for discovering the results reported in Chapter 6. Although the dynamic protocols II and IV are well motivated from an information-theoretic perspective, the following reasoning highlights some problems with the dynamic approach. Consider what happens at the interface between the two sequences; that is, when the learning continues into the testing sequence after the training sequence has been completed. Consider the case when the languages of the testing and training documents are clearly distinct and independent, for example, as when the languages being tested for are the natural languages English and Welsh (or English and French). There will be some common English/Welsh or English/French sequences in both sequences, but comparatively few compared to the length of the texts, and usually there is no mechanism for the learning algorithm to disambiguate between the different languages (i.e. a combination of both languages is being learnt). For this reason, it is unclear whether the co-adaptation of both the training and test sequences is desirable in these cases. Similarly, concatenation of training documents has merits as it maximizes training data, and from an information theoretic point of view, one can argue that documents in the same category can be considered to be from the same language source. But with non-concatenated documents, ranking across all documents will ensure that only the best match of the testing document out of all training documents is used to provide the category estimate.

## Chapter Discussion

This chapter has reviewed important concepts within the field of text categorization. The amount of uncategorised data in digital format is continuing to grow and text categorization techniques have good success rates at categorising this data. This chapter has shown that there are indeed a number of applications of text categorization techniques, varying from indexing to filtering through to identification of language or even an author of text. The details of these techniques and successes of each have also been highlighted within this chapter.

The more common feature based approaches perform pre-processing techniques which consumes both time and resources, but it has been shown that stream based approaches do not. Some stream based approaches which already exist have been discussed and though the research has so far been limited in this area, the results have been promising and therefore warrant further investigation. One of the problems has been that the current implementations of the algorithms require additional resources and the implementation of these using suffix trees as an alternative method are discussed in the next chapter.

Some concerns have also been noted. Datasets, pre processing steps and evaluation techniques have been discussed, and although these are all well known among the community, problems still arise. Lack of details concerning the experimental setup coupled with the proven existence of inaccurate figures leads to the inability to perform true comparisons between each of the many number of text categorization.

## Chapter 3

# Extensions for stream based models

### Chapter Summary

The purpose of this chapter is to explain the new techniques which have been explored during the time of the study. The three stream-based methods that are examined within this thesis are C-Measure, R-Measure and PPM. This chapter discusses all new work and improvements relating to these models and explains the extensions found for each of the algorithms. For C-Measure the substring lengths that can now be calculated are detailed, for PPM it is shown how the calculations can be performed by using the suffix tree and for R-Measure, all new variants of the original algorithm are detailed.

### Summary of each section

Section 3.1 discusses the extensions of R-Measure and its variants. Section 3.2 discusses all new work relating to the C-Measure and describes both static and dynamic cases. Section 3.3 describes the modifications of two PPM variants, namely PPMC and PPMD, how they have been implemented and the differences when dealing with update exclusions, no exclusions as well as static or dynamic models. Section 3.4 details the time complexities of processing the stream-based models using suffix trees.

## 3.1 Extensions of R-Measure

The R-Measure was defined by Khemelev & Teahan (2003) using the lengths of the common substrings rather than their counts but it can also be defined based on a summation of the $c_k$ counts as follows:

$$R(T|S) = \frac{r(T|S)}{\frac{1}{2}|T|(|T|+1)}.$$

The following straightforward analysis reveals the two are equivalent. If the two substrings denoted by the sequence $x_i \dots x_{i+k-1}$ are common between $T$ (testing string) and $S$ (training string), they will have their first character $x_i$ also common – this corresponds to the $c_1$ counts across all common substrings between $T$ and $S$, and contributes $+1$ to the overall sum. Similarly, the common prefix $x_i, x_{i+1}$ corresponds to the $c_2$ counts and contributes a further

+1 to the overall sum. Further common prefixes of increasing length each contribute +1 to the overall sum until the length $n$ of the sequence is reached. Essentially the contribution to the $c_k$ counts is exactly the same as the lengths of the common substrings, and therefore the R-Measure can be equivalently defined either by counts or lengths.

**Example 1**

Consider the training string "abracadabra•" again. For case 1a below, let the test string be $T =$ "abrabra•", for 1b be "abracafabra•" and for 1c be "abradacabra•". The $c_k$ and $r$ counts for these cases are as follows:

$$c_1 = 10,\ c_2 = 7,\ c_3 = 5,\ c_4 = 3,\ c_5 = 1,\ c_{6...8} = 0 \qquad \text{1a}$$
$$r = 10 + 7 + 5 + 3 + 1 = 26$$

$$c_1 = 11,\ c_2 = 9,\ c_3 = 7,\ c_4 = 5,\ c_5 = 3,\ c_6 = 1,\ c_{7...12} = 0 \qquad \text{1b}$$
$$r = 11 + 9 + 7 + 5 + 3 + 1 = 36$$

$$c_1 = 12,\ c_2 = 11,\ c_3 = 7,\ c_4 = 3,\ c_5 = 1,\ c_{6...12} = 0 \qquad \text{1c}$$
$$r = 12 + 11 + 7 + 3 + 1 = 34$$

**Example 2**

Consider the case where the training string and test string are the same. In this case, all the $c_k$ counts and $r$ count have the maximum values:

$$\begin{aligned} r^{\max} &= \sum_{k-1}^{|T|} c_k^{\max} = \sum_{k-1}^{|T|} (|T| - k + 1) \\ &= |T|^2 - \sum_{k-1}^{|T|} k + |T| \\ &= \frac{1}{2}|T|(|T| + 1). \end{aligned}$$

$r^{max}$ is used to obtain the normalized R-Measure, with a minimum and maximum value between 0 and 1.

To date, only the "complete" R-Measure has been defined, which is of course the sum of the C-counts. However, further cumulative r counts can be obtained by counting only substrings whose lengths are ≥ to some minimum $q$, as follows:

$$r_{\geqslant q}(T \mid S) = \sum_{k=q}^{|T|} c_k(T \mid S).$$

Here $r \equiv r_{\geq 1}$. The series $r_{\geq 1}, r_{\geq 2}, \dots, r_{\geq |T|}$ decreases with $r_{\geq k} > r_{\geq k+1}$ except when $c_k = 0$ then all of the remaining $r_{\geq k+1,k+2,\dots,|T|} = 0$. Only when $T = S$ does $r_{|T|} = 1$, otherwise $r_{|T|} = 0$.

**Example 3**

Consider the training string "abracadabra•" again. For case 3a below, let the test string be $T =$ "abrabra•", for 1b be "abracafabra•" and for 1c be "abradacabra•". The $c_k, r$ counts and $r_{\geq q}$ values for these cases are as follows:

$c_1 = 10, c_2 = 7, c_3 = 5, c_4 = 3, c_5 = 1, c_{6...8} = 0$ 3a
$r = 10 + 7 + 5 + 3 + 1 = 26$
$r_{\geq 2} = 16, r_{\geq 3} = 9, r_{\geq 4} = 4, r_{\geq 5} = 1, r_{\geq 6,7,8} = 0$

$c_1 = 11, c_2 = 9, c_3 = 7, c_4 = 5, c_5 = 3, c_6 = 1, c_{7...12} = 0$ 3b
$r = 11 + 9 + 7 + 5 + 3 + 1 = 36$
$r_{\geq 2} = 25, r_{\geq 3} = 16, r_{\geq 4} = 9, r_{\geq 5} = 4, r_{\geq 6} = 1, r_{\geq 7,8,9,10,11,12} = 0$

$c_1 = 12, c_2 = 11, c_3 = 7, c_4 = 3, c_5 = 1, c_{6...12} = 0$ 3c
$r = 12 + 11 + 7 + 3 + 1 = 34$
$r_{\geq 2} = 22, r_{\geq 3} = 11, r_{\geq 4} = 4, r_{\geq 5} = 1, r_{\geq 6,7,8,9,10,11,12} = 0$

Alternatively, $r_{\leq q}$ counts can be obtained by counting only substrings whose lengths are ≤ to some maximum $q$, as follows:

$$r_{\leqslant q}(T \mid S) = \sum_{k=1}^{q} c_k(T \mid S).$$

In this case, $r = r_{\leq |T|}$. The series $r_{\leq 1}, r_{\leq 2}, \dots, r_{\leq |T|}$ increases with $r_{\leq k} < r_{\leq k+1}$ except when $c_k = 0$ as all remaining counts are equal, i.e. $r_{\leq k} = r_{\leq k+1,k+2,\dots,|T|}$.

Maximum values can also be calculated to normalise the $R_{\geq q}$-Measures and $R_{\leq q}$-Measures as follows:

$$R_{\geqslant q}(T \mid S) = \frac{r_{\geqslant q}(T \mid S)}{\frac{1}{2}(|T| - q + 1)(|T| - q + 2)},$$

$$R_{\leqslant q}(T \mid S) = \frac{r_{\leqslant q}(T \mid S)}{q\,|T| - \frac{1}{2}(1 - q)q}.$$

The R-Measure takes into account all substrings that are common between $T$ and $S$. However, in certain text categorization domains, such as text containing a large proportion of natural language, the shortest substrings are essentially poor for discriminating between many different $T$ and $S$ since these short substrings are common across all strings. The $q$ threshold used in the $R_{\geq q}$-Measure can be used to eliminate these strings from the calculations.

Indeed, Hunnisett & Teahan (2004) found in authorship experiments with the $C_k$-Measure that much longer substrings performed better at categorization compared to shorter ones – they found that $k = 13$ performed best but were unable to check beyond this because of memory constraints. It is possible that substrings of a greater length may indeed improve categorization performance and thanks to the toolkit discussed in the next chapter, much greater substring lengths can now be examined. In contrast, compression-based language modelling approaches using variable order Markov models base their measures only on the shorter substrings and eliminate the longer ones from their calculations in a manner similar to $R_{\leq q}$-Measure, most probably due to the exponentially large number of states for higher-order models. It is not clear which approach is preferable, or why there is a variance between the count-based and compression-based approaches. Part of the motivation behind the experiments in this thesis is to determine experimentally which measure performs better or whether different measures perform better in different domains.

### 3.1.1 R-Ranges $i \leq R \leq q$

A final R-Measure can be calculated that summates $c_k$ counts where $i \leq k \leq q$ where $i$ and $q$ are the desired minimum and maximum substring lengths respectively. As stated above, there are differences in opinion as to whether shorter or longer substrings are better at categorization and therefore it seems useful to also investigate the summation of counts between ranges of values. This will allow us to investigate results where both the shortest and longest substrings are ignored and it would be interesting to see what ranges achieve the best results and if the results are better than $R_{\geq q}$-Measure or $R_{\leq q}$-Measure, attempt to answer why. The R-Range measure is defined as follows:

$$r_{i \ldots q}(T|S) = \sum_{k=i}^{q} c_k(T|S) = r_{\leq q} - r_{\leq i-1}.$$

## 3.2 Extensions of C-Measure

As mentioned in 2.7.1, Hunnisett & Teahan (2004) were only able to calculate counts up to $c_{13}$. The new toolkit mentioned in chapter 5 is now able to surpass this point, though results are limited to $c_{50}$ in order to calculate results within reasonable time on a relatively standard desktop computer with 1GB of memory. It is possible to calculate results past this point but it will be shown that it is not beneficial to do so as substrings of great length are not useful for categorization and should only be considered for the task of duplicate document detection. The increase in performance has been achieved through a number of factors including the use of suffix tree models, pruning, and other techniques that are discussed in chapter 5.

The number of substrings of length *k* that are found within both the training document S and testing document *T* is defined as:

$$c_k(T|S) = \sum_{i=k}^{|T|} d_i(x_{i-k+1}, \dots, x_i). \qquad (2.1)$$

where $k$ is the order of the model and $C(x_i - k + 1 \dots x_i|S) = 1$ if context $x_i - k + 1 \dots x_i$ (all substrings) are present within the training text and is equal to 0 if all substrings are not present.

**Example 1**

Consider the training string $S$ = "abracadabra•" and testing string $T$ = "abrabra•". The count $C_4$ for substrings of length 4 is 2 as the testing substring "abra" appears twice within the training string.

The $c_k$ counts are then normalized to obtain the C-Measure, with minimum and maximum values between 0 and 1, as follows:

$$C_k(T|S) = c_k(T|S)/(|T| - k + 1).$$

**Example 2**

The normalized C-Measure for substrings of length 4 using the previous example is obtained as follows:

$$C_4(T|S) = \frac{2}{(12 - 4 + 1)} = 2/9 \approx 0.22222.$$

## 3.3 Modifications to PPM

In order to calculate the compression ratio of a testing file a suffix tree is created that represents the training model, with the testing file being read as a stream one symbol at a time. As the symbols from the testing stream are processed, we traverse the training suffix tree, using the counts of existing nodes in order to calculate the probability of the symbol occurring at the current context length. With the use of an array of context lengths with pointers into positions within nodes of the tree, we are able to successfully track the position of each context length within the suffix tree. As we traverse the current longest context length and calculate the probability of the current symbol, we also update the position of each of the lower order pointers (see table 3.1). This is performed as though an unseen symbol was discovered, as we calculate the probability of the unseen symbol and escape to a lower context length, whose position is already held in our array. After the probability of each symbol within the stream has been calculated, the probability of each is then added to the current total until the entire stream has been processed. The testing file is then attributed to the model that offers the lowest bit rate, i.e. the highest compression ratio. Table 3.1 shows a PPMC model after processing the string "abracadabra" with maximum order of 2.

| Order $k=2$ | | | Order $k=1$ | | | Order $k=0$ | | | Order $k=-1$ | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Predictions | $c$ | $p$ | Predictions | $c$ | $p$ | Predictions | $c$ | $p$ | Predictions | $c$ | $p$ |
| ab → r | 2 | $\frac{2}{3}$ | a → b | 2 | $\frac{2}{7}$ | → a | 5 | $\frac{5}{16}$ | → $\Lambda$ | 1 | $\frac{1}{\lvert A \rvert}$ |
| → Esc | 1 | $\frac{1}{3}$ | → c | 1 | $\frac{1}{7}$ | → b | 2 | $\frac{2}{16}$ | | | |
| ac → a | 1 | $\frac{1}{2}$ | → d | 1 | $\frac{1}{7}$ | → c | 1 | $\frac{1}{16}$ | | | |
| → Esc | 1 | $\frac{1}{2}$ | → Esc | 3 | $\frac{3}{7}$ | → d | 1 | $\frac{1}{16}$ | | | |
| ad → a | 1 | $\frac{1}{2}$ | b → r | 2 | $\frac{2}{3}$ | → r | 2 | $\frac{2}{16}$ | | | |
| → Esc | 1 | $\frac{1}{2}$ | → Esc | 1 | $\frac{1}{3}$ | → Esc | 5 | $\frac{5}{16}$ | | | |
| br → a | 2 | $\frac{2}{3}$ | c → a | 1 | $\frac{1}{2}$ | | | | | | |
| → Esc | 1 | $\frac{1}{3}$ | → Esc | 1 | $\frac{1}{2}$ | | | | | | |
| ca → d | 1 | $\frac{1}{2}$ | d → a | 1 | $\frac{1}{2}$ | | | | | | |
| → Esc | 1 | $\frac{1}{2}$ | → Esc | 1 | $\frac{1}{2}$ | | | | | | |
| da → b | 1 | $\frac{1}{2}$ | r → a | 2 | $\frac{2}{3}$ | | | | | | |
| → Esc | 1 | $\frac{1}{2}$ | → Esc | 1 | $\frac{1}{3}$ | | | | | | |
| ra → c | 1 | $\frac{1}{2}$ | | | | | | | | | |
| → Esc | 1 | $\frac{1}{2}$ | | | | | | | | | |

**Table 3.1: PPMC model after processing the string abracadabra with maximum order of 2.**

## 3.4 Complexity considerations

Consider the space and time complexities for the text categorization protocols when implemented using suffix trees. Assume there are $K$ classes, $M$ training documents, and $N$ testing documents. Typically$M \gg N \gg K$. The space and time complexities are dependent on the size of the suffix trees which are linear with the size of the text. If the suffix trees are created on demand during categorization, then the space is proportional to the length of the training and testing text currently being processed. However, consider the case where we wish to create the suffix trees in advance for all training and testing texts so that these do not have to be re-created multiple times. The non-concatenated protocols (III and IV) substantially increase the complexity of the classification experiments as they require the creation of $\mathrm{M}+\mathrm{N}$ suffix trees, plus the calculation of an $\mathrm{M} \times \mathrm{N}$ matrix of similarity judgments; this is opposed to $\mathrm{K}+\mathrm{N}$ suffix trees plus $\mathrm{K} \times \mathrm{N}$ similarity judgments for the concatenated protocols (I and II).

Considering the time complexities, for the frequency-based methods, all C–Measures and the R–Measure can be calculated simultaneously in a single co-traversal of both the training and testing suffix trees where non-matching branches are not followed. $\mathrm{r}_{\mathrm{p...q}}$require a further calculation to compute the different measures for all values of p and q. This can be done by filling in a $|T| \times |T|$ (where $T$ is a testing string) matrix by iterating over $p$ and $q$ but the worst-case time complexity and space complexity for this is $O(T^2)$ compared to $O(T)$ to calculate the measures for the other formulas. However, since the series of $c_k$ counts does not change beyond the length of the longest common prefix between $T$ and $S$ (where $S$ is the training string), the average case is much better, and both the time and space requirements can be reduced considerably by only calculating counts up to the longest common prefix length.

Let us now show how c-counts may be calculated for a specific example. Consider the training string $S$ = "to be or not to be•" and testing string $T$ = "to be not•". After constructing a suffix tree for each of the strings we navigate through all nodes of the testing tree and should the character within the testing node exist within the training tree at the same depth, the count for that depth is incremented by the count of the training node i.e. the number of times that the current suffix occurs within the training string.

After matching the EOF symbol between both trees at depth 1, $\mathrm{C}_1$ would then have a count of 1 as up to this point only a single suffix of length 1 occurs in both strings and this suffix occurs only once within the training string. As we have reached the end of the current branch we would move along the nodes of the testing tree at depth 1 until a suffix is matched. The next suffix to be matched would be ' ' (the space character) and this would again add the number of times it occurs within the training string to the count $\mathrm{C}_1$. The value is currently one and as the character appears five times within the training string, the count now becomes six. As the testing node has child nodes, we then attempt to match the suffixes at increasing depths within the training tree and for all strings that are matched, the number of times they occur within the training tree is added to the current total of counts for that depth. Two characters appear after spaces within the testing string, 'b' and 'n'. Therefore we would first attempt to match the suffix " b" (space at depth 1 and 'b' at depth 2) within the training tree. This substring occurs twice within the training string and so $C_2$ now becomes 2. The only node to appear after the 'b' in the testing node is 'e' so we then attempt to match 'e' at depth 3 on the current branch within the training node. This process continues until we have either processed all of the testing suffixes and at this time we will have the counts of suffixes for all lengths between 1 and the length of the longest common substring.

The longest common substring within this example is "to be ", which has a length of 6 and appears only once within the training text. Using equation 2.1 we can calculate the count $C_6$ as follows:

$$C_6(T|S) = \frac{1}{(19 - 6 + 1)} = 1/14 \approx 0.07143.$$

We may also wish to calculate the count $C_2$in a similar manner, with the count $C_2$ for substrings of length 2 being 12 as "to", "o ", " b" and "be" each appears twice within the training string and "e ", " n", "no" and "ot" each appears once.

$$C_2(T|S) = \frac{12}{(19 - 2 + 1)} = 12/18 \approx 0.66667.$$

## Chapter Discussion

The chapter has shown new approaches and investigations including explanation of how the text categorization performance for the stream-based algorithms can be performed using suffix trees. The investigation of C-Measure for suffix lengths greater than 13, $R_{\geq q}$-Measures, $R_{\leq q}$-Measures and $i \leq R \leq q$ and having results (see chapter 6) for these measures against the corpora mentioned in 2.4 is novel and collectively this allows us to compare these approaches against the current leading techniques.

## Chapter 4

# Implementation of stream-based models using Suffix Trees

### Chapter Summary

This chapter offers an overview of suffix trees and how they can be used to implement stream based algorithms. The chapter also shows how each protocol can be modeled for each of the algorithms through use of discussion and examples.

### Summary of each section

Section 4.1 discusses suffix trees and the advantages of its uses as a representation of a text document. Section 4.2 shows how each of the stream based algorithms are implemented using suffix trees and how each protocol is implemented through use of examples.

## 4.1 Suffix Trees

A suffix tree of a string (or a document should we consider the contents of a document as a string) is a trie holding all the suffixes of that string. As all suffixes are contained, we can say that all substrings are also contained. This powerful data structure allows for quick searching of substrings and also allows for strings to be dynamically added or removed. Suffix trees have also provided one of the first linear-time solutions for the longest common substring problem. These speedups do come at a cost as storing a string's suffix tree typically requires significantly more space than storing the string itself. This approach differs from bag of word approaches i.e. Naïve Bayes as we allow for phrases and streams of symbols/words/sentences and do not ignore the order of the sequences.

Weiner first introduced the concept as a position tree in 1973 (Weiner, 1973). The construction was then simplified and the space consumption lowered by McCreight in 1976 (McCreight, 1976), and also by Ukkonen in 1995 (Ukkonen, 1995; Giegerich & Kurtz, 1997). The first linear-time online construction of suffix tress was provided by Ukkonen and the construction method is now known as Ukkonen's algorithm, though it has been criticized for the lack of space efficiency (Giegerich & Kurtz, 1997).

Suffix trees have been studied and used extensively in fundamental string problems such as large volumes of biological sequence data searching, i.e. DNA or protein sequences (Bieganski & Carlis, 1994), approximate string matches (Ehrenfeucht & Haussler, 1988) and text features extraction in spam email classification (Pampapathi & Levene, 2006). It is important to note that for most applications a lexicographic trie is unnecessary, however, a lexicographic trie allows us to take advantage of search techniques i.e. binary search algorithm, which relies on the contents being sorted to find the desired child node within a position of the trie.

If the input string $S$ of length $n$ is terminated by a special end-of-string symbol ("•") then the suffix tree has $n + 1$ leaves, one for each nonempty suffix of $S$. The end-of-string symbol is important as it allows us to find the point at which we are processing the next text within a concatenated stream. Since all internal non-root nodes are branching, there can be at most $n - 1$ such nodes, and $n + 1 + (n - 1) + 1 = 2n + 1$ nodes in total. The most apparent use of the suffix tree is as an index that allows substrings of a longer string to be located efficiently. The suffix tree can be constructed, and the longest substring that matches a search string located, in asymptotically optimal time (Larsson, 1999). An edge label within the tree is represented by a pointer into the original string and this ensures that the storage space required for each node is constant.

A sample suffix tree indexing the string $S$, 'This is a threat•' is shown below with the counts of each node displayed to its right. The string $S$ contains 17 suffixes – "•", " a threat•", " is a threat•", " threat•", "This is a threat•", "a threat•", "at•", "eat•", "his is a threat•", "hreat•", "is a threat•", "is is a threat•", "reat•", "s a threat•", "s is a threat•", "t•" and "threat•", with the substrings " " (space) occurring 3 times, "a", "h", "is ", "s " and "t" twice, and the rest occurring only once.

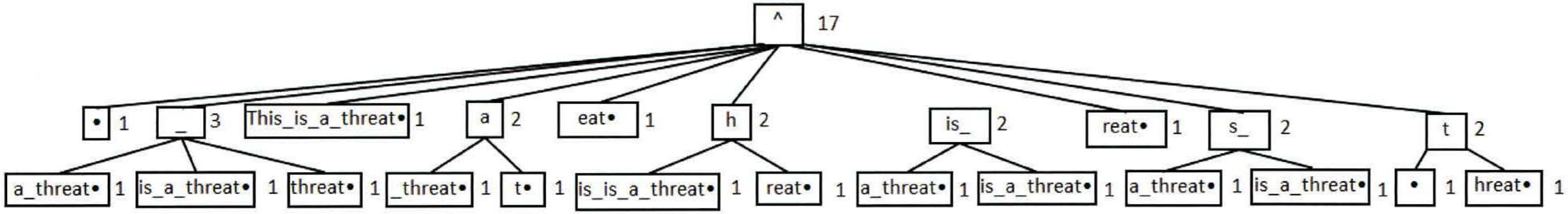


**Figure 4.1: Suffix tree representation of string 'This is a threat•'. '^' is the null string at the root of the tree and '•' is the end of string symbol. The counts of each substring are shown to the right of each node.**

## 4.2 Implementation

The remainder of this chapter will show how it is possible to compute in reasonable time and space all the stream-based methods outlined previously (C-measure, R-measure and PPM), using essentially a single pass through the test data (or its equivalent represented as a suffix tree). This step, of course, is necessarily an off-line process. Once the best measure is found, however, it can be used directly to classify unknown test strings and multiple calculations are no longer necessary.

The $C$ and $R$–Measure can be computed using this data structure in the following way. Assume that the training string $S$ be the same as the string used for figure 4.3, and the test string $T$ as that used for figure 4.2. By co-traversing both trees simultaneously in a single pass, each of the $c_k$ counts can be calculated by summing the counts of the common prefixes between $S$ and $T$. For example, the common prefixes of length 2 in the order they appear in figure 4.2 is "a•" (count is 1), "ab" (2), "br" (2) and "ra" (2) so the total sum of counts is 7. Likewise the common prefixes of length 4 are "abra" (2) and "bra•", so $c_4 = 3$. It is a simple exercise to derive the measures based solely on the $c_k$ counts and the length of string $T$ (which is the count associated with the root node). Note that this application of suffix trees to computing the $C$ and $R$–Measures is novel. In fact, Hunnisett & Teahan (2004) were not able to compute values for $C_k$ for $k > 13$ using their trie implementation because of memory constraints.

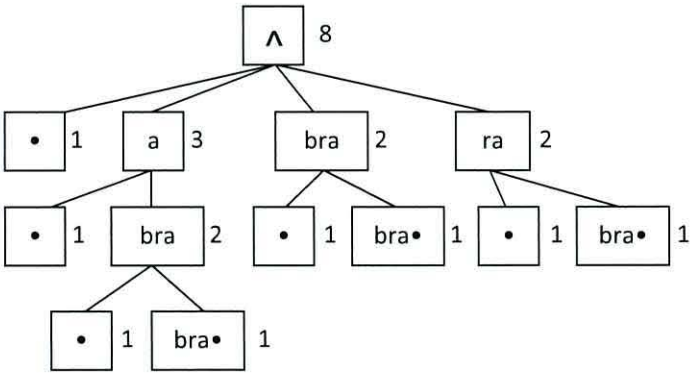


**Figure 4.2: Suffix tree representation of string 'abrabra•'. '^' is the null string at the root of the tree and '•' is the end of string symbol.**

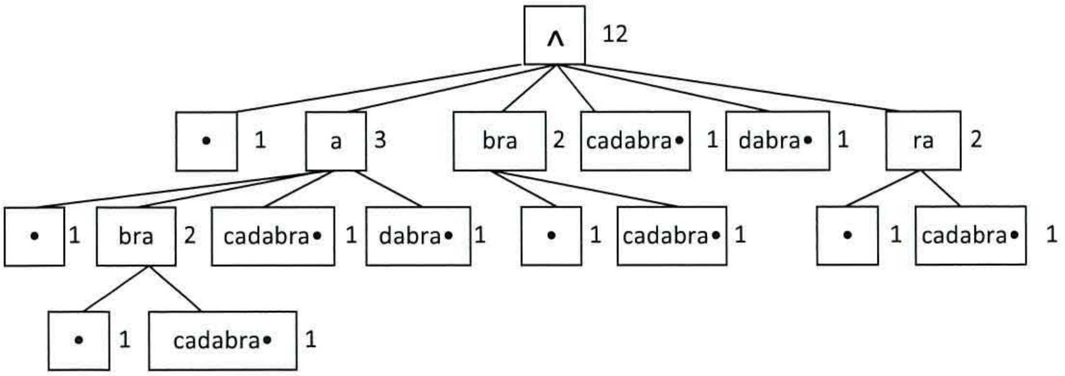


**Figure 4.3: Suffix tree representation of the string "abracadabra•".**

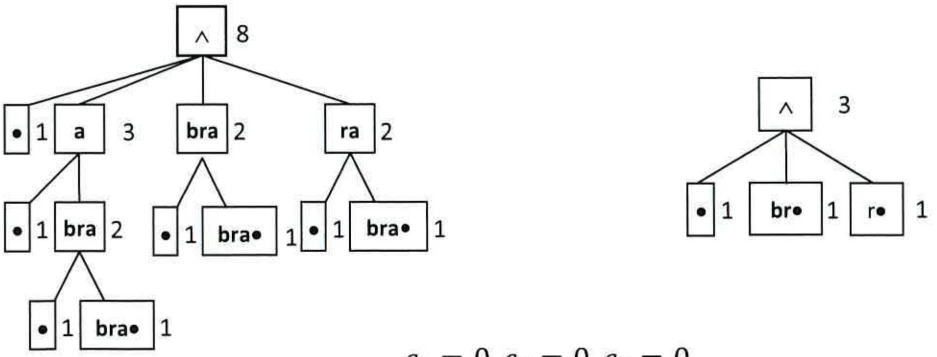
Concerning the four protocols, there is essentially no difference in the way the suffix trees are processed between the concatenated and non-concatenated protocols regardless of whether static or dynamic models are being used, apart from the size of the training text being used to prime the training suffix tree (i.e. the concatenation can be considered to be a simple pre-processing step done prior to the creation of the training suffix tree). For the $C$–Measure static case, the training and testing suffix trees are co-traversed, and counts of common nodes are accumulated with $C_1$ being the sum of counts of testing nodes at level 1 that match with training nodes, $C_2$ being the sum at level 2 and so on, as described above.

The dynamic protocols require dynamically updating the training suffix tree with information as either the testing suffix tree is being co-traversed (for frequency-based methods), or as the training plus testing text is processed sequentially (for entropy-based methods). For the frequency-based methods, the training suffix tree is dynamically updated in two ways: should a matching suffix be found, the counts of the nodes are incremented; and should a suffix contained within the testing tree not be found within the training tree, this new node is created and added. Unlike the static case, if during the traversal, a path within the testing tree is determined not to be common to the training tree, the traversal of this path will not now be abandoned. Instead, we continue traversing the path of the testing suffix as all uncommon nodes along a path are inserted into the training tree until the end of the path is reached.

Consider the training string $S$ = "abrabra•" and testing string $T$ = "xbrx•". The first suffix we would investigate is "xbrx•" and as this substring does not exist within the training suffix tree, the path would be dynamically inserted. When we then come to insert the suffix "x•" later on, rather than it again being ignored as with the static case, it will now match the character "x" at depth 1 as the substring "xbrx•" was inserted prior.

### 4.2.1 Static C-Measure

Consider the training string $S$ = "abrabra•" and testing string $T$ = "br•" where • denotes the end of file character. The series $c_1$, $c_2$ and $c_3$ (see 2.7.1) are all initialised as value zero and the value $c_k$ will increase as matching substrings of length $k$ are found. The following diagram displays both the training and testing tree once they have been created and no categorization has yet been performed.



$$c_1 = 0, c_2 = 0, c_3 = 0$$

**Figure 4.4: Suffix trees of training file "abrabra•" and test stream "br•".**

The following suffixes are contained within the testing string: “br•”, “r•” and “•”. In order to determine the similarity between the two strings, we need to determine the number of common substrings and we do this by simultaneously traversing both trees and determining how similar the two trees are.

As we are categorising the test string “br•” we shall be traversing each path within its suffix tree whilst simultaneously traversing the training tree to determine what nodes are common and at what depth. If during the traversal, a path within the training tree is determined not to be common to the training tree, the traversal of this path will be abandoned (not in dynamic case, with dynamic uncommon paths we keep traversing as all uncommon nodes along a path are inserted into the training tree until the end of the path is reached) and shall continue with the next until we have attempted to traverse each of the paths within the testing tree.

The first node to be found within the test suffix tree is node “•” at depth 1. The node “•” is found to be common within the training suffix tree. As the substring of length 1 was found to be common, the value $c_1$ is now increased by 1 and the $c_k$ counts are updated to the following: $c_1 = 1, c_2 = 0, c_3 = 0$.

The next node to be found within the test suffix tree is node “br•” and as we are now traversing a new node within a new path we are again at depth 1. We are therefore looking to find node “b” at depth 1 within the training suffix tree. We indeed find the node “bra” within the training suffix tree with the character “b” at depth 1 and the $c_k$ counts are updated to the following: $c_1 = 2, c_2 = 0, c_3 = 0$.

As we have matched the current character within the test node and have also not yet reached the end of this node, we shall remain within the testing node “br•” and now search for the character “r” which is at depth 2. Again the node is matched and so the counts are updated as follows: $c_1 = 2, c_2 = 1, c_3 = 0$.

Again we remain within the current testing node but we are now searching for “•” at depth 3 of the current path within the training suffix tree. As the node “•” was not found, the traversal along this path is abandoned and the counts remain unchanged.

We have reached the end of the current path and we now move on to the next and final node within the testing tree which is the node “r•”. We are first looking to find a node within the training tree which has the character “r” at depth 1, which we find in the form of the node “ra”. The common node the $c_k$ counts are updated to the following: $c_1 = 3, c_2 = 1, c_3 = 0$.

As we have found a matching node and have also not yet reached the end of the current path within the test suffix tree, we now look for the character “•” along path “r”.

A common node cannot be found and as we have now traversed all paths within the testing suffix tree we have completed our traversal and the final counts remain as follows: $c_1 = 3, c_2 = 1, c_3 = 0$.

### 4.2.2 Dynamic C-Measure

Again consider the training string $S$ = "abrabra•" and testing string $T$ = "br•" where • denotes the end of file character. The series $c_1, c_2$ and $c_3$ are again initialised as value zero and the value $c_k$ will again increase as matching substrings of length $k$ are found. As we are demonstrating an adaptive model therefore the training suffix tree will be dynamically updated in two ways. Should a matching suffix be found, the counts of the nodes shall increase and should a suffix contained within the testing tree not be found within the training tree, this new node will be created and added.

The suffix trees to begin with will be the same as the static case, see figure 4.4, with $c_1 = 0, c_2 = 0, c_3 = 0$ as no categorization has yet been performed. As we are now using a dynamic/adaptive model, if during the traversal, a path within the training tree is determined not to be common to the training tree, the traversal of this path will not now be abandoned. Instead, we continue traversing the path of the testing suffix as all uncommon nodes along a path are inserted into the training tree until the end of the path is reached.

The first node to be found within the test suffix tree is node "•" at depth 1. The node "•" is found to be common within the training suffix tree and as we are now using a dynamic/adaptive model, the count of this node within the training tree is increased from 1 to 2. Additionally, as the substring of length 1 was found to be common, the value of $c_1$ is now increased by 1.

The modified training suffix tree is shown here:

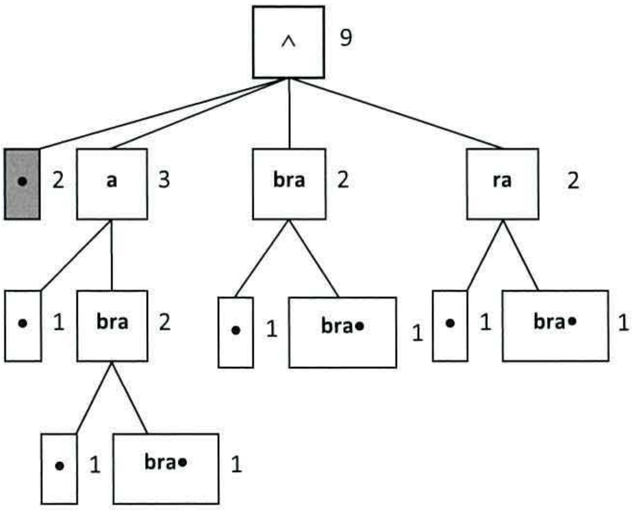


**Figure 4.5: Dynamic suffix tree of training file "abrabra•" once • has been processed.**

and the $c_k$ counts are updated to the following: $c_1 = 1, c_2 = 0, c_3 = 0$.

The next node to be found within the test suffix tree is node "br•" and as we are now traversing a new node within a new path we are again at depth 1. We are therefore looking to find node "b" at depth 1 within the training suffix tree. We indeed find the node "bra" within the training suffix tree with the character "b" at depth 1. It may be clear that other characters within this node are common between both trees and indeed we would continue until the end of the string before modifying the training suffix tree but for illustration purposes we shall break this into a number of steps so that the process remains clear.

The modified training suffix tree is shown here:

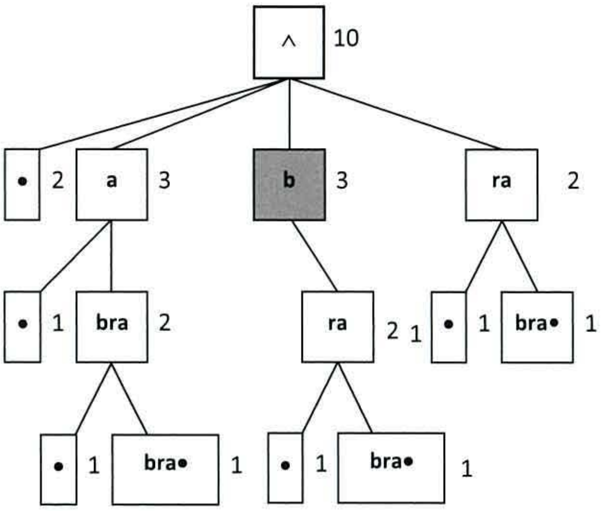


**Figure 4.6: Dynamic suffix tree of training file "abrabra•" once 'b' from within suffix br• has been processed.**

and the $c_k$ counts are updated to the following: $c_1 = 2, c_2 = 0, c_3 = 0$.

As we have matched the current character within the test node and have also not yet reached the end of this node, we shall remain within the testing node "br•" and now search for the character "r" which is at depth 2. Again the node is matched and so we update the training tree and counts accordingly.

The modified training suffix tree is shown here:

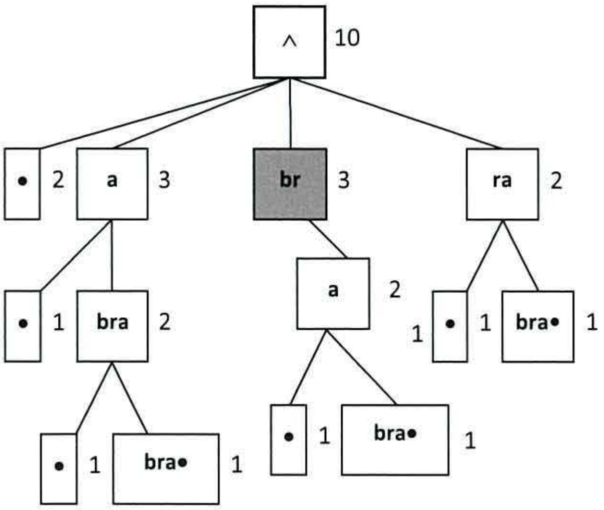


**Figure 4.7: Dynamic suffix tree of training file "abrabra•" once 'br' from within suffix br• has been processed.**

and the $c_k$ counts are updated to the following: $c_1 = 2, c_2 = 1, c_3 = 0$.

Again we remain within the current testing node but we are now searching for "•" at depth 3 of the current path within the training suffix tree. As the current training node "br" is non-branching, the only possible matching character is "a". Because of this, the node "a" shall remain a non-compressed node as it will remain to have a different count from its parent node "br".

As the node "•" was not found, this node at depth 3 shall now be inserted into the training suffix tree as a child of parent node "br". This is where the dynamic/adaptive model greatly differs from the static model as we are now dynamically altering the training tree to be more similar to the testing tree.

The modified training suffix tree is shown here:

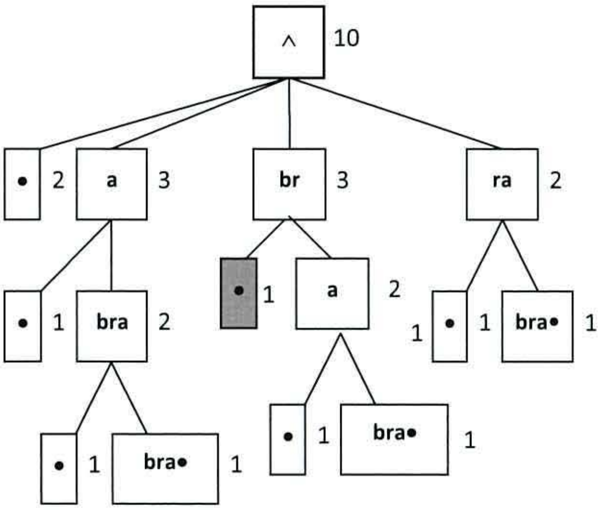


**Figure 4.8: Dynamic suffix tree of training file "abrabra•" once the suffix br• has been processed.**

and the $c_k$ counts remain unchanged as: $c_1 = 2, c_2 = 1, c_3 = 0$.

Again, as we have reached the end of the current path we now move on to the next and final node within the testing tree which is the node "r•". We are first looking to find a node within the training tree which has the character "r" at depth 1, which we find in the form of the node "ra".

As we find the common node we modify to the training suffix tree to be as follows:

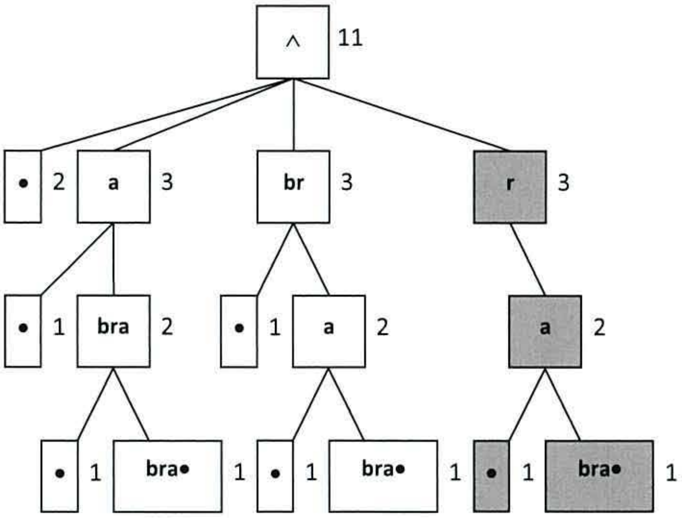


**Figure 4.9: Dynamic suffix tree of training file "abrabra•" once 'r' from within suffix r• has been processed.**

and the $c_k$ counts are updated to the following: $c_1 = 3, c_2 = 1, c_3 = 0$.

As we have found a matching node and have also not yet reached the end of the current path within the test suffix tree, we now look for the character "•" along path "r".

A common node cannot be found so we again insert this new node and the training tree is modified to the following:

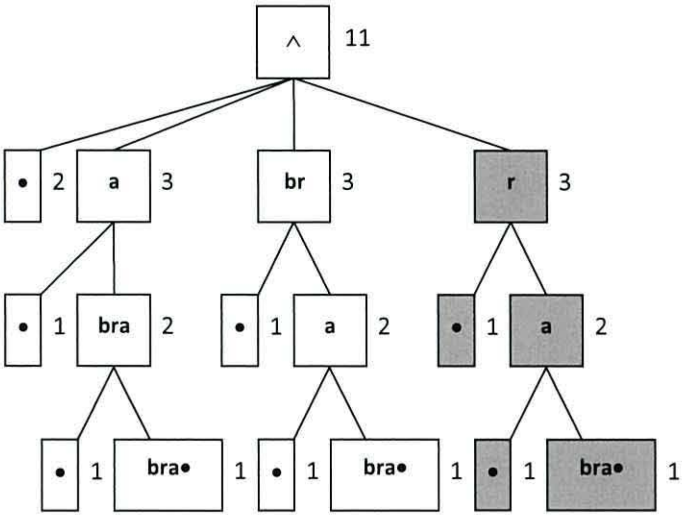


**Figure 4.10: Dynamic suffix tree of training file "abrabra•" once the suffix r• has been processed.**

As we have now traversed all paths within the testing suffix tree we have completed our traversal and the final counts are as follows: $c_1 = 3, c_2 = 1, c_3 = 0$. Using the example above you will notice that the counts are the same for both models. However, should an uncommon suffix be inserted, as it wasn't matched, and then be seen again, on all following occasions the substring would be matched as it was dynamically inserted.

As an example consider the training string $S$ = "abrabra•" and testing string $T$ = "xbrx•". The first suffix we would investigate is "xbrx•" and as this substring does not exist within the training suffix tree, the path would be dynamically inserted. Now when we then come to insert the suffix "x•", rather than it again being ignored as with the static case, it would now match the character "x" at depth 1 as the substring "xbrx•" was inserted prior.

The final training suffix tree once all comparisons have been made is displayed here:

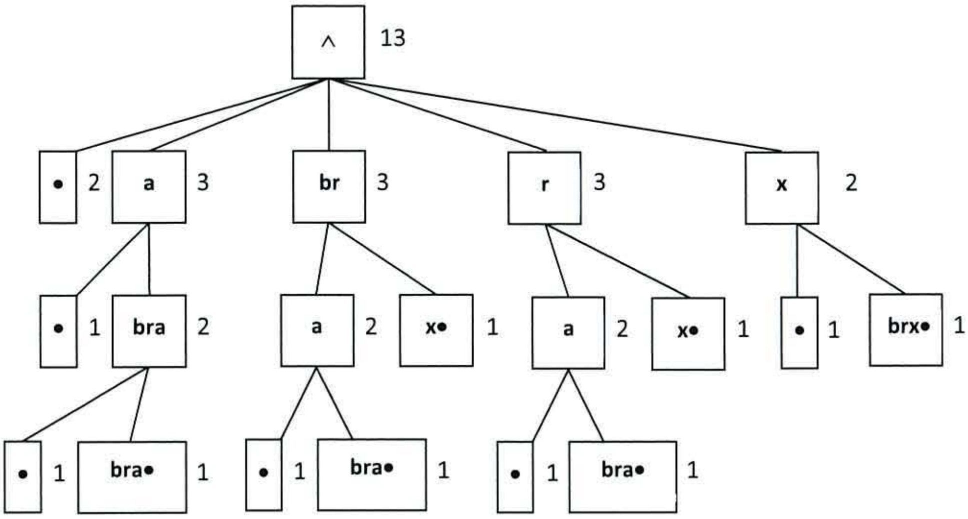


**Figure 4.11: Dynamic suffix tree of training file "abrabra•" once the testing stream xbrx• has been processed.**

### 4.2.3 PPM Without Full Exclusions

For table 3.1 the escape count is calculated as the number of known symbols at each point. For example if we had processed the symbol 'a' followed by 'b' we would be within position 'ab' at order 2 and the escape count would be 1 as 'r' is the only known symbol at this point.

| Order $k=2$ | | | Order $k=1$ | | | Order $k=0$ | | | Order $k=-1$ | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Predictions | $c$ | $p$ | Predictions | $c$ | $p$ | Predictions | $c$ | $p$ | Predictions | $c$ | $p$ |
| ab → r | 2 | $\frac{2}{3}$ | b → r | 2 | $\frac{2}{3}$ | → a | 5 | $\frac{5}{16}$ | → Λ | 1 | $\frac{1}{\|A\|}$ |
| → Esc | 1 | $\frac{1}{3}$ | → Esc | 1 | $\frac{1}{3}$ | → b | 2 | $\frac{2}{16}$ | | | |
| | | | | | | → c | 1 | $\frac{1}{16}$ | | | |
| | | | | | | → d | 1 | $\frac{1}{16}$ | | | |
| | | | | | | → r | 2 | $\frac{2}{16}$ | | | |
| | | | | | | → Esc | 5 | $\frac{5}{16}$ | | | |

**Table 4.1: List of pointers within context list after processing the symbols 'ab'.**

Referring to table 4.1, let us say that the next symbol to encode was indeed 'r', as the frequency of this symbol is 2, the probability of this symbol being encoded at this point is the frequency of the symbol divided by the sum of all frequencies (including the escape count) and in this case the probability would be 2/3.

However, if the symbol was unseen before at this point, i.e. the symbol 'i', then the probability would now be the escape count divided by the sum of all frequencies, i.e. 1/3. We would then escape to a lower context length, i.e. the pointer '^, b' and search for the symbol 'i' at this point. If the desired symbol existed at this point then the probability of finding the symbol at this context length would then be calculated with the probability being multiplied by 1/3, the probability that it was not found at the previous context length and then being found at the current order.

If the symbol continued to not be found (as is the case), we would then continue to escape down each order until either the symbol was found or we reached order -1. In the case of it not being found and us having to escape to order -1, the probabilities of escaping down each order would be multiplied by 1/256 (the number of symbols within the ASCII character set).

### 4.2.4 PPM With Full Exclusions

The difference in using full exclusions is that the counts of symbols at lower orders are affected if the symbol that have appeared at higher orders also appear at the current lower order after we have escaped. This then has an effect on the probability of the symbol being encoded. Referring to table 3.4, we were attempting to encode the symbol 'i' at order 2. The symbol 'i' was unseen at this point and so we escaped to order 1 with probability 1/3. At order 2 was the symbol 'r', which has now been seen and found to not match. Should this symbol now appear at order 1, it can be ignored as it has been seen previously at order 2.

As an example, let us say that we are instead presented with the contents of table 4.2. In the case where update exclusions are not used and with the previous example we have determined that "i" could not be encoded at order 2, having ruled out 'r', we could possibly be presented with the following at order 1:

| Order $k = 1$ | | | | |
|---|---|---|---|---|
| Predictions | | | $c$ | $p$ |
| b | → | a | 3 | $\frac{3}{11}$ |
| | → | i | 4 | $\frac{4}{11}$ |
| | → | r | 1 | $\frac{1}{11}$ |
| | → | Esc | 3 | $\frac{3}{11}$ |

**Table 4.2: Possible context list at order 1 without exclusions.**

In the case without exclusions, the probability of encoding the symbol 'i' at this point would be the frequency of the symbol 'i', which is 4, divided by the sum of all counts in addition to the escape count. The probability would therefore be:

$$\frac{4}{3 + 4 + 1 + 3} = 4/11.$$

This probability would then be multiplied by the probability of escaping from order 2, which was found to be 1/3. Now in the case of full exclusions, we have already ruled out the possibility of the symbol 'r' whilst at order 2, therefore the probability of encoding 'i' at this point with full exclusions would now be different. Symbols which have been seen at previous orders are now ignored leaving the following:

| Order $k = 1$ | | | | |
|---|---|---|---|---|
| Predictions | | | $c$ | $p$ |
| b | → | a | 3 | $\frac{3}{9}$ |
| | → | i | 4 | $\frac{4}{9}$ |
| | → | ~~r~~ | ~~1~~ | |
| | → | Esc | 2 | $\frac{2}{9}$ |

**Table 4.3: Possible context list at order 1 with full exclusions.**

Notice that there are now only two unseen symbols at this point as ‘r’ has been struck through, therefore the escape count is now reduced to 2. The probability of encoding at this point is now therefore:

$$\frac{4}{3+4+2} = 4/9.$$

This probability would then also be multiplied by the probability of us escaping from order 2, which was found to be 1/3.

### 4.2.5 Dynamic PPMC

Up to now only PPMC for static training models has been discussed. The models themselves can also be adaptive and this will allow us to compare the performance of PPMC across all four protocols, as we did with C-measure. As an example, let us consider the training string *S* = “abrabra•” and testing string *T* = “abrxbrx•” where • again denotes the end of file character. We would again create a suffix tree representation for *S* as shown in figure 4.4 but now rather than creating a suffix tree of *T*, we can simply treat it as a stream of symbols, processing each at a time.

After processing the symbols “a”, “b” and “r” we would be positioned at index 1 of the node “bra” which is a child of node “a” and we would also currently have a context length of 3. The next symbol to be encoded would be the symbol “x”, and as the only character seen at this point is the symbol “a”, we would need to calculate the escape probability.

Shown here is the current context lists showing the positions of each pointer into the suffix tree:

| Context Length | Pointer |
|---|---|
| 0 | ^ |
| 1 | ^r |
| 2 | ^br |
| 3 | ^abr |

**Table 4.4: Context list example after processing the string ‘abr’.**

As we are now using the adaptive protocol, we would insert the new symbol into the current position in the tree with count of 1, as well as also inserting this new node at the position of each pointer of lower context length. If the node already exists then the count of the node “x” would be increased by 1.

After inserting the symbol “x” at positions, “^abr”, “^br”, “r” and “^”, we now continue searching for the symbol “x” at context length 2. If we were using the static protocol then we know that the symbol “x” would not appear at this context length, nor at any other and we would then have to escape to context length 1 and continue processing from that point. However, as we have dynamically inserted the symbol “x” at position “^br”, we would now be presented with a different option. The symbol “a” is now no longer the only symbol to have been seen at this point, the node “x” now exists with frequency 1.

## Chapter Discussion

After introducing suffix trees as powerful data structures that allow fast searching we have shown that it is possible to compute the stream-based methods in reasonable time and space. It has also been shown that results for multiple algorithms, namely C-Measure and R-Measure can be calculated with only a single pass through the data structure. The next chapter details a toolkit that has been implemented to aid in the processing of the techniques detailed within this chapter.

# Chapter 5

# A Java based framework for implementing stream based models

## Chapter Summary

The purpose of this chapter is to detail an overview of the toolkit that has been created to aid in the calculation and comparison of the many different techniques discussed earlier. The toolkit has previously been described by Thomas and Teahan (2007), however, this chapter offers an overview of the class structure and also offer details on how the algorithms have been implemented; this is achieved through discussion, figures and code samples.

## Summary of each section

Section 5.1 displays a basic overview of the main components within the toolkit and gives examples of how simple the toolkit makes the process of executing experimentation. Section 5.2 discusses the process of preparing the corpora, including splitting the initial files and concatenating models and how the toolkit aids these processes. Section 5.2 also details the process of creating suffix trees through extracting suffixes and trimming the models. Section 5.3 offers more detailed information of the base classes and shows how they are extended through example code samples. Finally section 5.4 details the implementation of the algorithms and how their normalised values are calculated through code. Section 5.5 details the usage of the toolkit through the use of examples.

## 5.1 Overview

The motivation for creating the toolkit was to have a single application able to execute a number of different categorization algorithms at once and for us to be able to compare these results. This work is an important contribution to the field of text categorization and is an improvement over previous methods. Suffix trees have previously been used to implement PPM, however, they have not previously be used to implement C-Measure and R-Measure, and certainly not all together. The toolkit is extensible and offers common tools which are important if other algorithms are to be added and by having all experiments ran from a common toolkit on common corpora eliminates a number of the problems that currently exist when attempting to draw comparisons (Yang, 1999).

The toolkit was created using Java due to its platform independence and cost. Since Java is open source, it's completely free to develop and deploy applications with Java and its most popular IDE's are also free. In order to aid the process of adding algorithms and to keep a level of commonality a set of base classes was required from which each implementation could extend and make use of common functionality. A set of tools has also been created in order to prepare the corpora and also to be able to load suffix tree models from streams of text to be processed by the algorithms. These requirements are represented by the class structure in Figure 5.1 and it was from this that the toolkit was designed. For an exhaustive list of classes please refer to the API included within the attached DVD.

The main processes are contained within the base classes and these iterate through each testing model and process them against each of the training models. The base classes load each of the models and call a function to compare one against the other. The functions to compare the models are contained within the overriding classes as each are processed in a unique way. This model allows for extensibility as a new algorithm only requires the specified abstract functions to be implemented in order for it to function.

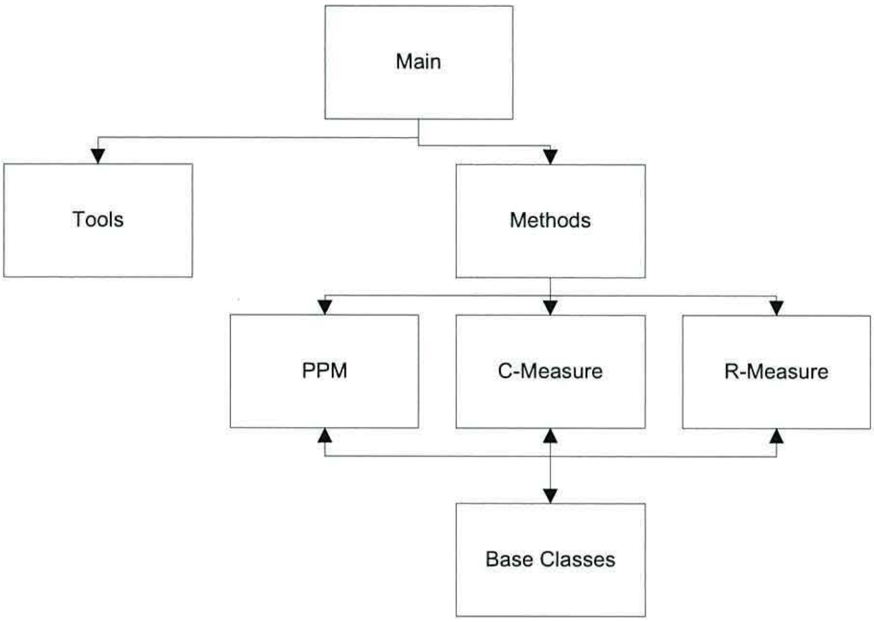


**Figure 5.1: High level overview of jSCat.**

## 5.2 Tools

This section describes some of the contents of the 'Tools' object within Figure 5.1. The next section describes how the toolkit can be used to split a corpus in order to perform cross validation. Section 5.2.2 details how the toolkit is able to concatenate files within specified training directories in order for us to investigate concatenated protocols. Section 5.2.3 describes how a suffix tree and nodes are represented within the toolkit and what properties are attributed to each. Section 5.2.4 explains the process of extracting suffixes from a text stream. Section 5.2.5 discusses some optimisations that have helped to construct and load the models in less time. Section 5.2.6 details the advantage of pruning the suffix tree to the maximum length that is required by the current experimentation. Section 5.2.7 discusses the toolkit's process of constructing the suffix tree and section 5.2.8 discusses how checking the counts of each node can help us to ensure that the tree has been correctly constructed.

### 5.2.1 Splitting the corpora

There are cases, as with Reuters-10 (mentioned in 2.5.2), where the corpus has already been preprocessed and the training data and testing data has been specified. In others cases such as Gutenberg and 20Newsgroups, this is not the case, and this step must be performed manually. In order to retrieve a fair result of how the algorithms have performed it is recommended to do cross validation. This process can introduce difficulties in recreating the setup of past experiments, as it is not often documented as to which documents were within which splits (see 2.5.7). Because of this, a listing of all documents within each split shall be included within the attached DVD.

In order to perform cross-validation the data is first split into a number of subsets, with either the same number of documents within each split, or as well as this, having the same number of documents from a class within each split also. Because the second method gives an even representation of each class within each split so this would be our preferred method. Once completed an output directory containing a folder for each split is outputted, as shown in Figure 5.2.

**Figure 5.2: Example output of split parent directories.**

And within each of these directories would be a directory for each of the categories, shown in Figure 5.3.

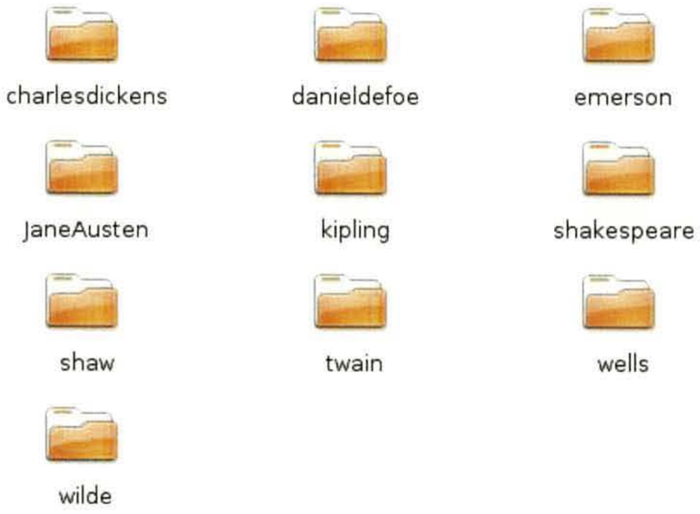


**Figure 5.3: Example directory listing found within each split.**

Each of the directories displayed in Figure 5.3 contain a subset of the original set of category files. It is important to note that if we had not ensured the files of each category were spread evenly across all splits, it is possible that as well as the category not holding an equal presence across each cross-validation stage. The Gutenberg corpus has only 40 works in total, it would therefore be possible that the category could not be present at all within some splits.

### 5.2.2 Concatenating categories

As stated earlier, it is not known whether or not concatenating the training data improves categorization performance and so it shall be investigated in order to determine its effectiveness, if any. The first thing to determine is the full location of the split directories and also the directory in which to output the concatenated models. Once these have been determined the concatenateAllFilesInDir method within the Concatenate class can be called. The concatenated files are added to a folder named Training within the root directory as shown in Figure 5.4. The concatenated files are given a unique filename equal to that of the category it represents, as shown in Figure 5.5.

**Figure 5.4: Output of the concatenated files parent directory.**

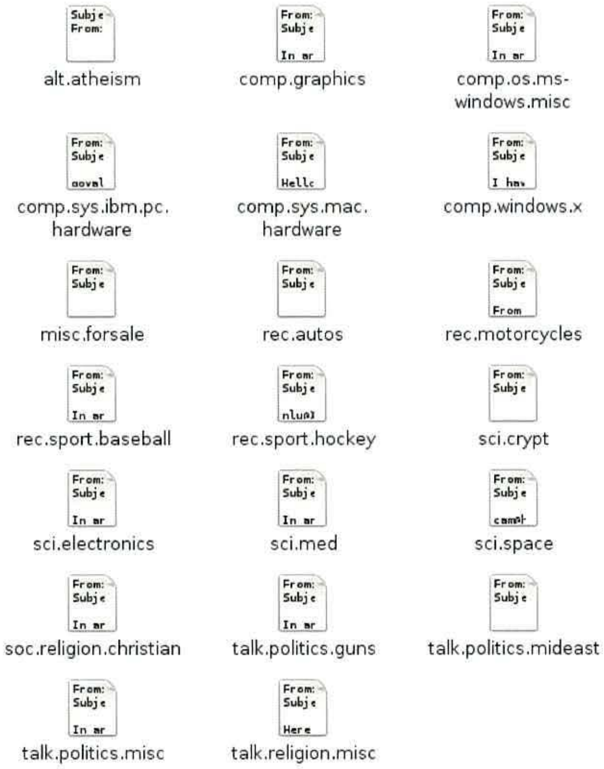


**Figure 5.5: Example output of concatenated training files.**

### 5.2.3 Suffix Tree representation

The construction of suffix trees in an efficient manner is a non-trivial problem and key to the success of the toolkit as a whole. This section describes the design chosen for the toolkit. An instance of the Node class (see Figure 5.6) represents a position within a suffix tree and holds information on its count, parent, child and so on. The RootNode class represents the root of the tree and holds additional information such as the filename of the stream it represents. OptimisedRootNode extends RootNode and as well as representing a suffix tree it is within this class that operations such as building the tree and trimming the depth of the tree and so on are contained.

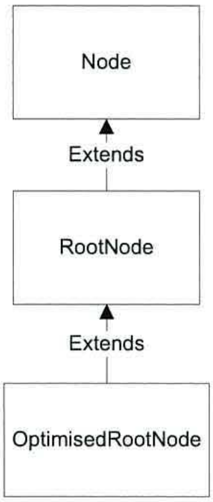


**Figure 5.6: Suffix tree representation classes.**

More details can be found about the classes in the attached DVD.

### 5.2.4 Extracting suffixes

The first stage of the method used to create a suffix tree is of course to load the contents of the file. The file may be a single stream of symbols or possibly a concatenated file, i.e. the output of the stage mentioned earlier. Irrespective of whether or not the file is concatenated, the next step is to locate the index of the eof symbols, the number of which is equal to the number of streams contained.

As an example, a concatenated file containing two streams may look like:

```
The cat sat on the mat.$The dog went out to play.$
```

This file contains two streams, denoted by two eof symbols $ whose indexes are then stored within a list. It is important to locate the indexes of the eof symbols, otherwise the concatenated file, a collection of several streams, would be treated as a single stream and this is not the case. Let's say that we have two indexes, i and j, whose values are the start index and end index respectively. i is set to be 0 in the first case, and then the index of the previous eof symbol +1 for each of the following streams. j is set to be the current eof index within the list. Quite simply i and j allow us to process each stream at a time i.e. “The cat sat on the mat.$” followed by “The dog went out to play.$”. The next step is to extract the suffixes from each of the streams and store each of these within a list. The list of suffixes is then sorted in order to speed up the process of building the tree from the list of suffixes.

### 5.2.5 Optimisation note

It is possible to create the tree without sorting the list of suffixes or without even listing the suffixes and simply adding them on the fly from the original text. However, by profiling the operation of building a tree, the most common task was found to be comparing two symbols, and it is required to do this in order to determine the location to insert the current Node. By sorting the list of Nodes, we are able to reduce the number of times this operation is executed as we know our current location within the suffix tree, always working from left to right and never having to return.

A further step which has allowed us to greatly reduce the time of creating the suffix tree is to not only store the location of the current Node within the tree that we are to add the next Node but also the location within that Node. As an example, we may have two suffixes within our sorted list, 'at on the mat.$' and 'at sat on the mat.$'. By having a sorted list of Nodes, we would not have to sort through the entire contents of the top level of the tree attempting to find character 'a' and then working from this point. We would simply have to check if the first character of the Node to be added was equal to the current symbol at depth 0, which happens to be 'a' and work from there. If the symbols were not equal then we would create a new branch at depth 0 and add the new Node.

What was found to take a long time even after this optimisation was that when a match was found at depth 0, we then had to traverse the tree, comparing symbols and finding the exact location to insert the new Node. A number of symbols would often need to be matched before we had to split and add the new Node. The case is often worse with the worst case of adding the same suffixes twice which is possible when adding numerous streams into a single suffix tree. This is why after sorting the list of suffixes, we then iterate through the list determining the common prefix between a Node and its previous. This allows us to determine the exact location within the current Node to insert the next without having to traverse the branch.

We know that we must store N suffixes, with N being the length of the text stream. However, it is essential to try and reduce the amount of information stored within each instance of a Node. We created a class named Node which extends the Java class DefaultMutableTreeNode as this allows us to store a reference to a parent Node and also to the child Nodes. It is important though to store enough information within this class to allow us to complete operations quickly, but we must also keep the amount of memory space used by each instance of the class, simply due to the number created when dealing with large streams.

One method of storing the contents of each Node would be to simply store the suffix that it represents as a String. Unfortunately due to the amount of suffixes and the sizes of each, this is not feasible. A better approach is to store the original stream, which we shall store as an array of characters within a class called RootNode which extends the Node class. Now, rather than storing the substring of each Node, we can simply store the index within the original string as the starting index of the substring and also the length of the substring. There is the added operation of retrieving the substring from the original stream, however, by storing the stream as an array it is a very quick operation and saves a massive amount of memory use.

We store the count of each Node within an int, and a reference to the RootNode which allows us to access the original text stream. We also store the depth of the Node as an int, as this allows us to easily set the depth of the next added Node and is also useful when computing e.g. C-measure as we need to determine at what length we are to increase the count. The final information stored is the number of common characters between this Node and the last to be added, as this allows for easy insertion when constructing the tree.

We now have enough information to quickly construct the tree as we have a sorted list of suffixes, represented as indexes into the original stream, the length of the suffix and also the number of common symbols between the current Node and the last to be added. A snapshot of this information would be similar to that shown within Table 5.1.

| **Index** | **Length** | **Common** |
|---|---|---|
| 55630 | 24113 | 4 |
| 492977 | 618 | 12 |
| 1318077 | 1278 | 7 |
| 569464 | 318 | 3 |

**Table 5.1: Example subset of suffix model information, from which we construct a suffix tree.**

Loading a 1.3MB concatenated file ten times as an example takes 45 seconds yet loading a standard file which is typically several Kb's ten times takes less than a second. This shows that loading non-concatenated files is done within an acceptable time but the concatenated files should ideally be improved, especially due to the fact that there will be several concatenated files to load (one for each class) and each will be loaded thousands of times. By storing the information displayed in Table 5.1 within a text file, each concatenated model can be loaded from the point at which we have the sorted list of nodes and the positions at which each is to be inserted. By using this method, the execution time required to load the above mentioned file can be nearly halved.

Another optimisation was found during the experimental stage of comparing the models to the testing files, and it was found that by changing the order of comparison we can make further improvements. The typical methodology would be to load a testing file and then compare this file against each of the training models. As mentioned in 3.4, assuming there are $K$ classes and $N$ testing documents, typically $N \gg K$ and the time to load a single model from $K$ is much greater than the time taken to load a model from within $N$. Each large model from within $K$ would typically be loaded $N$ number of times, however, this can be dramatically reduced by reordering the comparison and instead loading a training model only once and comparing the model against all testing files whilst it is in memory.

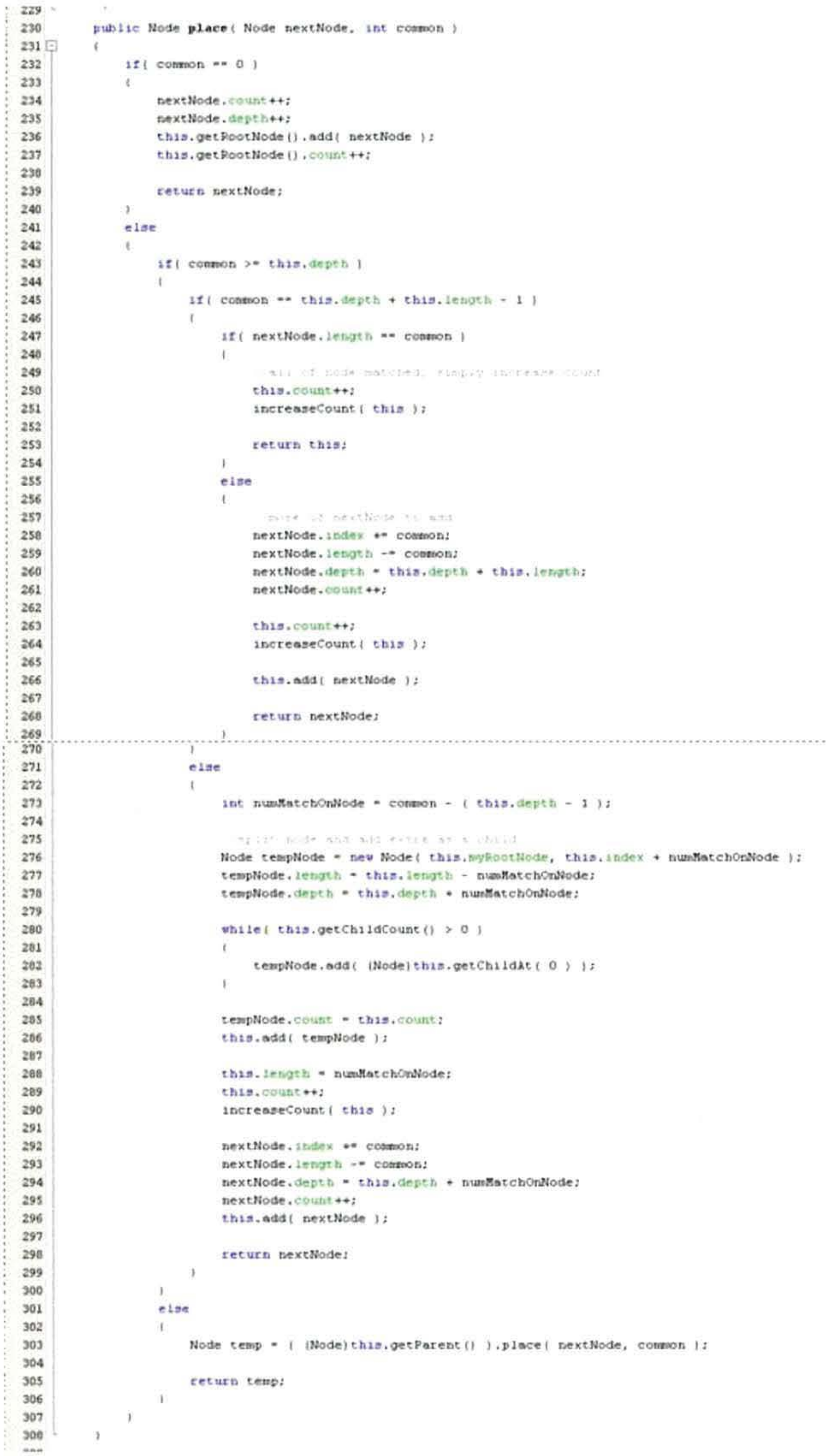
### 5.2.6 Trimming concatenated models

It is possible to further optimise time and space consumption by pruning the suffix trees to the maximum size required by the algorithms. When computing PPM Measures we only investigate up to a depth of 8 and with C-Measure we investigate up to depth 50, and we can prune the concatenated trees (due to their sheer size compared to non-concatenated files) respectively. The difference in lengths between the algorithms is due to the high computational overheads for high order models of PPM. This pruning has no outcome on the results but does serve to increase loading times of the trees and also the size of the trees when held in memory.

As a test experiment we executed the same experiment of loading a tree ten times, but in this case we used a very large 10MB file. It took 219 seconds to load the un-pruned tree ten times, 168 seconds when the tree was pruned to a depth of 50, and 81 seconds when the tree was pruned to a depth of 8.

### 5.2.7 Building the tree

Code sample 5.1 was particularly difficult and is used to insert a new node into the tree using common character substring lengths mentioned in 5.2.5. The method is not static and is therefore called upon a current instance of the Node class, which will always be the last Node added to the tree i.e. lastnode.place(newnode, number of common characters). We therefore have the location of the last node to be added, the information pertaining to the new node and also the number of common symbols between each of the suffixes.



```
229
230     public Node place( Node nextNode, int common )
231     {
232         if( common == 0 )
233         {
234             nextNode.count++;
235             nextNode.depth++;
236             this.getRootNode().add( nextNode );
237             this.getRootNode().count++;
238
239             return nextNode;
240         }
241         else
242         {
243             if( common >= this.depth )
244             {
245                 if( common == this.depth + this.length - 1 )
246                 {
247                     if( nextNode.length == common )
248                     {
249                         // all of node matched, simply increase count
250                         this.count++;
251                         increaseCount( this );
252
253                         return this;
254                     }
255                     else
256                     {
257                         // rest of nextNode to add
258                         nextNode.index += common;
259                         nextNode.length -= common;
260                         nextNode.depth = this.depth + this.length;
261                         nextNode.count++;
262
263                         this.count++;
264                         increaseCount( this );
265
266                         this.add( nextNode );
267
268                         return nextNode;
269                     }
270                 }
271                 else
272                 {
273                     int numMatchOnNode = common - ( this.depth - 1 );
274
275                     // split node and add extra as a child
276                     Node tempNode = new Node( this.myRootNode, this.index + numMatchOnNode );
277                     tempNode.length = this.length - numMatchOnNode;
278                     tempNode.depth = this.depth + numMatchOnNode;
279
280                     while( this.getChildCount() > 0 )
281                     {
282                         tempNode.add( (Node)this.getChildAt( 0 ) );
283                     }
284
285                     tempNode.count = this.count;
286                     this.add( tempNode );
287
288                     this.length = numMatchOnNode;
289                     this.count++;
290                     increaseCount( this );
291
292                     nextNode.index += common;
293                     nextNode.length -= common;
294                     nextNode.depth = this.depth + numMatchOnNode;
295                     nextNode.count++;
296                     this.add( nextNode );
297
298                     return nextNode;
299                 }
300             }
301             else
302             {
303                 Node temp = ( (Node)this.getParent() ).place( nextNode, common );
304
305                 return temp;
306             }
307         }
308     }
```

**Code Sample 5.1: Inserting the next node into our tree.**

The first thing checked is whether or not the common value is set to 0 (line 232), if so then the first symbol of the next suffix does not match the previous and so the new branch/node is added as a child to the root of the tree. The count is set to be 1, its depth is set to be 1 and in order to balance the counts within the tree correctly, the count of the root node of the tree is also increased by 1 (see lines 234-237). If, however, the common value is greater than 1, we have to add the new suffix into the last branch added to the tree. We do not know where within the branch though as so we check as to whether or not the common value is greater than or equal to the depth of the current Node within the branch (line 243). If not then we call the place method again but this time we call it on the parent node and not the current one (line 303). We are basically traversing up towards the root of the branch until the insertion location is found.

If the common value is greater than or equal to the depth of the current node then we must make further checks. If the length of the node to be added is equal to its number of common characters then it is a direct duplicate of the current node (247), we therefore increase the count of the current node and each of its parent's nodes until the root node is reached. To do this we use an iterative method which continues to increase the count of the current nodes parent node until the current node becomes the root node of our tree.

Within lines 255-269, if the length of the new node is greater than the number of common symbols then the next suffix does match the last to be added but there are more symbols i.e. it is longer. What we therefore have to do is effectively create a new node to be placed as a child of the last to be added. We change its length to be its current suffix length minus the length of the last added suffix, we set its depth to be the suffix length of the last node to be added i.e. the depth of the last node plus its length. Its index is also incremented by the number of common symbols and its count is set to be 1. The count of the last node to be added and all of its parent nodes are incremented by 1 and then the new node is placed as a child node.

In the case that the number of common symbols is less than the depth of the last added node in addition to its length then we must break up the last added node and create two children, one will be the part of the suffix following the break, and the other will be to represent the newly added suffix (lines 271-299).

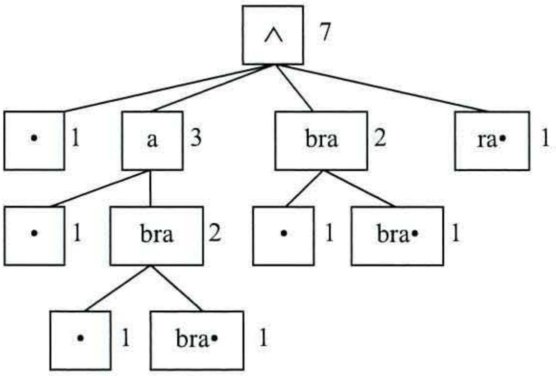


**Figure 5.7: Original tree before adding node which matches all characters within the current node.**

Say that we are currently located at the highlighted node "ra$" as displayed in Figure 5.7. If we were to then insert the stream "rabra$" then "ra" would be common to both suffixes, however, we would split the "ra$" node so that its suffix becomes common i.e. "ra" and its previous remainder

"$" is added as a child node with the same count but its length, depth and index updated accordingly. We are then able to add what was not common between the two suffixes as a new child node, which in this case is "bra$". All counts of parent nodes are then updated and the insertion is complete, see Figure 5.8.

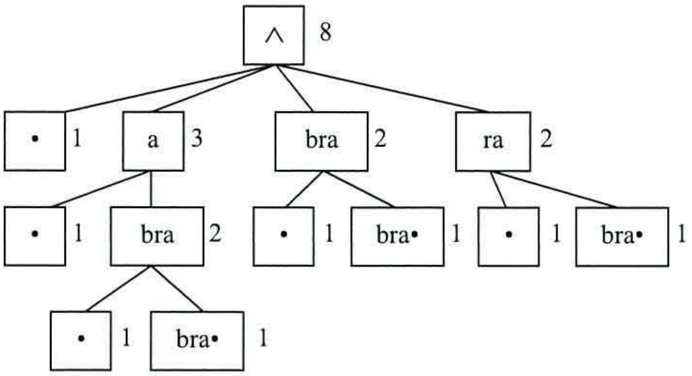


**Figure 5.8: Tree shown in 4.10 after inserted the next node.**

### 5.2.8 Checking the counts within the suffix tree

Once the tree has been built it is possible to check that the counts are correct by iterating through every non-leaf node (node which has children) and ensuring that the count of the parent node is equal to the sum of the counts of its children. If at any time this is not true, then the tree has not been constructed correctly. This is because the count at the root of the tree is equal to the number of suffixes and this number should equal the total number of leaf nodes within the tree.

## 5.3 Base classes

The relationship between the base classes and the other components within the toolkit is shown in Figure 5.1 and was introduced in section 5.1 as allowing extensibility as a new algorithm only requires the specified abstract functions to be implemented in order for it to function.

Results are stored within a combination of comparisons and collections. Comparison is an abstract class that is used to store information regarding a single comparison between a testing file and a training file. We say an instance of the TestCollection class holds all comparative values relating to a single testing file. Figure 5.9 shows that for each algorithm we extend TestCollection to all comparative values relating to its own technique. Further information regarding TestCollection and extending the class can be found in 5.3.2. Collection as a base class is used to hold an array of TestCollections, a list of training files, testing files and also the current protocol. Further information regarding the Collection class and how it is further extended for each algorithm can be found in 5.3.4.

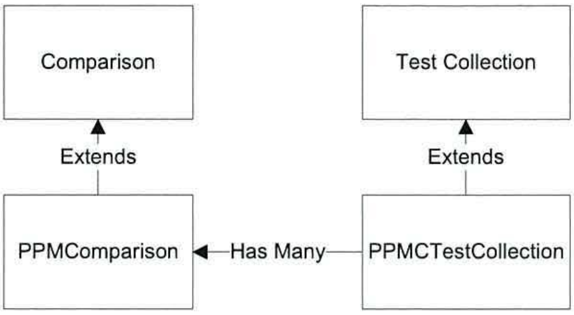


**Figure 5.9: Example extension of the base classes.**

### 5.3.1 Comparison class

Comparison.java is an abstract class that is used to store information regarding a single comparison between a testing file and a training file/model. As each testing file is compared against a number of training files we would say that each instance of a testing file would have a number of comparisons i.e. a one to many relationship. Comparison.java stores the training file used for the instance of a comparison and returns basic information such as the location of the training file and the category to which the training file belongs. The class also contains an abstract method named getNumValuesPerResult and this is needed as the algorithms may have differing amount of results per comparison. The method therefore returns the number of values outputted from a single comparison i.e. C-Measure outputs C-Counts for each matching substring length, however, PPM outputs only a single comparative value.

This class can now be extended by each instance of an algorithm to store results pertaining to a comparison. C-Measure will generate a number of counts, and a result is outputted for each length of substring compared. In this case an array of integers is used to store the results and methods are included to fill the contents of this array as well as retrieving them to a calling method.

Because the base class Comparison.java is extended, the information which is common to all comparisons, i.e. information regarding the training file used can be passed to the base class by making use of Java's keyword 'super' (used to call the constructor of the superclass in the base class). As mentioned earlier, each class which extends Comparison.java must also implement the method getNumValuesPerResult and in this case the size of the array cCounts would be returned.

### 5.3.2 Test Collection class

We say a TestCollection holds all comparative values relating to a single testing file. The TestCollection class is therefore used to hold information regarding the testing file and also all comparisons (instances of Comparison.java) which have been created by comparing each training model against this testing stream. The Boolean isConcatenated is needed in order to determine the category of the training model as a concatenated training models category would be set as its file name but a non-concatenated training model would be held in a folder with the name of the category to which it belongs. The testing file is stored within a variable of the class instance and with the information contained within this class and each comparison we now know the training file and testing file involved in each comparison. The length of the testing file is required a number of times when calculating measures and as it takes time to compute it is more efficient to have this value stored within a variable also.

TestCollection is also an abstract class containing a number of abstract methods and each algorithm must extend this class as each algorithm will have its own method of creating new comparisons and also retrieving them. Many functions will be common to all algorithms and that is why this abstract class has been created. When an instance of the TestCollection class is created, all algorithms will need to specify the testing file and whether or not the testing file will be compared to a concatenated training model or not and that is why the methods relating to the setting of this information is contained within this class. The retrieval of this information as well as the category to which the testing file belongs will also be common and is again contained within this class.

As the constructor for this class specifies that an array containing all training files must be supplied, and though the type of comparisons and the array type each will be held in are of different class types, each will be done in much the same way and that is why the ordering of the calls to the abstract methods is also held within this class file. We say that the closest matching training model is the one that outputs the highest comparative value when compared against a testing file.

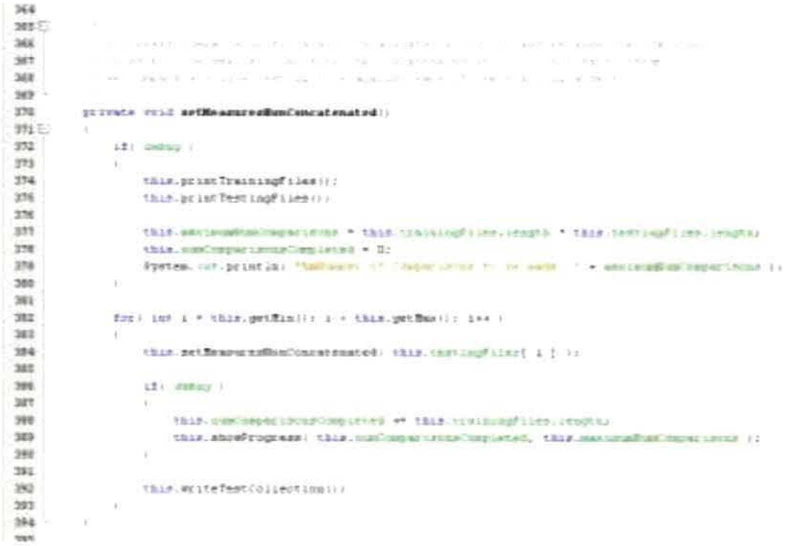
### 5.3.3 Extending TestCollection class

Take PPM and Figure 5.9 as an example of how each algorithm would extend the base class. The constructors are very simple and this is the intention of using inheritance within the code. The call to 'super' is made which calls the constructor of the base class, which as we saw will handle the setting of the testing file and then call abstract methods which are contained within the classes that extend it. Two of the abstract methods contained were createComparisonArray and createNewComparison, which handled the creation of the type of comparisons to be instantiated, in this case PPMComparison. Each algorithm will implement these methods in similar ways except they shall substitute PPMComparison for its own type, possibly CComparison for C-Measure. The array of comparison types are now stored within this class so that all comparisons for a TestCollection are easily accessible. The comparative values will also be set from methods within this class as it is from this class that we are able to access all of the comparisons but each algorithm may have its own way in which it sets the values and also what the methods are called.

### 5.3.4 Collection class

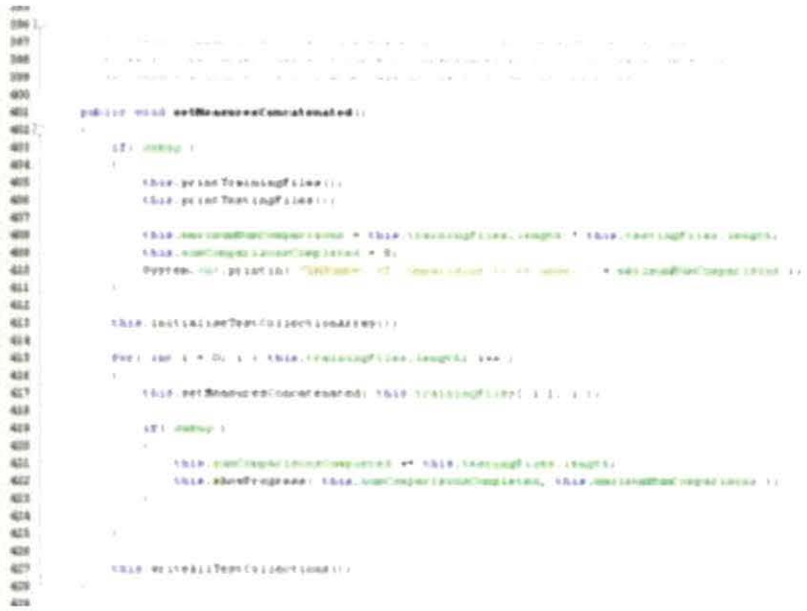
The Collection class as a base class is used to hold an array of TestCollections as well as the set up information such as a list of all training files, testing files and also the current protocol. The most important method within the class is setMeasures as this is the method which starts the experimental process once all of the initial setup has been completed. The method will determine whether the current protocol is concatenated or not and call the relevant method in each case.

Once called, each functions in much the same way. Both are able to print out useful debugging information such as information on each of the files being processed and the current progress of the comparisons. In the case of non-concatenated, each testing file is accessed in turn and passed to the method setMeasuresNonConcatenated which requires a testing file as a parameter, and these methods are located within the classes that extend this one, of which each algorithm must have. Within this class the methods are abstract and the implementation of these methods shall be discussed later. As mentioned earlier, it is more efficient in the case of concatenated training models to load them a minimum number of times and that is why is the case of setMeasuresConcatenated each trianing model is processed individually rather than each testing file. These differences can be seen within lines 382 and 392 in Code sample 5.2, 415 and 425 in Code sample 5.3. The algorithm specific implementation for non-concatenated protocols is called by line 384 in Code sample 5.2, and line 417 for concatenated protocols.



```
364
365     /**
366      [illegible]
367      [illegible]
368      [illegible]
369      */
370     private void setMeasuresNonConcatenated()
371     {
372         if( debug )
373         {
374             this.printTrainingFiles();
375             this.printTestingFiles();
376
377             this.maximumNumComparisons = this.trainingFiles.length * this.testingFiles.length;
378             this.numComparisonsCompleted = 0;
379             System.out.println( "[illegible]" + maximumNumComparisons );
380         }
381
382         for( int i = this.getMin(); i < this.getMax(); i++ )
383         {
384             this.setMeasuresNonConcatenated( this.testingFiles[ i ] );
385
386             if( debug )
387             {
388                 this.numComparisonsCompleted += this.trainingFiles.length;
389                 this.showProgress( this.numComparisonsCompleted, this.maximumNumComparisons );
390             }
391
392             this.writeTestCollection();
393         }
394     }
395
```

**Code Sample 5.2: Base processing of non-concatenated comparisons.**



```
395
396     /**
397      [illegible]
398      [illegible]
399      [illegible]
400      */
401     public void setMeasuresConcatenated()
402     {
403         if( debug )
404         {
405             this.printTrainingFiles();
406             this.printTestingFiles();
407
408             this.maximumNumComparisons = this.trainingFiles.length * this.testingFiles.length;
409             this.numComparisonsCompleted = 0;
410             System.out.println( "[illegible]" + maximumNumComparisons );
411         }
412
413         this.initialiseTestCollectionArrays();
414
415         for( int i = 0; i < this.trainingFiles.length; i++ )
416         {
417             this.setMeasuresConcatenated( this.trainingFiles[ i ], i );
418
419             if( debug )
420             {
421                 this.numComparisonsCompleted += this.testingFiles.length;
422                 this.showProgress( this.numComparisonsCompleted, this.maximumNumComparisons );
423             }
424
425         }
426
427         this.writeAllTestCollections();
428     }
429
```

**Code Sample 5.3: Base processing of concatenated comparisons.**

The general purpose of the methods contained within the Collection class is to fill a multidirectional array of results. After each testing file has been compared against all of the training models, the results of the comparisons are outputted to a text file so that the results can be stored for repeated viewing without having to re-run the experiments.

## 5.4 Implementation of the algorithms

### 5.4.1 C-Measure

### 5.4.1.1 Static case

The method setCounts was particularly difficult and so shall be explained in depth within this section. It is a recursive method that tests whether the current symbol we are processing within the testing suffix tree matches the current symbol within the training suffix tree. If so, then the C-Counts are updated for the current depth of the substring, if not then we move on to the next symbol. Both the training tree and testing tree are traversed simultaneously and shall continue until we have checked all paths within the testing tree or the end of the training suffix tree is reached.

The setCounts method was built after identifying all possible cases when simultaneously traversing two trees. The first condition within the method tests whether this is the first call i.e. we are at the root of the tree. The route of the tree holds no characters and is not to be compared against the route of the training tree, this condition allows us to gather each of the testing trees children of the root node and iterate through them sequentially. Both trees are sorted and so when we are searching for an insertion position for the current testing node, if this value is equal to the number of children then this tells us that none of the remainder will match and so we return. Until this condition is met, we recursively call the setCounts method but replace the root node with the current child of the root node. We are not yet concerned with whether or not the first characters match as this will be dealt with at the next stage.

When the method is recursively called, we have five essential parameters as displayed in table 5.2.

| Name | Type | Description |
|---|---|---|
| test | Node | Current Node within testing tree |
| testOffset | int | Position within current testing node i.e. current testing character |
| train | Node | Current Node within testing tree |
| trainOffset | int | Position within current training node i.e. current training character |
| currentLength | int | Length of substring and position within array in which we increase count |

**Table 5.2: Parameter information for C-Measure setCounts method.**

What we are essentially doing is keeping track of our positions within each of the trees and comparing the characters, continuing to traverse whilst they are matching and returning when they do not, and then moving onto the next testing branch. There are six possible cases when you are asked to compare the next characters within the current nodes:

**Case 1:** We have reached the end of the current testing node and the current testing node has no children. In this case we have no need to continue as we have matched the entire current match and so we return.

**Case 2:** We have reached the end of the training node and the current training node has no children. In this case, although we would like to continue, the training branch has no further paths i.e. this part of the current suffix is unseen within the training text and so we return.

**Case 3:** We have reached the end of the testing node but not the current training node, however, the testing node does have child nodes. In this case we remain at the same position within the training tree but we now iterate through the children of the testing node to see if the following symbol within the training node exists. There is no need to iterate through all of the testing children and attempting to compare these with the training tree as we are within a node and there is only one possibility so it is quicker to attempt to find this within the list of testing children. If a matching character is found, we then continue by making the matching testing node the current test node. If no match is found we return to the calling method.

**Case 4:** We have reached the end of the training node but not the current testing node, however, the training node does have child nodes. In this case we remain at the same position within the testing tree but we now iterate through the children of the training node to see if the following symbol within the testing node exists. If a matching character is found, we then continue by making the matching training node the current node. If no match is found we return to the calling method.

**Case 5:** We have reached the end of both the current testing and training node, and both of these nodes have child nodes. This case involves more processing than the other cases as we now need to iterate through each of the child nodes and recursively process each against each of the training nodes and their children.

**Case 6:** If none of the above conditions are satisfied then we continue to shift positions along both the current nodes, updating the counts array as we progress. This loop will then continue until we reach the end of either of the nodes or we find a symbol which does not match.

#### 5.4.1.2 Dynamic case

The dynamic case is processed differently as we do not build a suffix tree. We do create every node which would be contained within the tree but these nodes are kept within a list and not added to a tree. The reason stems from the fact that the symbols are not actually stored within nodes, we instead have a reference to the original input string. With the dynamic case it is very likely that we will be inserting suffixes that are not contained within this input string and so it must change. Also when a node is added or modified, it is also very likely that indexes would change and we would need to know which input string the index refers to. To tackle this, it works well to concatenate the testing string onto the end of the training string (see 5.2.4) and treat the index of the first symbol of the testing string as N, with N being the length of the training string and us beginning at value 0. This would now ensure that there is no confusion between the reference location of a suffix.

```
Training String: The cat sat on the mat.

Testing String: The dog went out to play.

The cat sat on the mat.$The dog went out to play.$

Index of first testing character is 24.
```

**Figure 5.10: Example of testing string being concatenated onto training string for dynamic cases.**

We would then begin extracting suffixes from the testing string and compare these to the training tree. Using the above example we would begin with the suffix found between index 24 and 49 which is effectively the entire testing stream and then shift right one position each time until we reach the end of the testing stream. The suffixes are extracted and created as Nodes through use of the method insertSuffixes which passes each node to a dynamicInsert function within the same class. The method dynamicInsert is built logically in much the same way as setcCounts. If the current training node has no children and is the RootNode of the tree then we increase the count at the root and also the count of the new node to be inserted and then insert the node as a child of the RootNode. If the current training node has no children but is not the root there is no need to split the node or add as a child, we simply increase the length of the node by one and alter the index so that is refers to the position within the testing stream rather than the training stream.

If the current training node does have children then we must find where within the current depth to insert the new testing suffix. We do a binary search of the children and the insertion position is returned as an integer. If this value is equal to the number of children at this depth then the ASCII value of the first symbol is greater than any of the children within this depth. The new node is therefore inserted as the last child due to the ordered nature of the suffix tree. The counts are adjusted accordingly and the depth is calculated as the depth of the current node in addition to its length.

If the value returned from the binary search is not equal to the number of children we must then treat the value as the desired insertion position. The next step is to determine whether the node that is currently situated at this position needs to be shifted to the right (as the tree is ordered) or at least some of the current node is matched and so we must insert the new node into the current node and possibly split it at some point.

In the case where the first symbol of the node that exists at the insertion point is equal to that of the new node to be inserted, this is the time that we would now increase the counts within the C-Counts array as a match has been found. We would then loop, moving along both nodes and increasing the C-Counts at the relevant depth until we reach the end of either node or the next symbols are found to not match. If all characters within the new node are matched then we simply increase the count of the current node and return to the calling method. If there exists more symbols on the new node then the new node must be dynamically inserted as a child to the current node and so the dynamicInsert method is called with the current training node which we have reached the end of as the node at which we want to insert and the new node's index and length are altered to support the fact that some of the symbols have already been matched before the remainder is passed as the new node parameter. If the current node within the training tree was a leaf node then it is this case that makes the suffix tree lose its balance of counts, i.e. counts of the parent node being equal to the sum of the counts of its children. This is an example of where the eof symbol is important, it ensures that there is no case where all symbols of a lead node can be matched with the testing node still having more characters.

If we have not reached the end of either the current training or testing node then we must split the current training node. The description of dynamic C-Measure in 3.3.2 shows an example where we must split the node “bra” within the training tree as “br$” is inserted. You will see that the node “bra” is modified to become “br” and the removed “a” is created as a child node with all previous children of the “bra” node now becoming children of the node “a”. This function was again particularly difficult to implement and can be seen within the source code on the attached DVD.

### 5.4.2 R-Measure

All R-Measure results can be calculated using C-Measure results and this is the approach used within the toolkit. Rather than calculating R-Measure results independently or simultaneously with C-Measure, the toolkit loads the values of C-Measure comparisons and places them within RTestCollection's, which are extensions of TestCollection class. The base class Accuracy calls the method findResults and when this is overridden within the Raccuracy class, any R-Measure variances can then be calculated by adding calls to find accuracies for each variant in which we are interested.

#### 5.4.2.1 $r^{max}$

$r^{max}$ is an alternative name for the standard $R$-Measure and is defined as the sum of the C-Counts and so we gather the value from getrCount. The value is normalised by adhering to the formula displayed in 3.1 and its coded equivalent is displayed in Code sample 5.4.

**Code Sample 5.4: Coded Normalised R-Measure Value.**

#### 5.4.2.2 $R_{\leq q}$

$R_{\leq q}$ is easily obtained by summing all C-Counts found between 1 and the set limit and we explore all maximum values between 1 and a given maximum in order to determine which maximum value achieves the greatest results. It is the responsibility of getRMeasureLessThanEqualTo to summate the C-Counts between 1 and the upper limit and getNormalisedMeasure shall then normalise this value by implementing the formula displayed in 3.1.

#### 5.4.2.3 $R_{\geq q}$

The $R_{\geq q}$-Measure is very similar to $R_{\leq q}$ except that we decrease towards 1 as an upper limit rather than increasing from 1 as a lower limit and in this case it is the upper limit which changes and the lower limit 1 remains static. Again a total is determined by an alternative method, namely getRMeasureGreaterThanEqualTo but this figure is then normalised as discussed in 3.1.

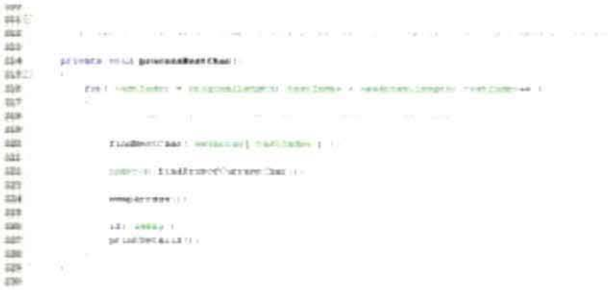
### 5.4.2.4 R-Ranges

The function findRRangesAccuracy makes use of two loops in order to accumulate C-Counts within the ranges. There is no need to normalise the total and so we determine the comparison that returns the highest total between the set ranges to contain the correct author, topic or type and so on depending on the current situation.

## 5.4.3 PPM (Prediction By Partial Matching)

As with the dynamic case of C-Measure we concatenate the testing stream to the training stream as this allows us to work with unique indexes/positions within the stream. We then create two arrays, one of which acts as a temporary store which holds pointers that are updated, the other acts as the context list once all pointers have finished updating. Once all updates have finished the contents of the updating pointers are transferred to the second array.

The method processNextChar (see Code sample 5.5) holds the outer loop operation and its purpose is to attempt to encode all symbols within the testing stream until all have been processed and this is done in three steps. Whilst there are still unprocessed testing symbols we first fetch the next symbol to be encoded, we then calculate the probability for this current symbol and we then swap the contents of the arrays i.e. pointers before processing the next symbol.



```
private void processNextChar()
{
    for ( [illegible] )
    {
        [illegible]

        findNextChar( [illegible] );

        [illegible]

        swapArrays();

        [illegible]
        printDetails();
    }
}
```

**Code Sample 5.5: Coded method for encoding all symbols for PPM.**

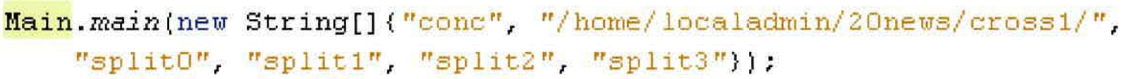
There is a great amount of work involved in calculating the probability of the current symbol being encoded. It is the method findNextChar  which attempts to encode the next symbol at the furthest point in the context list, and drops down this list should it be unsuccessful at the current depth. As we escape down through context lengths we have to combine probabilities and so we store all of the probabilities to be combined within a list and this list is first of all reset along with the list of exclusions (should there be any) when we process a new symbol.

As symbols are found we simply update pointers within the suffix tree and update the contents of the probability list and this process continues until we reach the end of the testing stream or the current symbol cannot be encoded. If the symbol cannot be encoded we decrease the context length and continue but as well as this, with the dynamic case, should there be an unseen symbol then the symbol must be inserted at all current positions within the tree and this is made easy due to us having stored each of the insertion locations within our array.

## 5.5 Using the toolkit

The object named 'Main' in Figure 5.1 is the main entry point of the application and allows a level of abstraction between the user and the underlying methodology. It is from here that user commands are executed and these commands perform underlying operations. A user must first state the operation they wish to perform, such as "conc" for concatenating files within a location by also passing parameters indicating folder names within a base location, see Example 1.

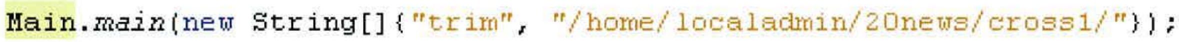
**Example 1**



```
Main.main(new String[]{"conc", "/home/localadmin/20news/cross1/",
    "split0", "split1", "split2", "split3"});
```

Example 2 shows that in order to then trim these models for optimisation again only a single call is required no matter what algorithm(s) are to be used.

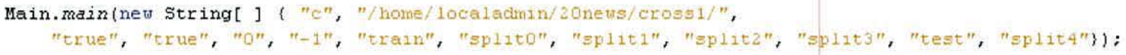
**Example 2**



```
Main.main(new String[]{"trim", "/home/localadmin/20news/cross1/"});
```

Example 3 shows how little code is needed in order to process the models and then perform a C-Measure calculation on them. First of all the parameter "c" is passed that indicated it is C-Measure we wish to be performed. We then pass the base location of the corpora from which we can find the training and testing documents. The next two parameters ("true" "true" in Example 3) indicate the protocol to be investigated, with the first parameter being a Boolean value, true indicating concatenated and the static for static or dynamic with true indicating static. Following this is the index of the testing files to be investigated, "0" indicates that we must start with the first file at index zero but "-1" is used to indicate all files, if say "99" were passed then only the first 100 testing documents would be categorized, from index 0 to 99. The following parameters indicate the names of the training directories and then the testing directory.

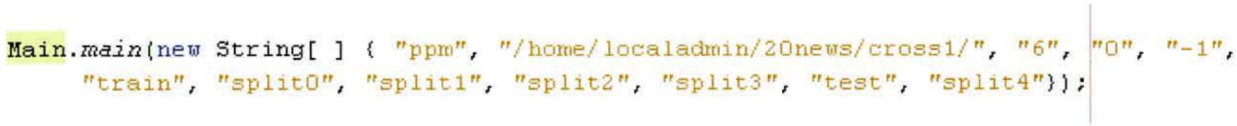
**Example 3**



```
Main.main(new String[ ] { "c", "/home/localadmin/20news/cross1/",
    "true", "true", "0", "-1", "train", "split0", "split1", "split2", "split3", "test", "split4"});
```

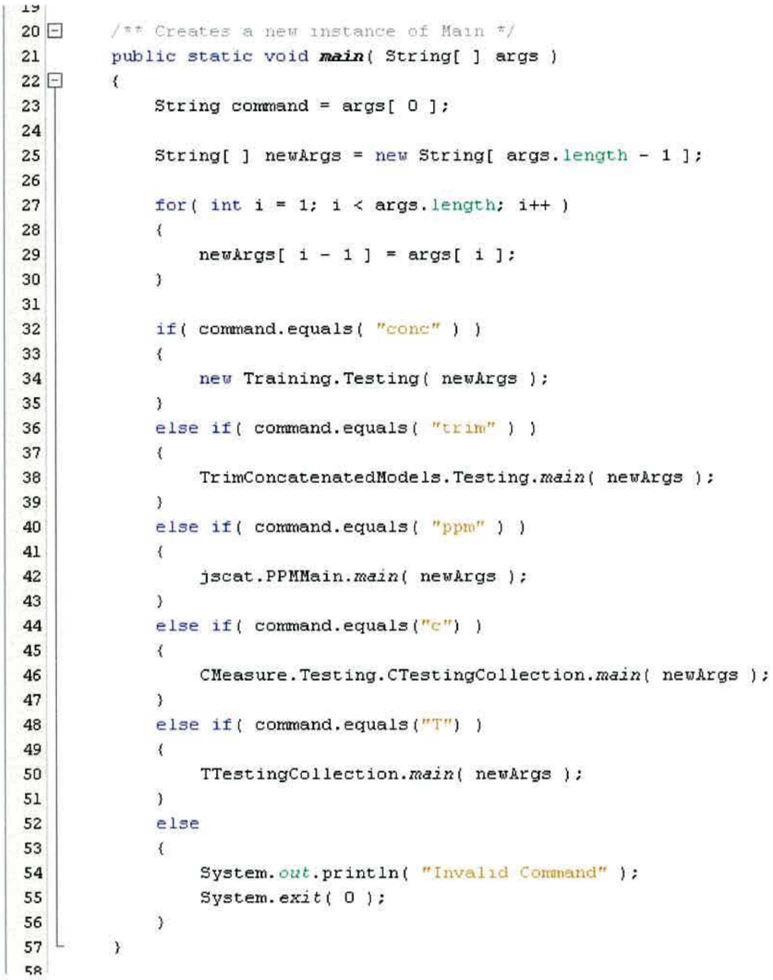
Example 4 shows how similar it is to process the existing models but on a different algorithm. This algorithm takes "6" as parameter and this indicated the PPM order in which we are interested. Each algorithm can take whichever parameters are required and loops can be used to perform experiments on all orders, all combinations of folders for cross validation and so on. These commands can also be bundled in order to further simplify the process or alternatively this information could be retrieved through a GUI if desired. A common entry point such as this is powerful in that it is possible to modify the parameters and perform experimentation of any algorithm, vary the training and testing data and also the order or substring length.

**Example 4**



```
Main.main(new String[ ] { "ppm", "/home/localadmin/20news/cross1/", "6", "0", "-1",
     "train", "split0", "split1", "split2", "split3", "test", "split4"});
```

Code sample 5.6 shows the main function and how the parameters are redirected depending on which operation the user asks to perform. Any new implementations that extend the base classes can add its case to this code and then be ran from the same common location as the other operators.



```
    /** Creates a new instance of Main */
    public static void main( String[ ] args )
    {
        String command = args[ 0 ];

        String[ ] newArgs = new String[ args.length - 1 ];

        for( int i = 1; i < args.length; i++ )
        {
            newArgs[ i - 1 ] = args[ i ];
        }

        if( command.equals( "conc" ) )
        {
            new Training.Testing( newArgs );
        }
        else if( command.equals( "trim" ) )
        {
            TrimConcatenatedModels.Testing.main( newArgs );
        }
        else if( command.equals( "ppm" ) )
        {
            jscat.PPMMain.main( newArgs );
        }
        else if( command.equals("c") )
        {
            CMeasure.Testing.CTestingCollection.main( newArgs );
        }
        else if( command.equals("T") )
        {
            TTestingCollection.main( newArgs );
        }
        else
        {
            System.out.println( "Invalid Command" );
            System.exit( 0 );
        }
    }
```

**Code Sample 5.6: jSCat's main entry point.**

## Chapter Discussion

Data preparation can be common among a number of algorithms and fits well within a common toolkit, allowing classes implementing different algorithms have been ran sequentially on a single data source within a single toolkit. This chapter has shown that optimisations can be found that drastically affect processing times and we have now been able to analyse stream based substring lengths that are much longer than previous research.

The use of base classes within the toolkit has made it possible for each algorithm to be introduced using very little code. The experimentation process has been simplified and experimentation can be started from a single location and can be varied by simply changing the input parameters.

There were a number of functions that were difficult to implement and a number of these have been explained in detail within this chapter, there was unfortunately too much code for each to be included within the chapter but they can be viewed within the source code on the attached DVD. The toolkit is available for download from http://aiia.cs.bangor.ac.uk.

## Chapter 6

# Experimental results

## Chapter Summary

The purpose of this chapter is to describe the experimental results for text categorization using stream-based methods. The methods have been implemented using suffix trees as described in the previous chapters. Results compare all algorithms within each dataset in order to discover the best performing within each corpus.

## Summary of each section

Section 6.1 details the experimental setup including how the datasets have been split to assist in experimentation Section 6.2 details all results collected from the experiments Section 6.3 lists timings received from experiments on 20newsgroups and allows us to compare the processing of each algorithm. Section 6.4 discusses results and notes all observations made from the comparisons.

## 6.1 Experimental setup

### 6.1.1 Corpora setup

The following corpora were used in the experiments. Note that the file names within each split for each corpus are detailed within the attached DVD.

#### 6.1.1.1 Reuters-10

The frequency of documents per category varies greatly; *earnings*, for example, contains 2877 training documents whilst the other nine, apart from *acq* (containing 1650) all contain less than 600, and this is also consistent across the testing documents. Table 6.1 shows the 10 most frequent categories and the number of documents within each.

The resulting corpus has 7193 training documents (5.9MB), and 2787 testing documents (2.1MB). The document sizes range from 47 bytes to 13.8 Kbytes. The training data per class varies from 213.7 Kbytes to 1.4MB.

| **Category** | **No. Training Docs** | **No. Testing Docs** |
|---|---|---|
| *earn* | 2877 | 1087 |
| *acq* | 1650 | 719 |
| *money-fx* | 538 | 179 |
| *grain* | 433 | 149 |
| *crude* | 389 | 189 |
| *trade* | 369 | 117 |
| *interest* | 347 | 131 |
| *ship* | 197 | 89 |
| *wheat* | 212 | 71 |
| *corn* | 181 | 56 |
| **Total** | **7193** | **2787** |

**Table 6.1: The number of testing and training documents for each category of Reuters 10 (R10).**

### 6.1.1.2 RCV1-Author

Here we select the top 50 authors (with respect to total size of articles). The authors and documents per set are detailed in Table 6.2.

| Author | No. Testing Docs | Training Document Size (KB) |
|---|---|---|
| Alan Baldwin | 26 | 785 |
| Alan Crosby | 26 | 705 |
| Alan Wheatley | 23 | 695 |
| Alastair Macdonald | 26 | 848 |
| Alexander Smith | 30 | 962 |
| Alistair Lyon | 29 | 918 |
| Amelia Torres | 23 | 653 |
| Andrew Browne | 20 | 655 |
| Andrew Cawthorne | 22 | 759 |
| Andrew Hill | 28 | 848 |
| Anthony Goodman | 26 | 694 |
| Arshad Mohammed | 23 | 696 |
| Benjamin Kang Lim | 27 | 768 |
| Carol Giacomo | 34 | 1134 |
| Charles Aldinger | 34 | 942 |
| Christian Jennings | 21 | 667 |
| David Crossland | 22 | 677 |
| David Lawder | 32 | 940 |
| Douglas Busvine | 23 | 719 |
| Ellen Freilich | 31 | 855 |
| Erik Kirschbaum | 24 | 735 |
| Evelyn Leopold | 45 | 1246 |
| Gene Gibbons | 25 | 756 |
| Glenn Somerville | 24 | 766 |
| Jane Macartney | 24 | 754 |
| John Gilardi | 26 | 768 |
| Laurence McQuillan | 23 | 684 |
| Leonard Santorelli | 25 | 892 |
| Linda Sieg | 19 | 657 |
| Maggie Fox | 23 | 665 |
| Marcel Michelson | 27 | 804 |
| Martin Cowley | 36 | 969 |
| Mike Collett | 27 | 805 |
| Mure Dickie | 28 | 805 |
| Nelson Graves | 23 | 704 |
| Oleg Shchedrov | 24 | 690 |
| Paul Holmes | 21 | 666 |
| Paul Majendie | 29 | 729 |
| Paul Mylrea | 25 | 730 |
| Paul Taylor | 23 | 811 |
| Peter Blackburn | 24 | 625 |
| Philippa Fletcher | 21 | 668 |
| Richard Melville | 38 | 1067 |
| Robert Evans | 26 | 774 |
| Robin Sidel | 24 | 646 |
| Steve Holland | 28 | 843 |
| Timothy Heritage | 27 | 864 |
| Todd Nissen | 27 | 732 |
| William Boston | 25 | 815 |
| William Wallis | 29 | 826 |
| **Total** | **1316** | |

**Table 6.2: The number of testing documents and size of category for each author within RCV1-Author.**

### 6.1.1.3 20Newsgroups

Table 6.3 shows the categories in 20-Newsgroups and their numbers of texts. There is no fixed way to split 20-newsgroup into a training set and a test set. This table also shows that the sizes of categories are relatively uniform compared with those of Reuters-21578. Five random splits of 80/20 training/testing were used as in Marton et al. (2005).

| Category | No. Docs | Category Size (Mbytes) |
|---|---|---|
| alt.atheism | 799 | 1.6 |
| comp.graphics | 973 | 1.6 |
| comp.os.ms-windows.misc | 985 | 2.3 |
| comp.sys.ibm.pc.hardware | 982 | 1.1 |
| comp.sys.mac.hardware | 961 | 1.0 |
| comp.windows.x | 980 | 1.8 |
| misc.forsale | 972 | 0.9 |
| rec.autos | 990 | 1.2 |
| rec.motorcycles | 994 | 1.1 |
| rec.sport.baseball | 994 | 1.3 |
| rec.sport.hockey | 999 | 1.7 |
| sci.crypt | 991 | 2.0 |
| sci.electronics | 981 | 1.2 |
| sci.med | 990 | 1.8 |
| sci.space | 987 | 1.7 |
| soc.religion.christian | 997 | 2.2 |
| talk.politics.guns | 910 | 1.8 |
| talk.politics.mideast | 940 | 2.8 |
| talk.politics.misc | 775 | 2.0 |
| talk.religion.misc | 628 | 1.3 |
| **Total** | **18828** | |

**Table 6.3: The number of documents and size of each category of 20-Newsgroups.**

### 6.1.1.4 Gutenberg

Table 6.4 lists the authors contained within the Gutenberg corpus, the number of documents from each author and also the total size of the documents. Some of the documents are as short as 98.4 Kbytes, and some as long as 1.1MB (many are novels). The training data per class ranges from 559.8 Kbytes to 3.1MB.

| Category | No. Docs | Category Size (Mbytes) |
|---|---|---|
| Charles Dickens | 4 | 2.8 |
| Daniel Defoe | 4 | 1.7 |
| Emerson | 4 | 1.4 |
| Jane Austen | 4 | 3.1 |
| Kipling | 4 | 1.4 |
| Shakespeare | 4 | 0.6 |
| Shaw | 4 | 1.2 |
| Twain | 4 | 3.0 |
| Wells | 4 | 2.0 |
| Wilde | 4 | 1.1 |
| **Total** | **40** | |

**Table 6.4: The number of documents and size of category for each author of Gutenberg.**

Note that the text inserted by Gutenberg was removed i.e. the disclaimer text was removed from each of the documents before processing.

4-fold cross-validation was used, with 3 training and 1 test document per class in each fold. Some works are as short as 98.4 Kbytes, and some as long as 1.1MB (many are novels). The training data per class ranges from 559.8 Kbytes to 3.1MB.

Table 6.5 allows us to easily compare the corpora's and shows that the four corpora are quite different and will allow for conclusions to be drawn from their differences.

| DataSet Name | No. Test Docs | No. Train Docs | No. Categories | Cross-Validation |
|---|---|---|---|---|
| 20-Newsgroups | 3792* | 15036* | 20 | Yes |
| Reuters 10 | 2237 | 5677 | 10 | No |
| RCV1-Author | 1316 | 50 | 50 | No |
| GutenBerg | 10 | 30 | 10 | Yes |

**Table 6.5: Summary of data sets used.**

* approximately as cross validation is performed and final split will have less

### 6.1.2 Hardware details

In order to obtain the results, 8 Dual Core PC's with 2GB RAM were used separately with no distributed computing, each is used to process a single algorithm at a time.

## 6.2 Results

This section describes the experimental results for the stream-based methods and protocols when used for text categorization on all of the data sets. Accuracy has been quoted since the experimentation was performed with data sets variants that have only singly labelled documents. In this setting, Bekkerman, R. (2001) states that the accepted performance measure is accuracy, and this was the evaluation measure that was specified the most in previously published experiments for each of the studied singly labelled variants of the data sets, and therefore provides a broader comparison than the alternative evaluation measures, recall and precision, and the breakeven point, as used by Yang (1999) for multiply labelled documents, for example.

### 6.2.1 C-Measure

This section displays C-Measure results for each of the corpora through use of tables and graphs. The highest accuracy achieved for each protocol are highlighted in bold font.

#### 6.2.1.1 20Newsgroups

| | **Concatenated Dynamic** | **Concatenated Static** | **NonConcatenated Dynamic** | **NonConcatenated Static** |
|---|---|---|---|---|
| **1** | 0.0427 | 0.0427 | 0.0644 | 0.0676 |
| **2** | 0.2379 | 0.2443 | 0.1424 | 0.1737 |
| **3** | 0.7875 | 0.7913 | 0.2789 | 0.3067 |
| **4** | 0.8792 | 0.8789 | 0.4534 | 0.4807 |
| **5** | 0.8979 | 0.8985 | 0.5716 | 0.5963 |
| **6** | 0.9045 | 0.9048 | 0.6626 | 0.6767 |
| **7** | 0.9041 | 0.9053 | 0.7303 | 0.7375 |
| **8** | 0.9053 | 0.9063 | 0.7762 | 0.7790 |
| **9** | 0.9063 | **0.9066** | 0.8087 | 0.8067 |
| **10** | **0.9070** | **0.9066** | 0.8309 | 0.8244 |
| **11** | 0.9057 | 0.9046 | 0.8440 | 0.8346 |
| **12** | 0.9038 | 0.9032 | 0.8504 | 0.8395 |
| **13** | 0.9017 | 0.9010 | 0.8513 | 0.8399 |
| **14** | 0.8979 | 0.8964 | 0.8515 | 0.8393 |
| **15** | 0.8923 | 0.8912 | **0.8520** | **0.8400** |
| **16** | 0.8885 | 0.8869 | 0.8508 | 0.8386 |
| **17** | 0.8847 | 0.8830 | 0.8501 | 0.8378 |
| **18** | 0.8788 | 0.8782 | 0.8488 | 0.8367 |
| **19** | 0.8749 | 0.8741 | 0.8477 | 0.8359 |
| **20** | 0.8712 | 0.8700 | 0.8469 | 0.8343 |
| **21** | 0.8675 | 0.8667 | 0.8455 | 0.8330 |
| **22** | 0.8636 | 0.8630 | 0.8442 | 0.8315 |
| **23** | 0.8607 | 0.8598 | 0.8427 | 0.8299 |
| **24** | 0.8569 | 0.8561 | 0.8407 | 0.8273 |
| **25** | 0.8549 | 0.8537 | 0.8391 | 0.8258 |
| **26** | 0.8519 | 0.8504 | 0.8377 | 0.8239 |
| **27** | 0.8478 | 0.8466 | 0.8360 | 0.8213 |
| **28** | 0.8450 | 0.8441 | 0.8355 | 0.8200 |
| **29** | 0.8417 | 0.8403 | 0.8332 | 0.8177 |
| **30** | 0.8386 | 0.8376 | 0.8313 | 0.8162 |
| **31** | 0.8350 | 0.8341 | 0.8299 | 0.8142 |
| **32** | 0.8320 | 0.8311 | 0.8275 | 0.8118 |
| **33** | 0.8299 | 0.8292 | 0.8262 | 0.8100 |
| **34** | 0.8266 | 0.8259 | 0.8238 | 0.8071 |
| **35** | 0.8242 | 0.8236 | 0.8216 | 0.8050 |
| **36** | 0.8214 | 0.8207 | 0.8197 | 0.8033 |
| **37** | 0.8192 | 0.8183 | 0.8173 | 0.8004 |
| **38** | 0.8167 | 0.8157 | 0.8153 | 0.7983 |
| **39** | 0.8142 | 0.8131 | 0.8132 | 0.7962 |
| **40** | 0.8126 | 0.8116 | 0.8119 | 0.7950 |

**Table 6.6: Accuracies achieved by applying C-Measure (up to length 40 due to page restriction) to 20Newsgroups for each protocol.**

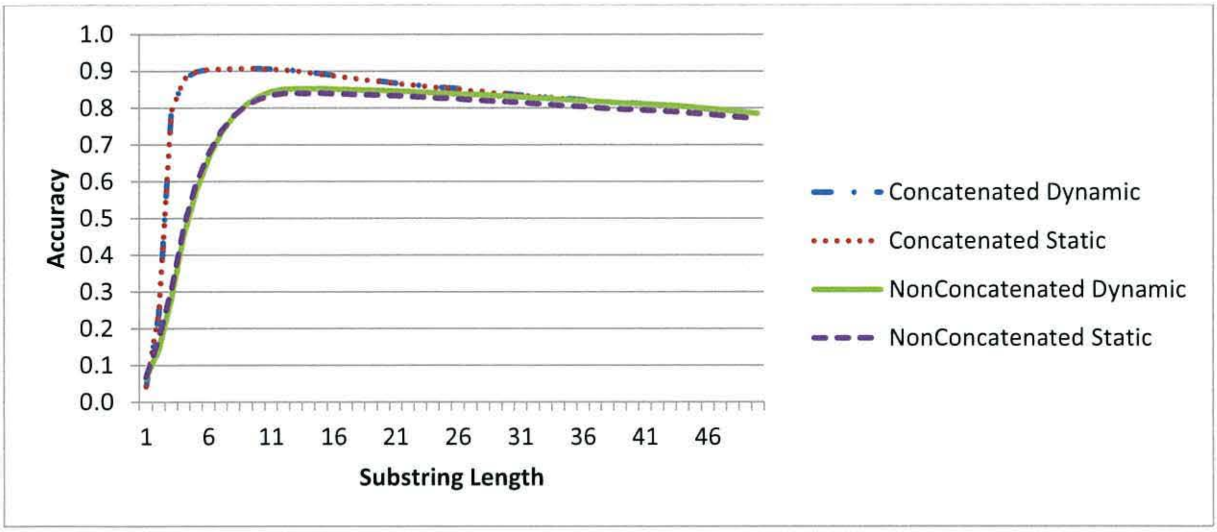


**Figure 6.1: Accuracies achieved by applying C-Measure to 20Newsgroups for each protocol.**

The results for the experiment are shown in Table 6.6 and graphed in Figure 6.1. For Table 6.6, the leftmost column indicates the substring length (shown as x axis in Figure 6.1) and for each protocol an average accuracy is shown (shown as y axis in Figure 6.1). The results show that concatenated models clearly outperform non-concatenated ones and that dynamic models marginally outperform static models. The results also indicate that shorter substring lengths perform better for concatenated cases than for their non-concatenated counterparts. The optimal substring length is shorter than that of for Gutenberg but is similar to RCV1-Author which is a more similar corpus in relation to the number of files per class and size of each file. It is also noticeable from the graph that even short substring lengths are good at categorizing which is important in situations where the available processing time is limited.

#### 6.2.1.2 Gutenberg

| | Concatenated Dynamic | Concatenated Static | NonConcatenated Dynamic | NonConcatenated Static |
|---|---|---|---|---|
| **1** | 0.08 | 0.08 | 0.18 | 0.13 |
| **2** | 0.18 | 0.20 | 0.13 | 0.23 |
| **3** | 0.18 | 0.23 | 0.15 | 0.18 |
| **4** | 0.23 | 0.23 | 0.18 | 0.18 |
| **5** | 0.30 | 0.40 | 0.15 | 0.25 |
| **6** | 0.30 | 0.40 | 0.25 | 0.25 |
| **7** | 0.30 | 0.40 | 0.28 | 0.38 |
| **8** | 0.35 | 0.48 | 0.30 | 0.45 |
| **9** | 0.40 | 0.48 | 0.38 | 0.45 |
| **10** | 0.45 | 0.55 | 0.43 | 0.48 |
| **11** | 0.48 | 0.58 | 0.40 | 0.48 |
| **12** | 0.55 | 0.58 | 0.50 | 0.50 |
| **13** | 0.55 | 0.60 | 0.53 | 0.55 |
| **14** | 0.58 | 0.60 | 0.55 | 0.60 |
| **15** | 0.60 | 0.63 | 0.60 | 0.63 |
| **16** | 0.63 | 0.63 | 0.63 | 0.63 |
| **17** | 0.63 | 0.63 | 0.63 | 0.65 |
| **18** | 0.63 | 0.70 | 0.70 | 0.68 |
| **19** | 0.70 | 0.73 | 0.68 | 0.68 |
| **20** | 0.70 | 0.73 | 0.68 | 0.68 |
| **21** | 0.75 | **0.78** | 0.75 | 0.73 |
| **22** | 0.75 | 0.75 | **0.78** | 0.73 |
| **23** | 0.75 | 0.75 | **0.78** | **0.78** |
| **24** | **0.78** | 0.75 | **0.78** | 0.75 |
| **25** | **0.78** | 0.75 | **0.78** | 0.75 |
| **26** | 0.75 | 0.75 | **0.78** | 0.75 |
| **27** | 0.75 | 0.75 | **0.78** | 0.73 |
| **28** | 0.75 | 0.73 | 0.75 | 0.70 |
| **29** | 0.73 | 0.73 | 0.73 | 0.63 |
| **30** | 0.68 | 0.65 | 0.70 | 0.60 |
| **31** | 0.63 | 0.63 | 0.58 | 0.55 |
| **32** | 0.58 | 0.60 | 0.55 | 0.53 |
| **33** | 0.50 | 0.53 | 0.48 | 0.45 |
| **34** | 0.45 | 0.48 | 0.48 | 0.45 |
| **35** | 0.43 | 0.43 | 0.43 | 0.38 |
| **36** | 0.43 | 0.40 | 0.48 | 0.43 |
| **37** | 0.45 | 0.43 | 0.48 | 0.45 |
| **38** | 0.45 | 0.43 | 0.53 | 0.50 |
| **39** | 0.40 | 0.40 | 0.50 | 0.48 |
| **40** | 0.38 | 0.38 | 0.50 | 0.48 |

**Table 6.7: Accuracies achieved by applying C-Measure (up to length 40 due to page restriction) to Gutenberg for each protocol.**

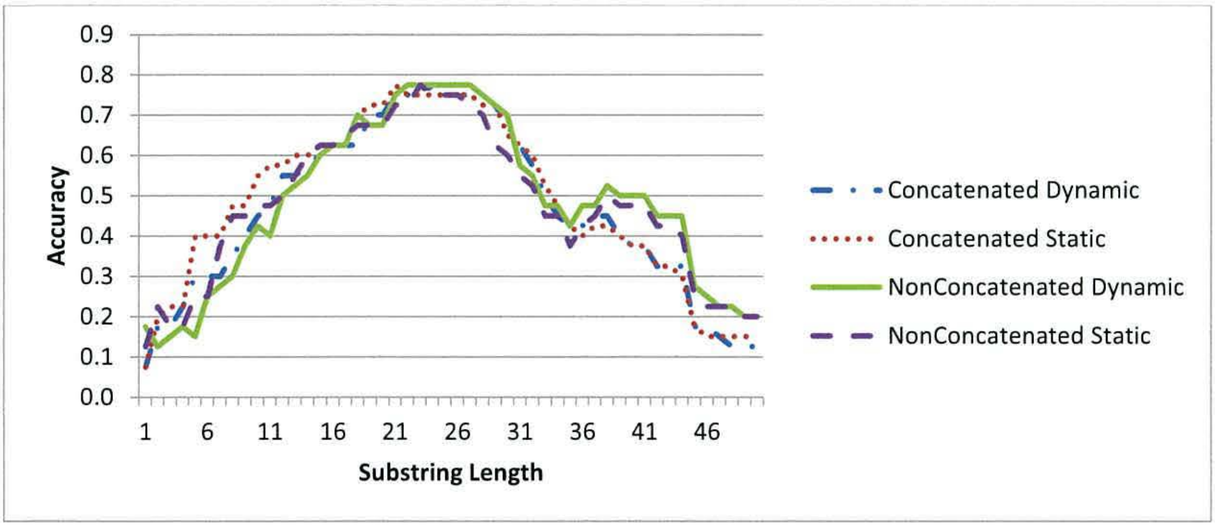


**Figure 6.2: Accuracies achieved by applying C-Measure to Gutenberg for each protocol.**

The results for the experiment are shown in Table 6.7 and graphed in Figure 6.2. For Table 6.7, the leftmost column indicates the substring length (shown as x axis in Figure 6.2) and for each protocol an average accuracy is shown (shown as y axis in Figure 6.2). Figure 6.2 shows that the optimal substring length is much larger for this corpus, typically between 21 and 28. Accuracy at lower lengths are not as effective as they were with 20Newsgroups and also trail off very quickly for substring lengths greater than around 30. Interestingly it is hard to distinguish between any of the protocols for this corpora, possibly the differences are because the texts are much larger, possibly because there are so few documents. Either way it appears to show that the effectiveness of each protocol differs between corpora and this is an important finding.

#### 6.2.1.3 RCV1-Author

| | Concatenated Dynamic | Concatenated Static |
|---|---|---|
| **1** | 0.0182 | 0.0182 |
| **2** | 0.4871 | 0.4894 |
| **3** | 0.7933 | 0.7971 |
| **4** | 0.8343 | 0.8381 |
| **5** | 0.8556 | 0.8564 |
| **6** | 0.8609 | 0.8609 |
| **7** | 0.8663 | 0.8655 |
| **8** | 0.8716 | 0.8731 |
| **9** | 0.8754 | 0.8754 |
| **10** | 0.8754 | 0.8769 |
| **11** | 0.8777 | 0.8799 |
| **12** | 0.8815 | **0.8837** |
| **13** | 0.8815 | 0.8815 |
| **14** | 0.8815 | 0.8830 |
| **15** | 0.8792 | 0.8822 |
| **16** | **0.8830** | 0.8807 |
| **17** | 0.8815 | 0.8815 |
| **18** | 0.8784 | 0.8761 |
| **19** | 0.8761 | 0.8754 |
| **20** | 0.8777 | 0.8761 |
| **21** | 0.8746 | 0.8731 |
| **22** | 0.8746 | 0.8739 |
| **23** | 0.8693 | 0.8701 |
| **24** | 0.8701 | 0.8701 |
| **25** | 0.8716 | 0.8716 |
| **26** | 0.8731 | 0.8731 |
| **27** | 0.8701 | 0.8701 |
| **28** | 0.8640 | 0.8647 |
| **29** | 0.8609 | 0.8625 |
| **30** | 0.8602 | 0.8617 |
| **31** | 0.8556 | 0.8564 |
| **32** | 0.8511 | 0.8518 |
| **33** | 0.8488 | 0.8488 |
| **34** | 0.8465 | 0.84650 |
| **35** | 0.8419 | 0.8419 |
| **36** | 0.8389 | 0.8389 |
| **37** | 0.8336 | 0.8336 |
| **38** | 0.8267 | 0.8267 |
| **39** | 0.8222 | 0.8214 |
| **40** | 0.8146 | 0.8138 |

**Table 6.8: Accuracies achieved by applying C-Measure (up to length 40 due to page restriction) to RCV1-Author for each protocol.**

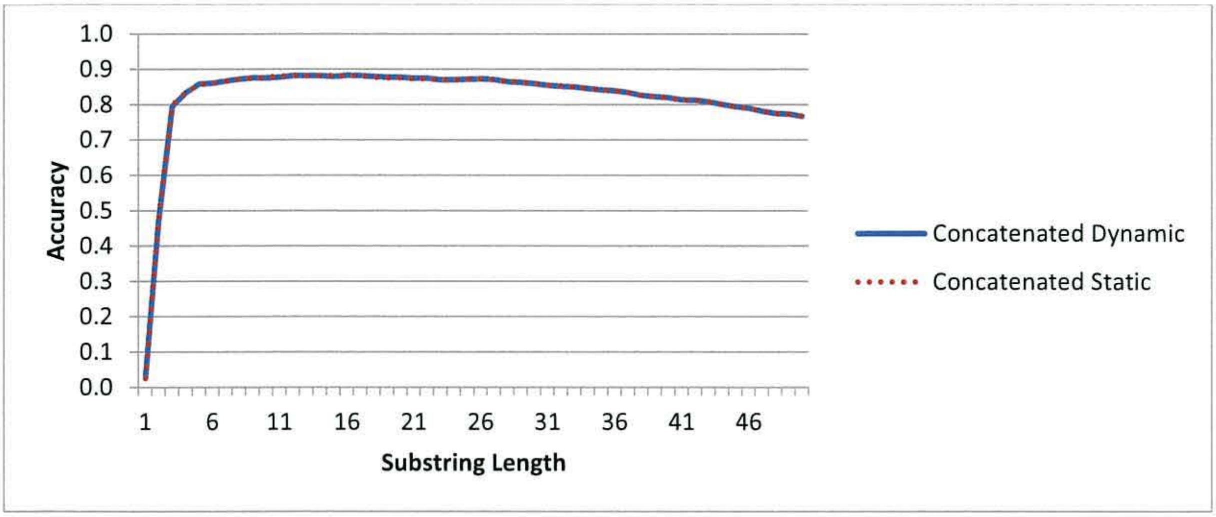


**Figure 6.3: Accuracies achieved by applying C-Measure to RCV1-Author for each protocol.**

The results for the experiment are shown in Table 6.8 and graphed in Figure 6.3. For Table 6.8, the leftmost column indicates the substring length (shown as x axis in Figure 6.3) and for each protocol an average accuracy is shown (shown as y axis in Figure 6.3). For this corpus it is very difficult to identify differences in the performance of each protocol for any of the substring lengths. The results are more similar to those obtained from 20newsgroups than from Gutenberg, possibly this is due to the number of files and their sizes being more similar to those within the 20 Newsgroups corpora than Gutenberg.

#### 6.2.1.4 Reuters-10

| | **Concatenated Dynamic** | **Concatenated Static** | **NonConcatenated Dynamic** | **NonConcatenated Static** |
|---|---|---|---|---|
| 1 | 0.3111 | 0.3111 | 0.4198 | 0.4184 |
| 2 | 0.6553 | 0.6553 | 0.6021 | 0.6442 |
| 3 | 0.7470 | 0.7501 | 0.6527 | 0.7050 |
| 4 | 0.7792 | 0.7921 | 0.7819 | 0.8114 |
| 5 | 0.8342 | 0.8386 | 0.8426 | 0.8713 |
| 6 | 0.8623 | 0.8659 | 0.8695 | 0.8838 |
| 7 | 0.8784 | 0.8820 | 0.8860 | 0.8909 |
| 8 | 0.8936 | 0.8967 | 0.8918 | 0.8949 |
| 9 | 0.8994 | 0.9008 | **0.8927** | **0.8954** |
| 10 | 0.9021 | 0.9048 | 0.8918 | 0.8923 |
| 11 | 0.9061 | 0.9080 | 0.8873 | 0.8860 |
| 12 | 0.9119 | 0.9137 | 0.8824 | 0.8811 |
| 13 | 0.9146 | 0.9169 | 0.8806 | 0.8757 |
| 14 | 0.9169 | **0.9173** | 0.8730 | 0.8730 |
| 15 | **0.9177** | 0.9169 | 0.8704 | 0.8681 |
| 16 | 0.9173 | 0.9169 | 0.8646 | 0.8650 |
| 17 | 0.9151 | 0.9137 | 0.8592 | 0.8610 |
| 18 | 0.9128 | 0.9133 | 0.8578 | 0.8614 |
| 19 | 0.9066 | 0.9079 | 0.8538 | 0.8543 |
| 20 | 0.9025 | 0.9025 | 0.8529 | 0.8552 |
| 21 | 0.8949 | 0.8945 | 0.8494 | 0.8502 |
| 22 | 0.8909 | 0.8914 | 0.8476 | 0.8480 |
| 23 | 0.8838 | 0.8838 | 0.8435 | 0.8444 |
| 24 | 0.8775 | 0.8771 | 0.8435 | 0.8453 |
| 25 | 0.8717 | 0.8708 | 0.8364 | 0.8368 |
| 26 | 0.8610 | 0.8596 | 0.8315 | 0.8310 |
| 27 | 0.8444 | 0.8435 | 0.8221 | 0.8220 |
| 28 | 0.8270 | 0.8266 | 0.8105 | 0.8105 |
| 29 | 0.7997 | 0.7997 | 0.7890 | 0.7890 |
| 30 | 0.7805 | 0.7805 | 0.7698 | 0.7698 |
| 31 | 0.7599 | 0.7599 | 0.7501 | 0.7501 |
| 32 | 0.7278 | 0.7278 | 0.7202 | 0.7202 |
| 33 | 0.6987 | 0.6987 | 0.6911 | 0.6911 |
| 34 | 0.6665 | 0.6665 | 0.6585 | 0.6585 |
| 35 | 0.6446 | 0.6446 | 0.6370 | 0.6370 |
| 36 | 0.6187 | 0.6187 | 0.6129 | 0.6129 |
| 37 | 0.5834 | 0.5834 | 0.5780 | 0.5780 |
| 38 | 0.5579 | 0.5579 | 0.5530 | 0.5530 |
| 39 | 0.5248 | 0.5248 | 0.5217 | 0.5217 |
| 40 | 0.4989 | 0.4989 | 0.4962 | 0.4962 |

**Table 6.9: Accuracies achieved by applying C-Measure (up to length 40 due to page restriction) to Reuters-10 for each protocol.**

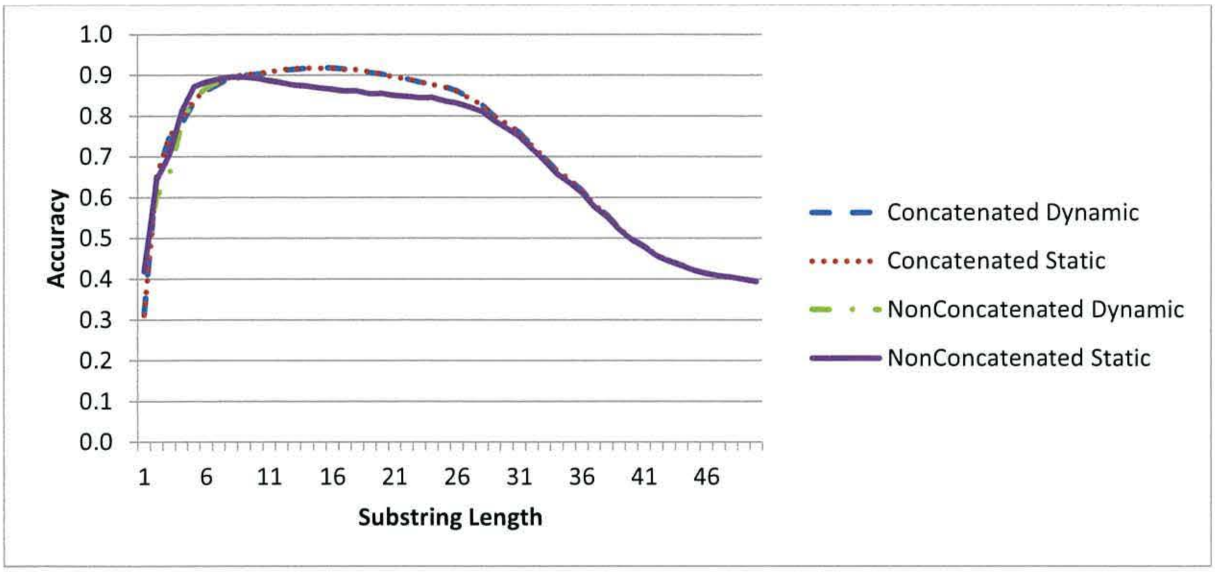


**Figure 6.4: Accuracies achieved by applying C-Measure to Reuters-10 for each protocol.**

The results for the experiment are shown in Table 6.9 and graphed in Figure 6.4. For Table 6.9, the leftmost column indicates the substring length (shown as x axis in Figure 6.4) and for each protocol an average accuracy is shown (shown as y axis in Figure 6.4). The results show that concatenated models outperform non-concatenated ones as was the case for 20newsgroups (see 6.2.1.1), but not as clearly. The results differ from 6.2.1.1 in that for R10 concatenated models achieve their highest accuracy at a longer substring length, typically between 14 and 15. The optimal substring lengths for non-concatenated models also differ to 6.2.1.1 in that the optimal substring length is shorter at a length of 9.

### 6.2.2 PPM

This section displays PPMC and PPMD accuracies achieved for each of the corpora, both with and without update exclusions. The results are again presented in both tabulated and graphical format and the best results for each protocol are highlighted in bold font for easy comparison. Although it would have been desirable to have attained results up to order 6 for all corpora, in reality the resources were not available to have computed these results because of the high computational overheads (both memory and execution time) for these high order models.

#### 6.2.2.1 20 Newsgroups

| | Concatenated Dynamic | | Concatenated Static | | NonConcatenated Dynamic | | NonConcatenated Static | |
|---|---|---|---|---|---|---|---|---|
| | With Exclusions | Without Exclusions | With Exclusions | Without Exclusions | With Exclusions | Without Exclusions | With Exclusions | Without Exclusions |
| 2 | 0.8886 | 0.8930 | 0.8851 | 0.8903 | 0.7828 | 0.7659 | 0.7529 | 0.7412 |

**Table 6.10: Accuracies achieved by applying PPMC to 20Newsgroups for each protocol.**

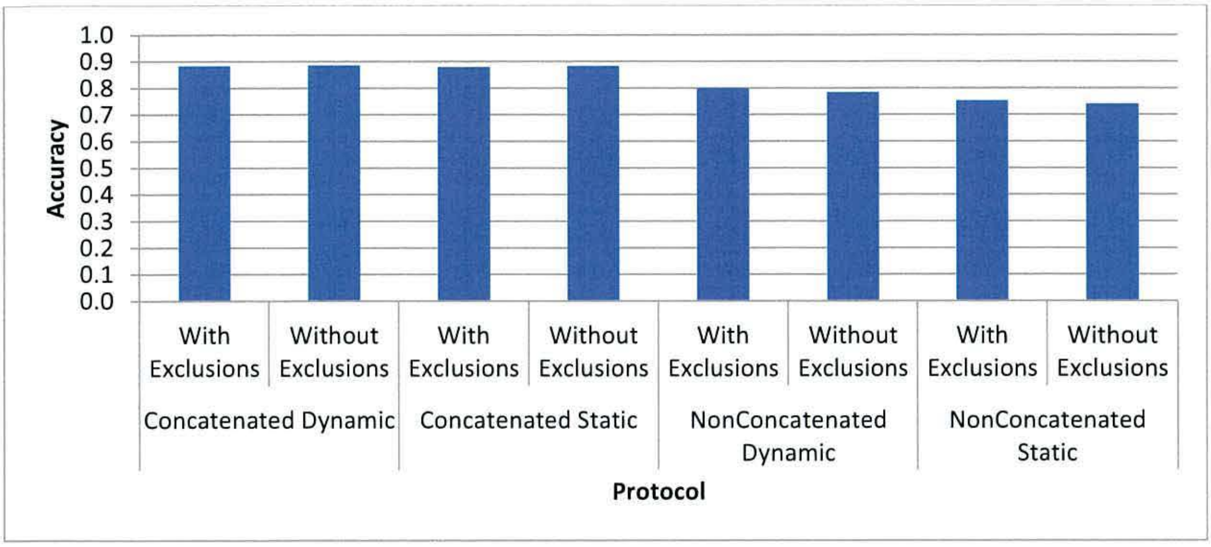


**Figure 6.5: Accuracies achieved by applying PPMC order 2 to 20Newsgroups for each protocol.**

| | Concatenated Dynamic | | Concatenated Static | | NonConcatenated Dynamic | | NonConcatenated Static | |
|---|---|---|---|---|---|---|---|---|
| | With Exclusions | Without Exclusions | With Exclusions | Without Exclusions | With Exclusions | Without Exclusions | With Exclusions | Without Exclusions |
| 2 | 0.8920 | 0.8955 | 0.8877 | 0.8910 | 0.7812 | 0.7629 | 0.7537 | 0.7372 |

**Table 6.11: Accuracies achieved by applying PPMD to 20Newsgroups for each protocol.**

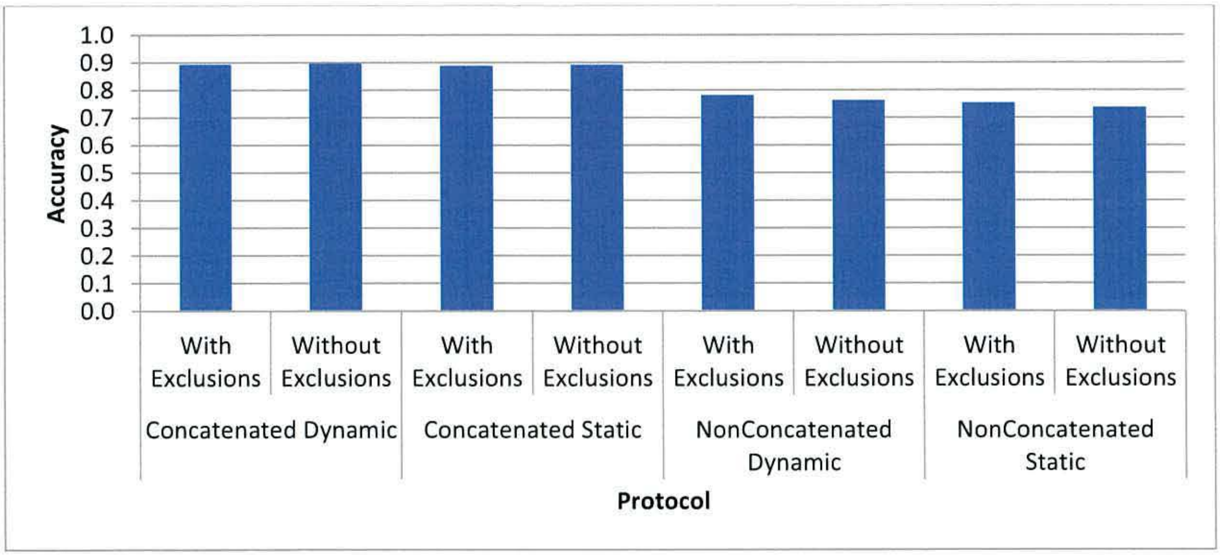


**Figure 6.6: Accuracies achieved by applying PPMD order 2 to 20Newsgroups for each protocol.**

The results for the PPMC experiments are shown in Table 6.10 and graphed in Figure 6.5 and the results for PPMD are shown in Table 6.11 and Figure 6.6. For Table 6.10 and 6.11, the leftmost column indicates the substring length (shown as x axis in Figure 6.5 and 6.6) and for each protocol an average accuracy is shown (shown as y axis in Figure 6.5 and 6.6).

It is clear that for PPMC concatenated models performed better than non-concatenated and dynamic models performed better than static models, a finding that was unclear for C-Measure results. The results also show that without exclusions achieved best results for concatenated models, but the opposite is true for non-concatenated models.

For PPMD, concatenated models again performed better than non-concatenated and dynamic models performed better than static ones. Without exclusions achieved best results for concatenated models, but the opposite is true for non-concatenated models.

### 6.2.2.2 Gutenberg

| | **Concatenated Dynamic** | | **Concatenated Static** | | **NonConcatenated Dynamic** | | **NonConcatenated Static** | |
|---|---|---|---|---|---|---|---|---|
| | **With Exclusions** | **Without Exclusions** | **With Exclusions** | **Without Exclusions** | **With Exclusions** | **Without Exclusions** | **With Exclusions** | **Without Exclusions** |
| **2** | 0.8 | 0.8 | **0.75** | **0.7** | 0.625 | 0.65 | 0.55 | **0.55** |
| **3** | **0.925** | **0.95** | 0.725 | 0.675 | **0.925** | **0.9** | 0.575 | **0.55** |
| **4** | 0.875 | 0.875 | 0.725 | 0.65 | 0.9 | 0.8 | 0.575 | 0.5 |
| **5** | 0.875 | 0.875 | 0.7 | 0.6 | 0.875 | 0.875 | 0.575 | 0.525 |
| **6** | 0.85 | 0.875 | 0.7 | 0.625 | 0.55 | 0.5 | **0.6** | 0.525 |
| **7** | 0.4 | 0.375 | 0.7 | 0.65 | 0.25 | 0.225 | **0.6** | **0.55** |

**Table 6.12: Accuracies achieved by applying PPMC to Gutenberg for each protocol.**

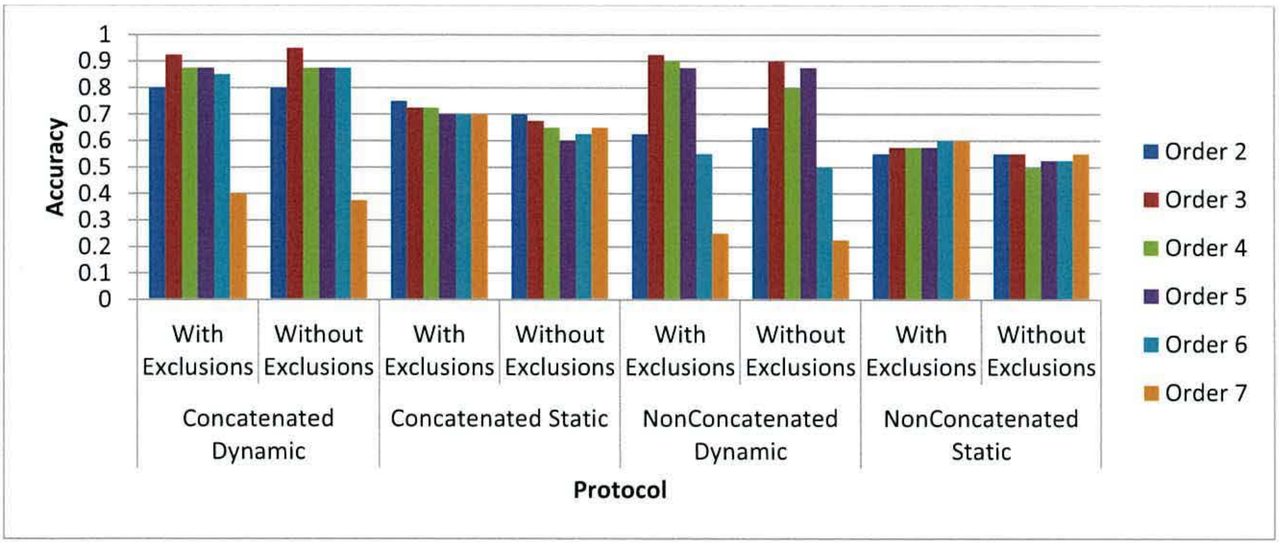


**Figure 6.7: Accuracies achieved by applying PPMC to Gutenberg for each protocol.**

|  | Concatenated Dynamic |  | Concatenated Static |  | NonConcatenated Dynamic |  | NonConcatenated Static |  |
|---|---|---|---|---|---|---|---|---|
|  | With Exclusions | Without Exclusions | With Exclusions | Without Exclusions | With Exclusions | Without Exclusions | With Exclusions | Without Exclusions |
| **2** | 0.75 | 0.775 | **0.75** | **0.7** | 0.6 | 0.575 | 0.55 | 0.525 |
| **3** | **0.95** | **0.95** | **0.75** | 0.675 | 0.875 | **0.875** | 0.55 | **0.55** |
| **4** | 0.925 | 0.9 | **0.75** | 0.65 | 0.9 | 0.825 | 0.575 | 0.5 |
| **5** | 0.9 | 0.875 | 0.7 | 0.575 | **0.925** | **0.875** | 0.55 | 0.525 |
| **6** | 0.875 | 0.9 | 0.7 | 0.625 | 0.575 | 0.45 | **0.6** | 0.525 |
| **7** | 0.425 | 0.35 | 0.7 | 0.65 | 0.275 | 0.225 | **0.6** | 0.525 |

**Table 6.13: Accuracies achieved by applying PPMD to Gutenberg for each protocol.**

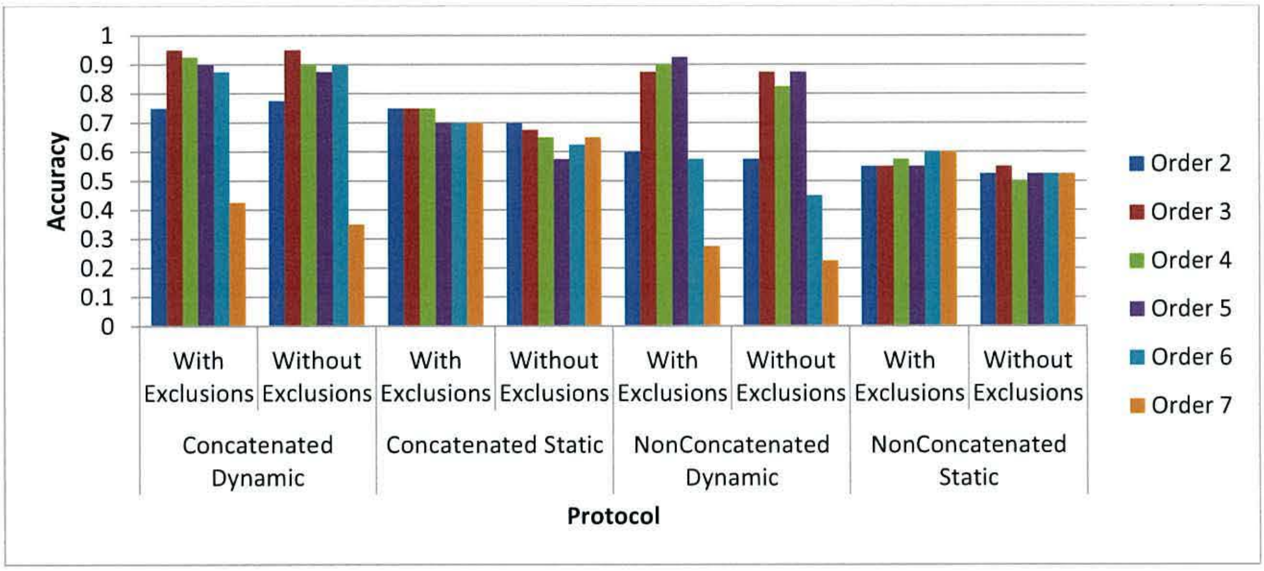


**Figure 6.8: Accuracies achieved by applying PPMD to Gutenberg for each protocol.**

The results for the PPMC experiments are shown in Table 6.12 and graphed in Figure 6.7 and the results for PPMD are shown in Table 6.13 and Figure 6.8. For Table 6.12 and 6.13, the leftmost column indicates the substring length (shown as x axis in Figure 6.7 and 6.8) and for each protocol an average accuracy is shown (shown as y axis in Figure 6.7 and 6.8).

For PPMC, in all cases, with exclusions outperforms without, dynamic models perform much better than static ones and concatenated models easily outperform its non-concatenated counterpart. It appears that shorter context lengths provide the best categorization for this corpus in concatenated cases, but it is less clear as to which is best for non-concatenated. With the number of testing documents within the corpus being so few we see a large difference in results, with one file accounting for 10% accuracy in each cross validation performed.

Notice that for this corpus there is little difference between PPMC and PPMD. For PPMD, with exclusions outperformed without exclusions as was the case with its PPMC results. Dynamic models again performed much better than static ones in all cases, as did concatenated, again better than non-concatenated in all cases. It would be fair to say that context lengths of 2 provided the highest categorization as on more occasions than any other it achieved the highest accuracy.

### 6.2.2.3 RCV1-Author

| | Concatenated Dynamic | | Concatenated Static | |
|---|---|---|---|---|
| | **With Exclusions** | **Without Exclusions** | **With Exclusions** | **Without Exclusions** |
| **2** | 0.7994 | 0.8055 | 0.8002 | 0.8123 |
| **3** | **0.8480** | **0.8503** | **0.8495** | **0.8511** |

**Table 6.14: Accuracies achieved by applying PPMC to RCV1-Author for each protocol.**

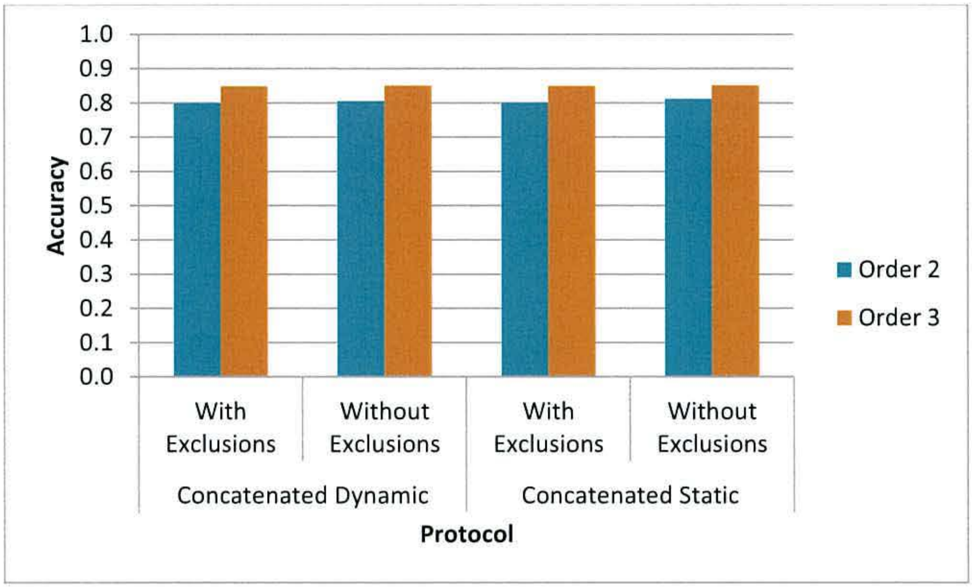


**Figure 6.9: Accuracies achieved by applying PPMC to RCV1-Author for each protocol.**

| | Concatenated Dynamic | | Concatenated Static | |
|---|---|---|---|---|
| | **With Exclusions** | **Without Exclusions** | **With Exclusions** | **Without Exclusions** |
| **2** | 0.8062 | 0.8146 | 0.8047 | 0.8123 |
| **3** | **0.8503** | **0.8533** | **0.8488** | **0.8518** |

**Table 6.15: Accuracies achieved by applying PPMD to RCV1-Author for each protocol.**

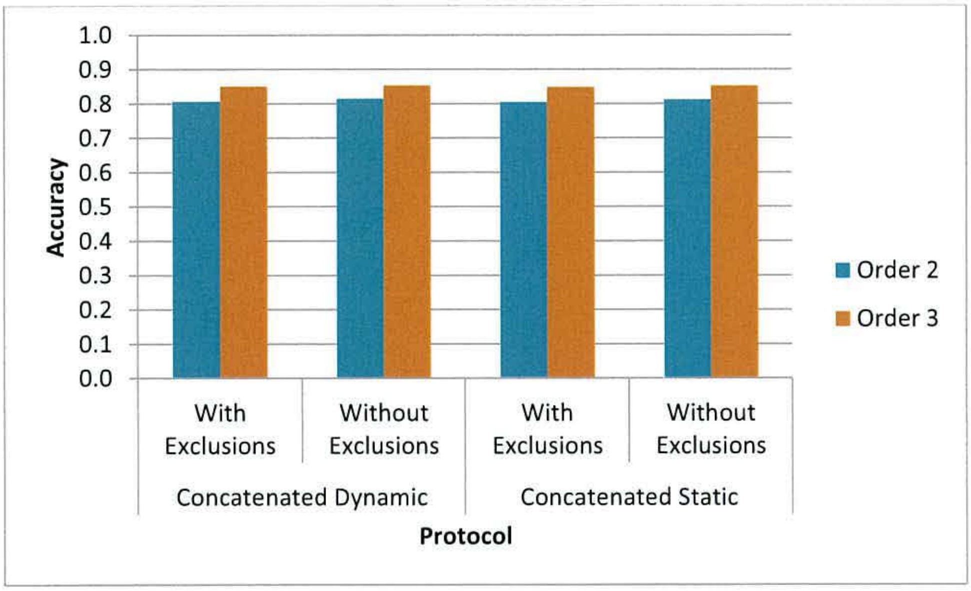


**Figure 6.10: Accuracies achieved by applying PPMD to RCV1-Author for each protocol.**

The results for the PPMC experiments are shown in Table 6.14 and graphed in Figure 6.9 and the results for PPMD are shown in Table 6.15 and Figure 6.10. For Table 6.14 and 6.15, the leftmost column indicates the substring length (shown as x axis in Figure 6.9 and 6.10) and for each protocol an average accuracy is shown (shown as y axis in Figure 6.9 and 6.10).

In all cases without exclusions performed better than with and interestingly dynamic models performed better for PPMD but the opposite is true for PPMC. The results also show that order 3 greatly improved the accuracies compared to those received for order 2 as is the case with Gutenberg (see 6.2.2.2) and Reuters-10 (see 6.2.2.4).

### 6.2.2.4 Reuters-10

| | **Concatenated Dynamic** | | **Concatenated Static** | | **NonConcatenated Dynamic** | | **NonConcatenated Static** | |
|---|---|---|---|---|---|---|---|---|
| | **With Exclusions** | **Without Exclusions** | **With Exclusions** | **Without Exclusions** | **With Exclusions** | **Without Exclusions** | **With Exclusions** | **Without Exclusions** |
| 2 | 0.9477 | 0.9437 | 0.9450 | 0.9392 | **0.8556** | **0.8489** | **0.8990** | 0.8972 |
| 3 | **0.9531** | **0.9513** | **0.9455** | **0.9410** | 0.4962 | 0.4864 | **0.8990** | 0.9021 |
| 4 | 0.9227 | 0.9253 | 0.9405 | 0.9343 | 0.4680 | 0.4193 | 0.8963 | **0.9080** |

**Table 6.16: Accuracies achieved by applying PPMC to Reuters-10 for each protocol.**

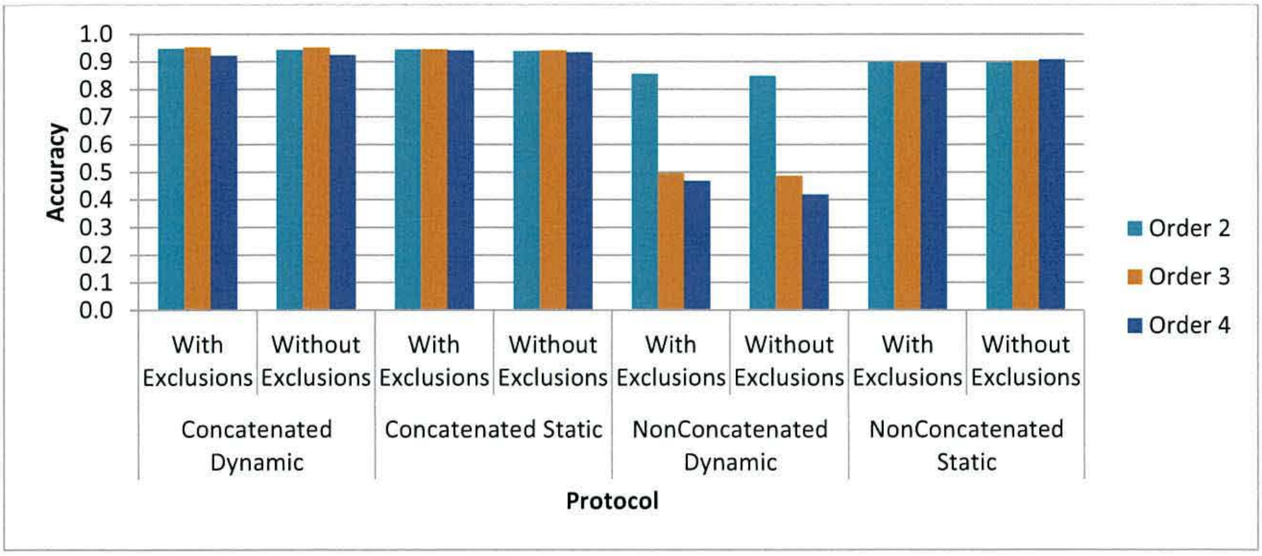


**Figure 6.11: Accuracies achieved by applying PPMC to Reuters-10 for each protocol.**

| | Concatenated Dynamic | | Concatenated Static | | NonConcatenated Dynamic | | NonConcatenated Static | |
|---|---|---|---|---|---|---|---|---|
| | **With Exclusions** | **Without Exclusions** | **With Exclusions** | **Without Exclusions** | **With Exclusions** | **Without Exclusions** | **With Exclusions** | **Without Exclusions** |
| 2 | 0.9455 | 0.9450 | 0.9441 | **0.9392** | **0.8538** | **0.8476** | **0.9003** | 0.8976 |
| 3 | **0.9517** | **0.9490** | **0.9450** | 0.9374 | 0.4975 | 0.4949 | 0.8976 | 0.9034 |
| 4 | 0.9298 | 0.9280 | 0.9397 | 0.9338 | 0.5069 | 0.4662 | 0.8990 | **0.9052** |

**Table 6.17: Accuracies achieved by applying PPMD to Reuters-10 for each protocol.**

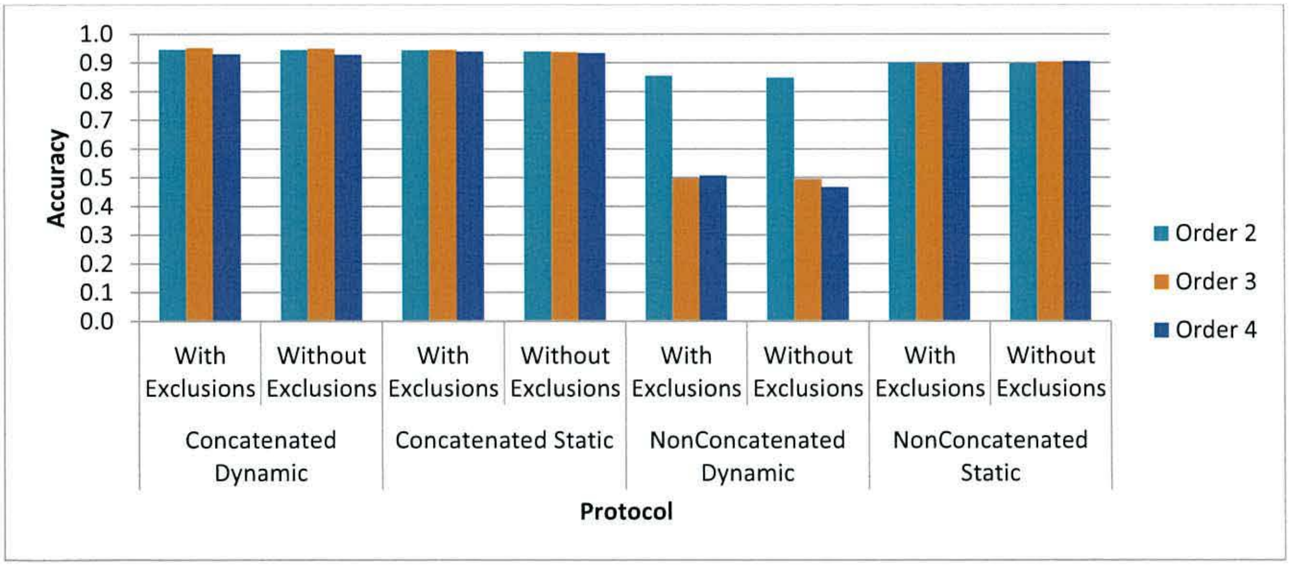


**Figure 6.12: Accuracies achieved by applying PPMD to Reuters-10 for each protocol.**

The results for the PPMC experiments are shown in Table 6.16 and graphed in Figure 6.11 and the results for PPMD are shown in Table 6.17 and Figure 6.12. For Table 6.16 and 6.17, the leftmost column indicates the substring length (shown as x axis in Figure 6.11 and 6.12) and for each protocol an average accuracy is shown (shown as y axis in Graph 6.11 and 6.12).

The most inconsistent result was found within R10 as there is a noticeable drop in accuracy for an order of 2 and 3 for non-concatenated models for both PPMC and PPMD. The highest accuracies for concatenated models were achieved for order 3 for both PPMC and PPMD. It is difficult to determine the best order for non-concatenated static though it would appear that lower orders perform better when update exclusions are performed, and longer ones for when they are not. As with 20newsgroups and Gutenberg, concatenated models outperformed non-concatenated, in this case quite significantly for both PPMC and PPMD.

### 6.2.3 R-Measure

This section displays accuracies achieved for each of the corpora using each of the R-Measure algorithms discussed in 3.1. For R-Ranges, the substring lengths investigated have a minimum from 1 up to 29 and a maximum from 2 up to 30. Substring lengths of up to length 30 are investigated for both $R_{\geq q}$-Measure and $R_{\leq q}$-Measure.

#### 6.2.3.1 20Newsgroups

The results for the $R_{\leq q}$-Measure experiments are shown in Table 6.18, $R_{\geq q}$-Measure in Table 6.19 and Tables 6.20-6.23 show results for R-Ranges. Tables 6.18 and 6.19 display accuracies for all four protocols with the leftmost column indicating the lower substring limit for each algorithm. Tables 6.20-6.23 show the accuracy for each range with a single table displaying results for a single protocol. The highest accuracies are again highlighted in bold font.

| | **Concatenated Dynamic** | **Concatenated Static** | **NonConcatenated Dynamic** | **NonConcatenated Static** |
|---|---|---|---|---|
| 1 | 0.0427 | 0.0427 | 0.0644 | 0.0676 |
| 2 | 0.2379 | 0.2443 | 0.1403 | 0.1687 |
| 3 | 0.7805 | 0.7831 | 0.2324 | 0.2620 |
| 4 | 0.8727 | 0.8719 | 0.3598 | 0.3789 |
| 5 | 0.8939 | 0.8946 | 0.4513 | 0.4647 |
| 6 | 0.9032 | 0.9027 | 0.5183 | 0.5270 |
| 7 | 0.9056 | 0.9064 | 0.5735 | 0.5795 |
| 8 | 0.9084 | 0.9084 | 0.6171 | 0.6193 |
| 9 | 0.9094 | 0.9098 | 0.6543 | 0.6517 |
| 10 | 0.9107 | 0.9111 | 0.6828 | 0.6772 |
| 11 | 0.9113 | 0.9110 | 0.7043 | 0.6970 |
| 12 | 0.9130 | 0.9117 | 0.7202 | 0.7128 |
| 13 | 0.9133 | 0.9122 | 0.7319 | 0.7253 |
| 14 | 0.9141 | 0.9130 | 0.7423 | 0.7352 |
| 15 | **0.9145** | 0.9131 | 0.7508 | 0.7429 |
| 16 | 0.9140 | **0.9133** | 0.7582 | 0.7511 |
| 17 | 0.9135 | 0.9125 | 0.7643 | 0.7559 |
| 18 | 0.9126 | 0.9117 | 0.7701 | 0.7615 |
| 19 | 0.9119 | 0.9110 | 0.7757 | 0.7684 |
| 20 | 0.9099 | 0.9084 | 0.7841 | 0.7758 |
| 21 | 0.9072 | 0.9046 | 0.7907 | 0.7822 |
| 22 | 0.9047 | 0.9005 | 0.7970 | 0.7897 |
| 23 | 0.9008 | 0.8956 | 0.8031 | 0.7968 |
| 24 | 0.8949 | 0.8903 | 0.8085 | 0.8016 |
| 25 | 0.8904 | 0.8853 | 0.8108 | 0.8052 |
| 26 | 0.8853 | 0.8809 | 0.8142 | 0.8088 |
| 27 | 0.8806 | 0.8764 | 0.8178 | 0.8102 |
| 28 | 0.8766 | 0.8729 | 0.8211 | 0.8120 |
| 29 | 0.8726 | 0.8691 | 0.8213 | 0.8129 |
| 30 | 0.8697 | 0.8656 | **0.8229** | **0.8144** |

**Table 6.18: Accuracies achieved by applying $R_{\leq q}$-Measure to 20Newsgroups for each protocol.**

|  | Concatenated Dynamic | Concatenated Static | NonConcatenated Dynamic | NonConcatenated Static |
|---|---|---|---|---|
| 1 | **0.9070** | **0.9086** | 0.8033 | 0.7951 |
| 2 | **0.9070** | **0.9086** | 0.8030 | 0.7953 |
| 3 | **0.9070** | **0.9086** | 0.8045 | 0.7981 |
| 4 | 0.9066 | 0.9084 | 0.8127 | 0.8054 |
| 5 | 0.9063 | 0.9078 | 0.8218 | 0.8152 |
| 6 | 0.9048 | 0.9058 | 0.8295 | 0.8224 |
| 7 | 0.9023 | 0.9032 | 0.8330 | 0.8265 |
| 8 | 0.8994 | 0.9005 | 0.8378 | **0.8292** |
| 9 | 0.8969 | 0.8976 | **0.8383** | 0.8290 |
| 10 | 0.8932 | 0.8948 | 0.8377 | 0.8282 |
| 11 | 0.8896 | 0.8911 | 0.8372 | 0.8274 |
| 12 | 0.8858 | 0.8868 | 0.8357 | 0.8264 |
| 13 | 0.8807 | 0.8816 | 0.8344 | 0.8257 |
| 14 | 0.8761 | 0.8776 | 0.8336 | 0.8247 |
| 15 | 0.8721 | 0.8738 | 0.8332 | 0.8246 |
| 16 | 0.8684 | 0.8701 | 0.8323 | 0.8236 |
| 17 | 0.8649 | 0.8663 | 0.8314 | 0.8233 |
| 18 | 0.8627 | 0.8641 | 0.8306 | 0.8222 |
| 19 | 0.8595 | 0.8611 | 0.8298 | 0.8219 |
| 20 | 0.8569 | 0.8581 | 0.8288 | 0.8210 |
| 21 | 0.8548 | 0.8564 | 0.8281 | 0.8205 |
| 22 | 0.8521 | 0.8537 | 0.8274 | 0.8195 |
| 23 | 0.8495 | 0.8511 | 0.8263 | 0.8179 |
| 24 | 0.8460 | 0.8474 | 0.8247 | 0.8151 |
| 25 | 0.8432 | 0.8447 | 0.8235 | 0.8135 |
| 26 | 0.8407 | 0.8421 | 0.8221 | 0.8116 |
| 27 | 0.8378 | 0.8393 | 0.8206 | 0.8092 |
| 28 | 0.8358 | 0.8376 | 0.8196 | 0.8076 |
| 29 | 0.8331 | 0.8350 | 0.8181 | 0.8056 |
| 30 | 0.8303 | 0.8325 | 0.8162 | 0.8038 |

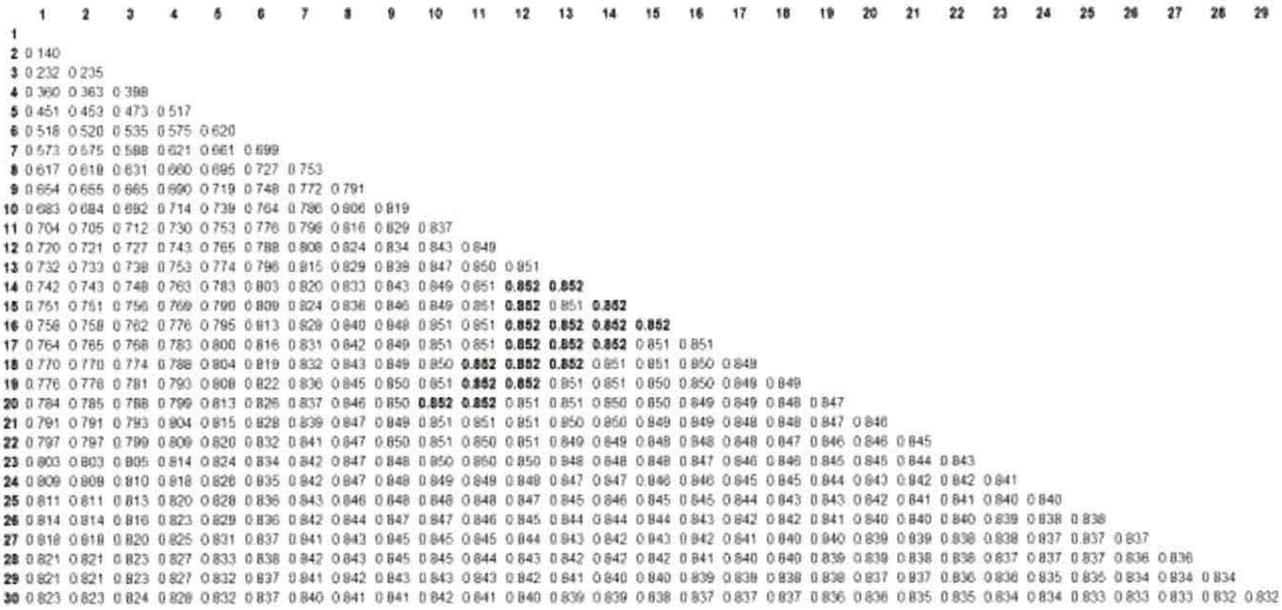
**Table 6.19: Accuracies achieved by applying $R_{\geq q}$-Measure to 20Newsgroups for each protocol.**

|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **1** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **2** | 0.238 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **3** | 0.780 | 0.780 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **4** | 0.873 | 0.873 | 0.874 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **5** | 0.894 | 0.894 | 0.894 | 0.896 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **6** | 0.903 | 0.903 | 0.904 | 0.903 | 0.903 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **7** | 0.906 | 0.906 | 0.906 | 0.905 | 0.906 | 0.905 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **8** | 0.908 | 0.908 | 0.908 | 0.908 | 0.908 | 0.907 | 0.906 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **9** | 0.909 | 0.909 | 0.910 | 0.910 | 0.909 | 0.908 | 0.907 | 0.906 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **10** | 0.911 | 0.911 | 0.911 | 0.910 | 0.910 | 0.909 | 0.908 | 0.907 | 0.907 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **11** | 0.911 | 0.911 | 0.911 | 0.911 | 0.911 | 0.910 | 0.909 | 0.908 | 0.908 | 0.906 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **12** | 0.913 | 0.913 | 0.913 | 0.913 | 0.913 | 0.911 | 0.911 | 0.909 | 0.908 | 0.907 | 0.906 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **13** | 0.913 | 0.913 | 0.913 | 0.913 | 0.913 | 0.912 | 0.911 | 0.909 | 0.908 | 0.907 | 0.905 | 0.903 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **14** | 0.914 | 0.914 | 0.914 | 0.914 | 0.914 | 0.913 | 0.911 | 0.909 | 0.908 | 0.906 | 0.904 | 0.903 | 0.900 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **15** | **0.915** | **0.915** | **0.915** | **0.915** | 0.914 | 0.913 | 0.911 | 0.909 | 0.908 | 0.905 | 0.903 | 0.901 | 0.900 | 0.898 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **16** | 0.914 | 0.914 | 0.914 | 0.914 | 0.913 | 0.912 | 0.911 | 0.909 | 0.907 | 0.904 | 0.903 | 0.900 | 0.897 | 0.894 | 0.891 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **17** | 0.913 | 0.913 | 0.914 | 0.913 | 0.912 | 0.911 | 0.909 | 0.907 | 0.906 | 0.904 | 0.902 | 0.899 | 0.896 | 0.893 | 0.890 | 0.887 |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **18** | 0.913 | 0.913 | 0.913 | 0.912 | 0.912 | 0.911 | 0.909 | 0.907 | 0.905 | 0.903 | 0.901 | 0.897 | 0.895 | 0.891 | 0.888 | 0.885 | 0.882 |  |  |  |  |  |  |  |  |  |  |  |  |
| **19** | 0.912 | 0.912 | 0.912 | 0.912 | 0.911 | 0.909 | 0.908 | 0.906 | 0.904 | 0.902 | 0.900 | 0.897 | 0.893 | 0.890 | 0.886 | 0.884 | 0.880 | 0.877 |  |  |  |  |  |  |  |  |  |  |  |
| **20** | 0.910 | 0.910 | 0.910 | 0.910 | 0.909 | 0.908 | 0.906 | 0.905 | 0.902 | 0.901 | 0.898 | 0.895 | 0.892 | 0.888 | 0.885 | 0.882 | 0.879 | 0.875 | 0.873 |  |  |  |  |  |  |  |  |  |  |
| **21** | 0.907 | 0.907 | 0.907 | 0.907 | 0.906 | 0.905 | 0.903 | 0.902 | 0.900 | 0.898 | 0.895 | 0.893 | 0.889 | 0.886 | 0.883 | 0.880 | 0.876 | 0.873 | 0.871 | 0.870 |  |  |  |  |  |  |  |  |  |
| **22** | 0.905 | 0.905 | 0.904 | 0.904 | 0.904 | 0.902 | 0.901 | 0.900 | 0.898 | 0.896 | 0.893 | 0.891 | 0.887 | 0.884 | 0.881 | 0.877 | 0.874 | 0.872 | 0.870 | 0.868 | 0.866 |  |  |  |  |  |  |  |  |
| **23** | 0.901 | 0.901 | 0.901 | 0.901 | 0.900 | 0.899 | 0.898 | 0.896 | 0.894 | 0.892 | 0.889 | 0.887 | 0.884 | 0.881 | 0.878 | 0.874 | 0.872 | 0.870 | 0.867 | 0.866 | 0.863 | 0.862 |  |  |  |  |  |  |  |
| **24** | 0.895 | 0.895 | 0.896 | 0.896 | 0.894 | 0.893 | 0.892 | 0.891 | 0.889 | 0.888 | 0.885 | 0.883 | 0.881 | 0.878 | 0.874 | 0.872 | 0.869 | 0.867 | 0.865 | 0.863 | 0.861 | 0.860 | 0.858 |  |  |  |  |  |  |
| **25** | 0.890 | 0.890 | 0.890 | 0.890 | 0.890 | 0.889 | 0.888 | 0.887 | 0.885 | 0.883 | 0.882 | 0.880 | 0.877 | 0.874 | 0.871 | 0.869 | 0.866 | 0.864 | 0.862 | 0.860 | 0.859 | 0.858 | 0.856 | 0.856 |  |  |  |  |  |
| **26** | 0.885 | 0.885 | 0.885 | 0.885 | 0.885 | 0.884 | 0.883 | 0.882 | 0.881 | 0.879 | 0.877 | 0.875 | 0.873 | 0.870 | 0.868 | 0.866 | 0.863 | 0.861 | 0.860 | 0.858 | 0.856 | 0.855 | 0.854 | 0.854 | 0.853 |  |  |  |  |
| **27** | 0.881 | 0.881 | 0.881 | 0.880 | 0.880 | 0.880 | 0.879 | 0.878 | 0.877 | 0.875 | 0.873 | 0.871 | 0.869 | 0.867 | 0.864 | 0.862 | 0.860 | 0.858 | 0.856 | 0.855 | 0.853 | 0.852 | 0.852 | 0.851 | 0.851 | 0.849 |  |  |  |
| **28** | 0.877 | 0.877 | 0.877 | 0.877 | 0.876 | 0.876 | 0.876 | 0.875 | 0.873 | 0.872 | 0.870 | 0.868 | 0.866 | 0.864 | 0.861 | 0.859 | 0.857 | 0.856 | 0.854 | 0.853 | 0.851 | 0.851 | 0.850 | 0.850 | 0.849 | 0.847 | 0.846 |  |  |
| **29** | 0.873 | 0.873 | 0.873 | 0.872 | 0.872 | 0.872 | 0.872 | 0.871 | 0.869 | 0.868 | 0.866 | 0.864 | 0.862 | 0.860 | 0.858 | 0.857 | 0.854 | 0.853 | 0.851 | 0.850 | 0.849 | 0.848 | 0.848 | 0.847 | 0.846 | 0.844 | 0.844 | 0.843 |  |
| **30** | 0.870 | 0.870 | 0.870 | 0.870 | 0.869 | 0.869 | 0.868 | 0.868 | 0.867 | 0.866 | 0.863 | 0.861 | 0.860 | 0.858 | 0.856 | 0.854 | 0.852 | 0.851 | 0.849 | 0.848 | 0.847 | 0.846 | 0.846 | 0.844 | 0.843 | 0.842 | 0.841 | 0.840 | 0.840 |

**Table 6.20: R-Range average accuracies for 20Newsgroups, Concatenated Dynamic. The lower range value is shown across the columns and the upper range value shown across the rows.**

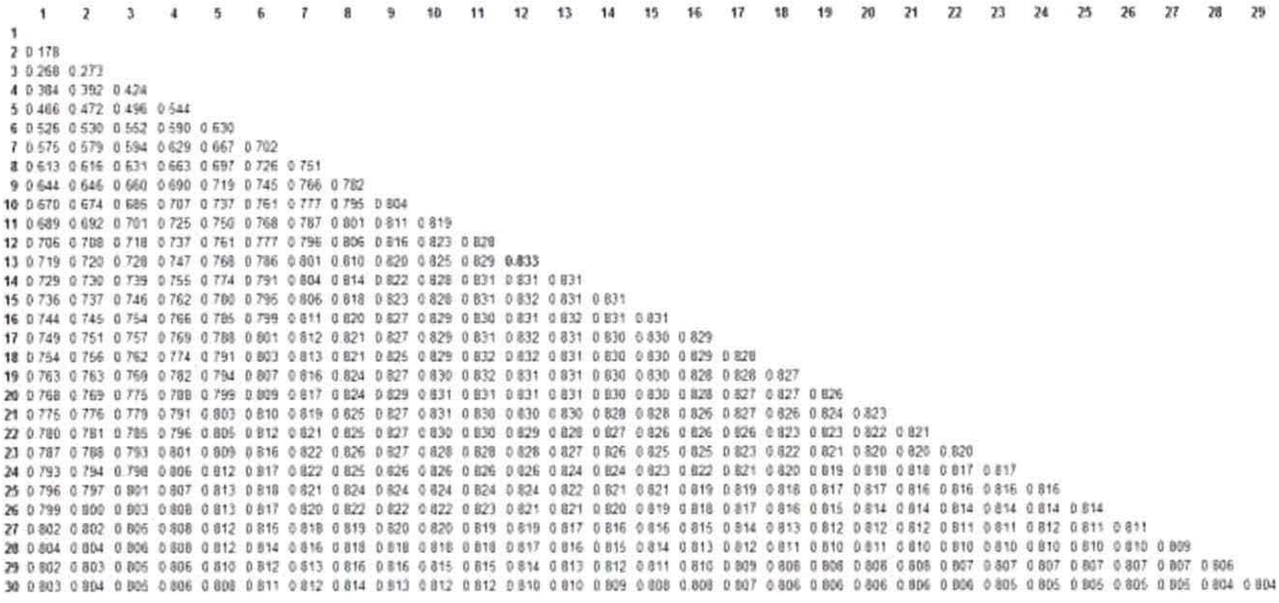
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **1** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **2** | 0.244 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **3** | 0.783 | 0.783 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **4** | 0.872 | 0.872 | 0.873 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **5** | 0.895 | 0.895 | 0.895 | 0.897 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **6** | 0.903 | 0.903 | 0.903 | 0.903 | 0.904 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **7** | 0.906 | 0.906 | 0.906 | 0.907 | 0.907 | 0.906 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **8** | 0.908 | 0.908 | 0.908 | 0.908 | 0.908 | 0.907 | 0.907 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **9** | 0.910 | 0.910 | 0.910 | 0.909 | 0.909 | 0.909 | 0.908 | 0.906 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **10** | 0.911 | 0.911 | 0.911 | 0.911 | 0.910 | 0.909 | 0.908 | 0.908 | 0.907 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **11** | 0.911 | 0.911 | 0.911 | 0.911 | 0.911 | 0.909 | 0.909 | 0.908 | 0.907 | 0.906 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **12** | 0.912 | 0.912 | 0.912 | 0.912 | 0.911 | 0.910 | 0.910 | 0.909 | 0.908 | 0.906 | 0.905 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **13** | 0.912 | 0.912 | 0.912 | 0.912 | 0.912 | 0.911 | 0.911 | 0.909 | 0.907 | 0.906 | 0.904 | 0.903 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **14** | 0.913 | 0.913 | 0.913 | 0.913 | 0.913 | 0.911 | 0.911 | 0.909 | 0.908 | 0.905 | 0.903 | 0.902 | 0.899 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **15** | 0.913 | 0.913 | 0.913 | **0.914** | 0.913 | 0.912 | 0.911 | 0.909 | 0.907 | 0.905 | 0.903 | 0.901 | 0.898 | 0.895 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **16** | 0.913 | 0.913 | 0.913 | 0.913 | 0.913 | 0.912 | 0.911 | 0.908 | 0.907 | 0.904 | 0.902 | 0.899 | 0.896 | 0.893 | 0.890 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **17** | 0.913 | 0.912 | 0.912 | 0.912 | 0.912 | 0.911 | 0.909 | 0.907 | 0.906 | 0.903 | 0.901 | 0.898 | 0.895 | 0.892 | 0.888 | 0.886 |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **18** | 0.912 | 0.912 | 0.912 | 0.911 | 0.911 | 0.910 | 0.908 | 0.907 | 0.905 | 0.903 | 0.900 | 0.896 | 0.893 | 0.891 | 0.887 | 0.884 | 0.881 |  |  |  |  |  |  |  |  |  |  |  |  |
| **19** | 0.911 | 0.911 | 0.911 | 0.911 | 0.910 | 0.909 | 0.907 | 0.906 | 0.903 | 0.902 | 0.899 | 0.895 | 0.893 | 0.888 | 0.885 | 0.882 | 0.879 | 0.876 |  |  |  |  |  |  |  |  |  |  |  |
| **20** | 0.908 | 0.908 | 0.909 | 0.908 | 0.908 | 0.907 | 0.905 | 0.904 | 0.901 | 0.900 | 0.897 | 0.893 | 0.890 | 0.886 | 0.884 | 0.880 | 0.877 | 0.874 | 0.872 |  |  |  |  |  |  |  |  |  |  |
| **21** | 0.905 | 0.905 | 0.905 | 0.905 | 0.904 | 0.903 | 0.902 | 0.901 | 0.899 | 0.897 | 0.893 | 0.891 | 0.888 | 0.885 | 0.881 | 0.878 | 0.875 | 0.872 | 0.870 | 0.869 |  |  |  |  |  |  |  |  |  |
| **22** | 0.900 | 0.900 | 0.901 | 0.901 | 0.900 | 0.899 | 0.898 | 0.897 | 0.895 | 0.894 | 0.891 | 0.888 | 0.885 | 0.882 | 0.879 | 0.875 | 0.873 | 0.871 | 0.869 | 0.867 | 0.865 |  |  |  |  |  |  |  |  |
| **23** | 0.896 | 0.896 | 0.896 | 0.896 | 0.895 | 0.894 | 0.894 | 0.892 | 0.891 | 0.889 | 0.886 | 0.884 | 0.881 | 0.879 | 0.875 | 0.872 | 0.870 | 0.868 | 0.866 | 0.864 | 0.862 | 0.861 |  |  |  |  |  |  |  |
| **24** | 0.890 | 0.890 | 0.890 | 0.890 | 0.890 | 0.889 | 0.889 | 0.887 | 0.886 | 0.884 | 0.882 | 0.880 | 0.878 | 0.875 | 0.872 | 0.869 | 0.867 | 0.865 | 0.863 | 0.861 | 0.859 | 0.858 | 0.857 |  |  |  |  |  |  |
| **25** | 0.885 | 0.885 | 0.885 | 0.885 | 0.885 | 0.884 | 0.884 | 0.882 | 0.881 | 0.880 | 0.878 | 0.876 | 0.874 | 0.871 | 0.868 | 0.866 | 0.864 | 0.862 | 0.860 | 0.859 | 0.857 | 0.856 | 0.855 | 0.855 |  |  |  |  |  |
| **26** | 0.881 | 0.881 | 0.881 | 0.881 | 0.880 | 0.880 | 0.879 | 0.878 | 0.877 | 0.876 | 0.874 | 0.872 | 0.870 | 0.867 | 0.865 | 0.863 | 0.861 | 0.859 | 0.858 | 0.856 | 0.855 | 0.853 | 0.853 | 0.852 | 0.851 |  |  |  |  |
| **27** | 0.876 | 0.876 | 0.876 | 0.876 | 0.876 | 0.876 | 0.875 | 0.874 | 0.873 | 0.872 | 0.870 | 0.868 | 0.866 | 0.864 | 0.861 | 0.859 | 0.857 | 0.856 | 0.855 | 0.853 | 0.852 | 0.851 | 0.850 | 0.849 | 0.848 | 0.847 |  |  |  |
| **28** | 0.873 | 0.873 | 0.873 | 0.873 | 0.873 | 0.872 | 0.872 | 0.871 | 0.870 | 0.869 | 0.866 | 0.865 | 0.863 | 0.861 | 0.858 | 0.856 | 0.855 | 0.854 | 0.852 | 0.851 | 0.850 | 0.849 | 0.848 | 0.848 | 0.846 | 0.845 | 0.845 |  |  |
| **29** | 0.869 | 0.869 | 0.869 | 0.869 | 0.869 | 0.868 | 0.868 | 0.867 | 0.866 | 0.865 | 0.863 | 0.861 | 0.860 | 0.858 | 0.856 | 0.853 | 0.852 | 0.851 | 0.849 | 0.848 | 0.847 | 0.846 | 0.845 | 0.844 | 0.844 | 0.843 | 0.842 | 0.841 |  |
| **30** | 0.866 | 0.866 | 0.866 | 0.866 | 0.866 | 0.865 | 0.865 | 0.864 | 0.863 | 0.862 | 0.860 | 0.859 | 0.857 | 0.855 | 0.852 | 0.851 | 0.850 | 0.849 | 0.847 | 0.845 | 0.845 | 0.844 | 0.843 | 0.842 | 0.841 | 0.841 | 0.839 | 0.839 | 0.838 |

**Table 6.21: R-Range average accuracies for 20Newsgroups, Concatenated Static. The lower range value is shown across the columns and the upper range value shown across the rows.**



| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **1** | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| **2** | 0.140 | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| **3** | 0.232 | 0.235 | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| **4** | 0.360 | 0.363 | 0.398 | | | | | | | | | | | | | | | | | | | | | | | | | | |
| **5** | 0.451 | 0.453 | 0.473 | 0.517 | | | | | | | | | | | | | | | | | | | | | | | | | |
| **6** | 0.518 | 0.520 | 0.535 | 0.575 | 0.620 | | | | | | | | | | | | | | | | | | | | | | | | |
| **7** | 0.573 | 0.575 | 0.588 | 0.621 | 0.661 | 0.699 | | | | | | | | | | | | | | | | | | | | | | | |
| **8** | 0.617 | 0.618 | 0.631 | 0.660 | 0.695 | 0.727 | 0.753 | | | | | | | | | | | | | | | | | | | | | | |
| **9** | 0.654 | 0.655 | 0.665 | 0.690 | 0.719 | 0.748 | 0.772 | 0.791 | | | | | | | | | | | | | | | | | | | | | |
| **10** | 0.683 | 0.684 | 0.692 | 0.714 | 0.739 | 0.764 | 0.786 | 0.806 | 0.819 | | | | | | | | | | | | | | | | | | | | |
| **11** | 0.704 | 0.705 | 0.712 | 0.730 | 0.753 | 0.776 | 0.796 | 0.816 | 0.829 | 0.837 | | | | | | | | | | | | | | | | | | | |
| **12** | 0.720 | 0.721 | 0.727 | 0.743 | 0.765 | 0.788 | 0.808 | 0.824 | 0.834 | 0.843 | 0.849 | | | | | | | | | | | | | | | | | | |
| **13** | 0.732 | 0.733 | 0.738 | 0.753 | 0.774 | 0.796 | 0.815 | 0.829 | 0.839 | 0.847 | 0.850 | 0.851 | | | | | | | | | | | | | | | | | |
| **14** | 0.742 | 0.743 | 0.748 | 0.763 | 0.783 | 0.803 | 0.820 | 0.833 | 0.843 | 0.849 | 0.851 | **0.852** | **0.852** | | | | | | | | | | | | | | | | |
| **15** | 0.751 | 0.751 | 0.756 | 0.769 | 0.790 | 0.809 | 0.824 | 0.836 | 0.846 | 0.849 | 0.851 | **0.852** | 0.851 | **0.852** | | | | | | | | | | | | | | | |
| **16** | 0.758 | 0.758 | 0.762 | 0.776 | 0.795 | 0.813 | 0.828 | 0.840 | 0.848 | 0.851 | 0.851 | **0.852** | **0.852** | **0.852** | **0.852** | | | | | | | | | | | | | | |
| **17** | 0.764 | 0.765 | 0.768 | 0.783 | 0.800 | 0.816 | 0.831 | 0.842 | 0.849 | 0.851 | 0.851 | **0.852** | **0.852** | **0.852** | 0.851 | 0.851 | | | | | | | | | | | | | |
| **18** | 0.770 | 0.770 | 0.774 | 0.788 | 0.804 | 0.819 | 0.832 | 0.843 | 0.849 | 0.850 | **0.852** | **0.852** | **0.852** | 0.851 | 0.851 | 0.850 | 0.849 | | | | | | | | | | | | |
| **19** | 0.776 | 0.776 | 0.781 | 0.790 | 0.808 | 0.822 | 0.836 | 0.845 | 0.850 | 0.851 | **0.852** | **0.852** | 0.851 | 0.851 | 0.850 | 0.850 | 0.849 | 0.849 | | | | | | | | | | | |
| **20** | 0.784 | 0.785 | 0.788 | 0.799 | 0.813 | 0.826 | 0.837 | 0.846 | 0.850 | **0.852** | **0.852** | 0.851 | 0.851 | 0.850 | 0.850 | 0.849 | 0.849 | 0.848 | 0.847 | | | | | | | | | | |
| **21** | 0.791 | 0.791 | 0.793 | 0.804 | 0.815 | 0.828 | 0.839 | 0.847 | 0.849 | 0.851 | 0.851 | 0.851 | 0.850 | 0.850 | 0.849 | 0.849 | 0.848 | 0.848 | 0.847 | 0.846 | | | | | | | | | |
| **22** | 0.797 | 0.797 | 0.799 | 0.809 | 0.820 | 0.832 | 0.841 | 0.847 | 0.850 | 0.851 | 0.850 | 0.851 | 0.849 | 0.849 | 0.848 | 0.848 | 0.848 | 0.847 | 0.846 | 0.846 | 0.845 | | | | | | | | |
| **23** | 0.803 | 0.803 | 0.805 | 0.814 | 0.824 | 0.834 | 0.842 | 0.847 | 0.848 | 0.850 | 0.850 | 0.850 | 0.848 | 0.848 | 0.848 | 0.847 | 0.846 | 0.846 | 0.845 | 0.845 | 0.844 | 0.843 | | | | | | | |
| **24** | 0.809 | 0.809 | 0.810 | 0.818 | 0.826 | 0.835 | 0.842 | 0.847 | 0.848 | 0.849 | 0.849 | 0.848 | 0.847 | 0.847 | 0.846 | 0.846 | 0.845 | 0.845 | 0.844 | 0.843 | 0.842 | 0.842 | 0.841 | | | | | | |
| **25** | 0.811 | 0.811 | 0.813 | 0.820 | 0.828 | 0.836 | 0.843 | 0.846 | 0.848 | 0.848 | 0.848 | 0.847 | 0.845 | 0.846 | 0.845 | 0.845 | 0.844 | 0.843 | 0.843 | 0.842 | 0.841 | 0.841 | 0.840 | 0.840 | | | | | |
| **26** | 0.814 | 0.814 | 0.816 | 0.823 | 0.829 | 0.836 | 0.842 | 0.844 | 0.847 | 0.847 | 0.846 | 0.845 | 0.844 | 0.844 | 0.844 | 0.843 | 0.842 | 0.842 | 0.841 | 0.840 | 0.840 | 0.840 | 0.839 | 0.838 | 0.838 | | | | |
| **27** | 0.818 | 0.818 | 0.820 | 0.825 | 0.831 | 0.837 | 0.841 | 0.843 | 0.845 | 0.845 | 0.845 | 0.844 | 0.843 | 0.842 | 0.843 | 0.842 | 0.841 | 0.840 | 0.840 | 0.839 | 0.839 | 0.838 | 0.838 | 0.837 | 0.837 | 0.837 | | | |
| **28** | 0.821 | 0.821 | 0.823 | 0.827 | 0.833 | 0.838 | 0.842 | 0.843 | 0.845 | 0.845 | 0.844 | 0.843 | 0.842 | 0.842 | 0.842 | 0.841 | 0.840 | 0.840 | 0.839 | 0.839 | 0.838 | 0.838 | 0.837 | 0.837 | 0.837 | 0.836 | 0.836 | | |
| **29** | 0.821 | 0.821 | 0.823 | 0.827 | 0.832 | 0.837 | 0.841 | 0.842 | 0.843 | 0.843 | 0.843 | 0.842 | 0.841 | 0.840 | 0.840 | 0.839 | 0.838 | 0.838 | 0.838 | 0.837 | 0.837 | 0.836 | 0.836 | 0.835 | 0.835 | 0.834 | 0.834 | 0.834 | |
| **30** | 0.823 | 0.823 | 0.824 | 0.828 | 0.832 | 0.837 | 0.840 | 0.841 | 0.841 | 0.842 | 0.841 | 0.840 | 0.839 | 0.839 | 0.838 | 0.837 | 0.837 | 0.837 | 0.836 | 0.836 | 0.835 | 0.835 | 0.834 | 0.834 | 0.833 | 0.833 | 0.833 | 0.832 | 0.832 |

**Table 6.22: R-Range average accuracies for 20Newsgroups, Non-concatenated Dynamic. . The lower range value is shown across the columns and the upper range value shown across the rows.**



| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **1** | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| **2** | 0.178 | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| **3** | 0.268 | 0.273 | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| **4** | 0.384 | 0.392 | 0.424 | | | | | | | | | | | | | | | | | | | | | | | | | | |
| **5** | 0.466 | 0.472 | 0.496 | 0.544 | | | | | | | | | | | | | | | | | | | | | | | | | |
| **6** | 0.526 | 0.530 | 0.552 | 0.590 | 0.630 | | | | | | | | | | | | | | | | | | | | | | | | |
| **7** | 0.575 | 0.579 | 0.594 | 0.629 | 0.667 | 0.702 | | | | | | | | | | | | | | | | | | | | | | | |
| **8** | 0.613 | 0.616 | 0.631 | 0.663 | 0.697 | 0.726 | 0.751 | | | | | | | | | | | | | | | | | | | | | | |
| **9** | 0.644 | 0.646 | 0.660 | 0.690 | 0.719 | 0.745 | 0.766 | 0.782 | | | | | | | | | | | | | | | | | | | | | |
| **10** | 0.670 | 0.674 | 0.686 | 0.707 | 0.737 | 0.761 | 0.777 | 0.795 | 0.804 | | | | | | | | | | | | | | | | | | | | |
| **11** | 0.689 | 0.692 | 0.701 | 0.725 | 0.750 | 0.768 | 0.787 | 0.801 | 0.811 | 0.819 | | | | | | | | | | | | | | | | | | | |
| **12** | 0.706 | 0.708 | 0.718 | 0.737 | 0.761 | 0.777 | 0.796 | 0.806 | 0.816 | 0.823 | 0.828 | | | | | | | | | | | | | | | | | | |
| **13** | 0.719 | 0.720 | 0.728 | 0.747 | 0.768 | 0.786 | 0.801 | 0.810 | 0.820 | 0.825 | 0.829 | **0.833** | | | | | | | | | | | | | | | | | |
| **14** | 0.729 | 0.730 | 0.739 | 0.755 | 0.774 | 0.791 | 0.804 | 0.814 | 0.822 | 0.828 | 0.831 | 0.831 | 0.831 | | | | | | | | | | | | | | | | |
| **15** | 0.736 | 0.737 | 0.746 | 0.762 | 0.780 | 0.796 | 0.806 | 0.818 | 0.823 | 0.828 | 0.831 | 0.832 | 0.831 | 0.831 | | | | | | | | | | | | | | | |
| **16** | 0.744 | 0.745 | 0.754 | 0.766 | 0.785 | 0.799 | 0.811 | 0.820 | 0.827 | 0.829 | 0.830 | 0.831 | 0.832 | 0.831 | 0.831 | | | | | | | | | | | | | | |
| **17** | 0.749 | 0.751 | 0.757 | 0.769 | 0.788 | 0.801 | 0.812 | 0.821 | 0.827 | 0.829 | 0.831 | 0.832 | 0.831 | 0.830 | 0.830 | 0.829 | | | | | | | | | | | | | |
| **18** | 0.754 | 0.756 | 0.762 | 0.774 | 0.791 | 0.803 | 0.813 | 0.821 | 0.825 | 0.829 | 0.832 | 0.832 | 0.831 | 0.830 | 0.830 | 0.829 | 0.828 | | | | | | | | | | | | |
| **19** | 0.763 | 0.763 | 0.769 | 0.782 | 0.794 | 0.807 | 0.816 | 0.824 | 0.827 | 0.830 | 0.832 | 0.831 | 0.831 | 0.830 | 0.830 | 0.828 | 0.828 | 0.827 | | | | | | | | | | | |
| **20** | 0.768 | 0.769 | 0.775 | 0.788 | 0.799 | 0.809 | 0.817 | 0.824 | 0.829 | 0.831 | 0.831 | 0.831 | 0.831 | 0.830 | 0.830 | 0.828 | 0.827 | 0.827 | 0.826 | | | | | | | | | | |
| **21** | 0.775 | 0.776 | 0.779 | 0.791 | 0.803 | 0.810 | 0.819 | 0.825 | 0.827 | 0.831 | 0.830 | 0.830 | 0.830 | 0.828 | 0.828 | 0.826 | 0.827 | 0.826 | 0.824 | 0.823 | | | | | | | | | |
| **22** | 0.780 | 0.781 | 0.785 | 0.796 | 0.805 | 0.812 | 0.821 | 0.825 | 0.827 | 0.830 | 0.830 | 0.829 | 0.828 | 0.827 | 0.826 | 0.826 | 0.826 | 0.823 | 0.823 | 0.822 | 0.821 | | | | | | | | |
| **23** | 0.787 | 0.788 | 0.793 | 0.801 | 0.809 | 0.816 | 0.822 | 0.826 | 0.827 | 0.828 | 0.828 | 0.828 | 0.827 | 0.826 | 0.825 | 0.825 | 0.823 | 0.822 | 0.821 | 0.820 | 0.820 | 0.820 | | | | | | | |
| **24** | 0.793 | 0.794 | 0.798 | 0.806 | 0.812 | 0.817 | 0.822 | 0.825 | 0.826 | 0.826 | 0.826 | 0.826 | 0.824 | 0.824 | 0.823 | 0.822 | 0.821 | 0.820 | 0.819 | 0.818 | 0.818 | 0.817 | 0.817 | | | | | | |
| **25** | 0.796 | 0.797 | 0.801 | 0.807 | 0.813 | 0.818 | 0.821 | 0.824 | 0.824 | 0.824 | 0.824 | 0.824 | 0.822 | 0.821 | 0.821 | 0.819 | 0.819 | 0.818 | 0.817 | 0.817 | 0.816 | 0.816 | 0.816 | 0.816 | | | | | |
| **26** | 0.799 | 0.800 | 0.803 | 0.808 | 0.813 | 0.817 | 0.820 | 0.822 | 0.822 | 0.822 | 0.823 | 0.821 | 0.821 | 0.820 | 0.819 | 0.818 | 0.817 | 0.816 | 0.815 | 0.814 | 0.814 | 0.814 | 0.814 | 0.814 | 0.814 | | | | |
| **27** | 0.802 | 0.802 | 0.806 | 0.808 | 0.812 | 0.816 | 0.818 | 0.819 | 0.820 | 0.820 | 0.819 | 0.819 | 0.817 | 0.816 | 0.816 | 0.815 | 0.814 | 0.813 | 0.812 | 0.812 | 0.812 | 0.811 | 0.811 | 0.812 | 0.811 | 0.811 | | | |
| **28** | 0.804 | 0.804 | 0.806 | 0.808 | 0.812 | 0.814 | 0.816 | 0.818 | 0.818 | 0.818 | 0.818 | 0.817 | 0.816 | 0.815 | 0.814 | 0.813 | 0.812 | 0.811 | 0.810 | 0.811 | 0.810 | 0.810 | 0.810 | 0.810 | 0.810 | 0.810 | 0.809 | | |
| **29** | 0.802 | 0.803 | 0.805 | 0.806 | 0.810 | 0.812 | 0.813 | 0.816 | 0.816 | 0.815 | 0.815 | 0.814 | 0.813 | 0.812 | 0.811 | 0.810 | 0.809 | 0.808 | 0.808 | 0.808 | 0.808 | 0.807 | 0.807 | 0.807 | 0.807 | 0.807 | 0.807 | 0.806 | |
| **30** | 0.803 | 0.804 | 0.805 | 0.806 | 0.808 | 0.811 | 0.812 | 0.814 | 0.813 | 0.812 | 0.812 | 0.810 | 0.810 | 0.809 | 0.808 | 0.808 | 0.807 | 0.806 | 0.806 | 0.806 | 0.806 | 0.806 | 0.805 | 0.805 | 0.805 | 0.805 | 0.805 | 0.804 | 0.804 |

**Table 6.23: R-Range average accuracies for 20Newsgroups, Non-concatenated Static. The lower range value is shown across the columns and the upper range value shown across the rows.**

| Concatenated Dynamic | Concatenated Static | NonConcatenated Dynamic | NonConcatenated Static |
|---|---|---|---|
| 0.907 | 0.909 | 0.803 | 0.795 |

**Table 6.24: $r^{max}$ average accuracies for 20Newsgroups.**

As with the results for C-Measure in 6.2.1.1 the results for concatenated protocols are similar no matter whether the models are static or dynamic and the same is true for non-concatenated protocols also. The R-Range results also resemble the C-Measure result in that lower ranges again offer better performance for concatenated protocols than for their counterpart. A noticeable difference is that much smaller ranges (difference between minimum range and maximum range) proving better for non-concatenated protocols, the best performing for static models being just a difference of one with 12-13. The best results achieved for concatenated protocols have a range of around ten. The best results for concatenated dynamic models are achieved between ranges 1-15 up to 5-15 having the highest accuracy of 0.915. The best results for concatenated static models are achieved between ranges 1-14 up to 5-16 with 4-15 again providing the highest accuracy, this time 0.914. The best results for non-concatenated dynamic models are achieved between ranges 10-14 up to 15-20 with the highest accuracy of 0.852. The best results for non-concatenated static models are achieved between ranges 11-12 up to 15-16 with 12-13 achieving the highest accuracy, in this case 0.8325. It is also worth noting that in all but one case that R-Ranges outperformed C-Measure.

Table 6.18 shows that for $R_{\leq q}$-Measure, concatenated models performed better than non-concatenated and dynamic models outperform static ones, as was the case with C-Measure for this corpus. The results suggest that a value of between 15 and 16 for q is optimal for concatenated models but a much larger value for non-concatenated, with $q = 30$ achieving the highest accuracy.

Table 6.19 shows that for $R_{\geq q}$-Measure, concatenated models again achieve the highest accuracies, though not as high as for concatenated in 6.18. There is a difference in the optimal value for $q$, which for 6.19 represents the minimum substring length to be included. 6.19 shows that $1 \leq q \leq 3$ is optimal for concatenated models and that a low length of between 8 and 9 is optimal for the non-concatenated models. Table 6.24 also further supports the finding that for 20Newsgroups, concatenated models achieve the highest accuracies. It is however unclear from the $r^{max}$ results whether dynamic or static models performed best, as was the case for $R_{\geq q}$.

### 6.2.3.2 Gutenberg

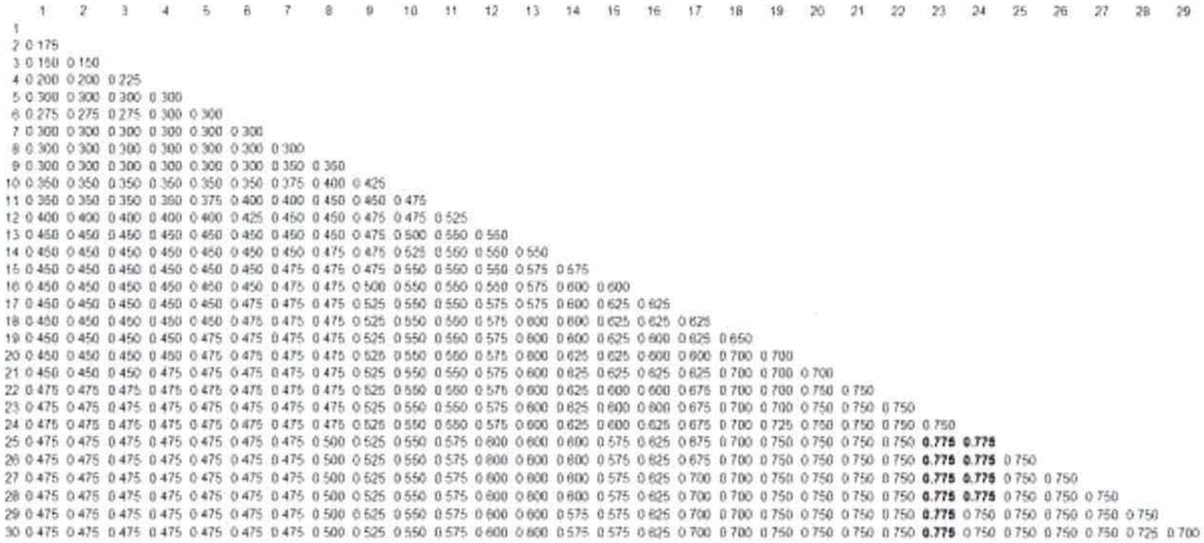
The results for the $R_{\leq q}$-Measure experiments are shown in Table 6.25, $R_{\geq q}$-Measure in Table 6.26, $r^{max}$ in 6.31 and Tables 6.27-6.30 show results for R-Ranges. Tables 6.25 and 6.26 display accuracies for all four protocols with the leftmost column indicating the lower substring limit for each algorithm. Tables 6.27-6.30 show the accuracy for each range with a single table displaying results for a single protocol. The highest accuracies are again highlighted in bold font.

| | Concatenated Dynamic | Concatenated Static | NonConcatenated Dynamic | NonConcatenated Static |
|---|---|---|---|---|
| 1 | 0.075 | 0.075 | 0.175 | 0.125 |
| 2 | 0.175 | 0.150 | 0.125 | 0.250 |
| 3 | 0.150 | 0.175 | 0.150 | 0.200 |
| 4 | 0.200 | 0.225 | 0.175 | 0.175 |
| 5 | 0.300 | 0.325 | 0.150 | 0.250 |
| 6 | 0.275 | 0.375 | 0.225 | 0.275 |
| 7 | 0.300 | 0.400 | 0.250 | 0.275 |
| 8 | 0.300 | 0.425 | 0.275 | 0.325 |
| 9 | 0.300 | 0.425 | 0.300 | 0.375 |
| 10 | 0.350 | 0.475 | 0.300 | 0.375 |
| 11 | 0.350 | 0.500 | 0.300 | 0.400 |
| 12 | 0.400 | 0.500 | 0.350 | 0.400 |
| 13 | 0.450 | 0.500 | 0.375 | 0.475 |
| 14 | 0.450 | **0.525** | 0.375 | 0.475 |
| 15 | 0.450 | **0.525** | 0.375 | 0.475 |
| 16 | 0.450 | **0.525** | **0.425** | 0.475 |
| 17 | 0.450 | **0.525** | **0.425** | 0.475 |
| 18 | 0.450 | **0.525** | **0.425** | 0.475 |
| 19 | 0.450 | **0.525** | **0.425** | 0.475 |
| 20 | 0.450 | **0.525** | **0.425** | 0.475 |
| 21 | 0.450 | **0.525** | **0.425** | 0.475 |
| 22 | **0.475** | **0.525** | **0.425** | 0.475 |
| 23 | **0.475** | **0.525** | **0.425** | 0.475 |
| 24 | **0.475** | **0.525** | **0.425** | 0.475 |
| 25 | **0.475** | **0.525** | **0.425** | 0.475 |
| 26 | **0.475** | **0.525** | **0.425** | 0.475 |
| 27 | **0.475** | **0.525** | **0.425** | 0.500 |
| 28 | **0.475** | **0.525** | **0.425** | 0.500 |
| 29 | **0.475** | **0.525** | **0.425** | 0.500 |
| 30 | **0.475** | **0.525** | **0.425** | **0.525** |

**Table 6.25: Accuracies achieved by applying $R_{\leq q}$-Measure to Gutenberg for each protocol.**

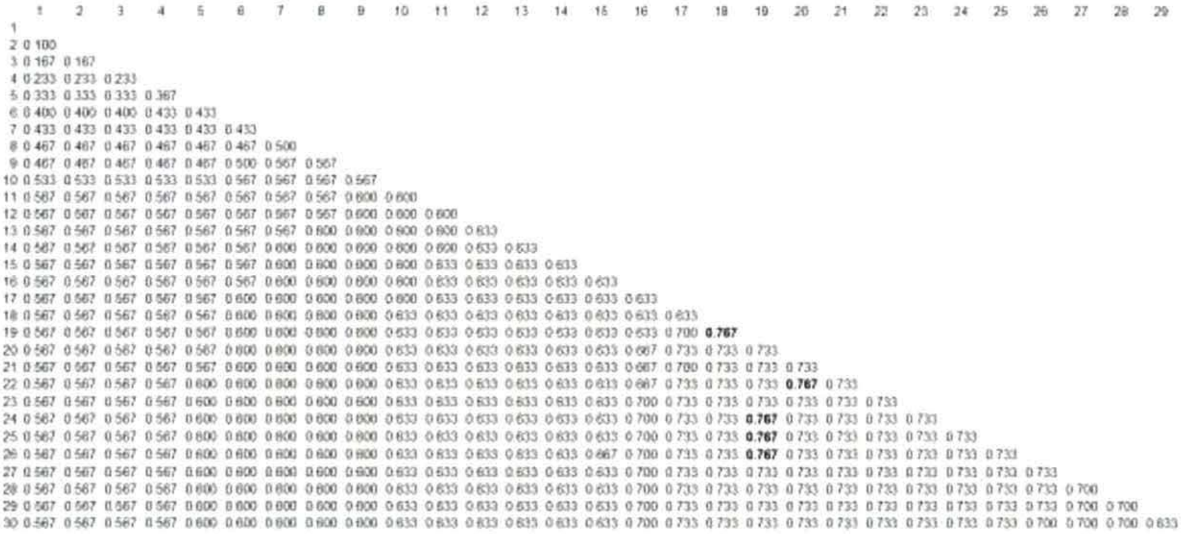
| | **Concatenated Dynamic** | **Concatenated Static** | **NonConcatenated Dynamic** | **NonConcatenated Static** |
|---|---|---|---|---|
| 1 | 0.475 | 0.525 | 0.475 | 0.475 |
| 2 | 0.475 | 0.525 | 0.475 | 0.475 |
| 3 | 0.475 | 0.525 | 0.475 | 0.475 |
| 4 | 0.475 | 0.500 | 0.475 | 0.475 |
| 5 | 0.475 | 0.525 | 0.475 | 0.475 |
| 6 | 0.475 | 0.550 | 0.475 | 0.475 |
| 7 | 0.475 | 0.600 | 0.475 | 0.500 |
| 8 | 0.500 | 0.575 | 0.500 | 0.500 |
| 9 | 0.525 | 0.575 | 0.475 | 0.500 |
| 10 | 0.525 | 0.575 | 0.500 | 0.550 |
| 11 | 0.550 | 0.600 | 0.500 | 0.550 |
| 12 | 0.525 | 0.600 | 0.525 | 0.550 |
| 13 | 0.525 | 0.600 | 0.550 | 0.600 |
| 14 | 0.550 | 0.600 | 0.600 | 0.600 |
| 15 | 0.550 | 0.625 | 0.625 | 0.625 |
| 16 | 0.600 | 0.650 | 0.650 | 0.625 |
| 17 | 0.675 | 0.700 | 0.675 | 0.650 |
| 18 | 0.700 | 0.700 | 0.700 | 0.675 |
| 19 | 0.725 | **0.750** | 0.725 | 0.700 |
| 20 | 0.725 | **0.750** | **0.775** | **0.750** |
| 21 | 0.725 | 0.725 | **0.775** | **0.750** |
| 22 | 0.725 | 0.725 | **0.775** | **0.750** |
| 23 | **0.750** | 0.725 | **0.775** | 0.725 |
| 24 | 0.725 | 0.725 | **0.775** | 0.725 |
| 25 | 0.725 | 0.725 | 0.750 | 0.700 |
| 26 | 0.725 | 0.700 | 0.675 | 0.625 |
| 27 | 0.675 | 0.675 | 0.625 | 0.550 |
| 28 | 0.625 | 0.650 | 0.600 | 0.525 |
| 29 | 0.575 | 0.600 | 0.550 | 0.500 |
| 30 | 0.525 | 0.525 | 0.500 | 0.475 |

**Table 6.26: Accuracies achieved by applying $R_{\geq q}$-Measure to Gutenberg for each protocol.**



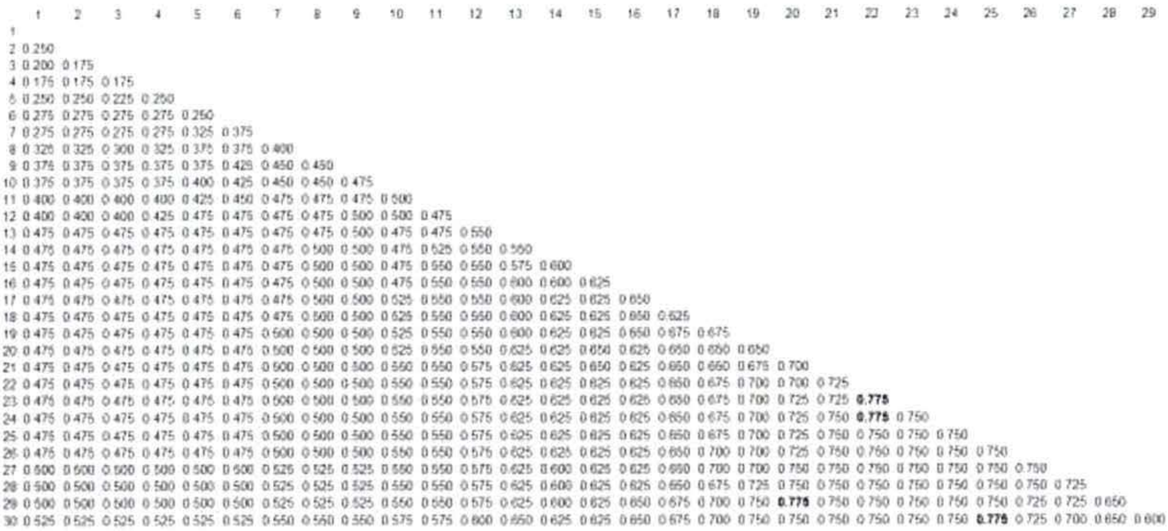
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 2 | 0.175 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 3 | 0.150 | 0.150 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 4 | 0.200 | 0.200 | 0.225 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 5 | 0.300 | 0.300 | 0.300 | 0.300 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 6 | 0.275 | 0.275 | 0.275 | 0.300 | 0.300 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 7 | 0.300 | 0.300 | 0.300 | 0.300 | 0.300 | 0.300 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 8 | 0.300 | 0.300 | 0.300 | 0.300 | 0.300 | 0.300 | 0.300 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 9 | 0.300 | 0.300 | 0.300 | 0.300 | 0.300 | 0.300 | 0.350 | 0.350 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 10 | 0.350 | 0.350 | 0.350 | 0.350 | 0.350 | 0.350 | 0.375 | 0.400 | 0.425 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 11 | 0.350 | 0.350 | 0.350 | 0.350 | 0.375 | 0.400 | 0.400 | 0.450 | 0.450 | 0.475 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 12 | 0.400 | 0.400 | 0.400 | 0.400 | 0.400 | 0.425 | 0.450 | 0.450 | 0.475 | 0.475 | 0.525 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 13 | 0.450 | 0.450 | 0.450 | 0.450 | 0.450 | 0.450 | 0.450 | 0.450 | 0.475 | 0.500 | 0.550 | 0.550 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 14 | 0.450 | 0.450 | 0.450 | 0.450 | 0.450 | 0.450 | 0.450 | 0.475 | 0.475 | 0.525 | 0.550 | 0.550 | 0.550 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 15 | 0.450 | 0.450 | 0.450 | 0.450 | 0.450 | 0.450 | 0.475 | 0.475 | 0.475 | 0.550 | 0.550 | 0.550 | 0.575 | 0.675 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 16 | 0.450 | 0.450 | 0.450 | 0.450 | 0.450 | 0.450 | 0.475 | 0.475 | 0.500 | 0.550 | 0.550 | 0.550 | 0.575 | 0.600 | 0.600 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 17 | 0.450 | 0.450 | 0.450 | 0.450 | 0.450 | 0.475 | 0.475 | 0.475 | 0.525 | 0.550 | 0.550 | 0.575 | 0.575 | 0.600 | 0.625 | 0.625 |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 18 | 0.450 | 0.450 | 0.450 | 0.450 | 0.450 | 0.475 | 0.475 | 0.475 | 0.525 | 0.550 | 0.550 | 0.575 | 0.600 | 0.600 | 0.625 | 0.625 | 0.625 |  |  |  |  |  |  |  |  |  |  |  |  |
| 19 | 0.450 | 0.450 | 0.450 | 0.450 | 0.475 | 0.475 | 0.475 | 0.475 | 0.525 | 0.550 | 0.550 | 0.575 | 0.600 | 0.600 | 0.625 | 0.600 | 0.625 | 0.650 |  |  |  |  |  |  |  |  |  |  |  |
| 20 | 0.450 | 0.450 | 0.450 | 0.450 | 0.475 | 0.475 | 0.475 | 0.475 | 0.525 | 0.550 | 0.550 | 0.575 | 0.600 | 0.625 | 0.625 | 0.600 | 0.600 | 0.700 | 0.700 |  |  |  |  |  |  |  |  |  |  |
| 21 | 0.450 | 0.450 | 0.450 | 0.475 | 0.475 | 0.475 | 0.475 | 0.475 | 0.525 | 0.550 | 0.550 | 0.575 | 0.600 | 0.625 | 0.625 | 0.625 | 0.625 | 0.700 | 0.700 | 0.700 |  |  |  |  |  |  |  |  |  |
| 22 | 0.475 | 0.475 | 0.475 | 0.475 | 0.475 | 0.475 | 0.475 | 0.475 | 0.525 | 0.550 | 0.550 | 0.575 | 0.600 | 0.625 | 0.600 | 0.600 | 0.675 | 0.700 | 0.700 | 0.750 | 0.750 |  |  |  |  |  |  |  |  |
| 23 | 0.475 | 0.475 | 0.475 | 0.475 | 0.475 | 0.475 | 0.475 | 0.475 | 0.525 | 0.550 | 0.550 | 0.575 | 0.600 | 0.625 | 0.600 | 0.600 | 0.675 | 0.700 | 0.700 | 0.750 | 0.750 | 0.750 |  |  |  |  |  |  |  |
| 24 | 0.475 | 0.475 | 0.475 | 0.475 | 0.475 | 0.475 | 0.475 | 0.475 | 0.525 | 0.550 | 0.550 | 0.575 | 0.600 | 0.625 | 0.600 | 0.625 | 0.675 | 0.700 | 0.725 | 0.750 | 0.750 | 0.750 | 0.750 |  |  |  |  |  |  |
| 25 | 0.475 | 0.475 | 0.475 | 0.475 | 0.475 | 0.475 | 0.475 | 0.500 | 0.525 | 0.550 | 0.575 | 0.600 | 0.600 | 0.600 | 0.575 | 0.625 | 0.675 | 0.700 | 0.750 | 0.750 | 0.750 | 0.750 | **0.775** | **0.775** |  |  |  |  |  |
| 26 | 0.475 | 0.475 | 0.475 | 0.475 | 0.475 | 0.475 | 0.475 | 0.500 | 0.525 | 0.550 | 0.575 | 0.600 | 0.600 | 0.600 | 0.575 | 0.625 | 0.675 | 0.700 | 0.750 | 0.750 | 0.750 | 0.750 | **0.775** | **0.775** | 0.750 |  |  |  |  |
| 27 | 0.475 | 0.475 | 0.475 | 0.475 | 0.475 | 0.475 | 0.475 | 0.500 | 0.525 | 0.550 | 0.575 | 0.600 | 0.600 | 0.600 | 0.575 | 0.625 | 0.700 | 0.700 | 0.750 | 0.750 | 0.750 | 0.750 | **0.775** | **0.775** | 0.750 | 0.750 |  |  |  |
| 28 | 0.475 | 0.475 | 0.475 | 0.475 | 0.475 | 0.475 | 0.475 | 0.500 | 0.525 | 0.550 | 0.575 | 0.600 | 0.600 | 0.600 | 0.575 | 0.625 | 0.700 | 0.700 | 0.750 | 0.750 | 0.750 | 0.750 | **0.775** | **0.775** | 0.750 | 0.750 | 0.750 |  |  |
| 29 | 0.475 | 0.475 | 0.475 | 0.475 | 0.475 | 0.475 | 0.475 | 0.500 | 0.525 | 0.550 | 0.575 | 0.600 | 0.600 | 0.575 | 0.575 | 0.625 | 0.700 | 0.700 | 0.750 | 0.750 | 0.750 | 0.750 | **0.775** | 0.750 | 0.750 | 0.750 | 0.750 | 0.750 |  |
| 30 | 0.475 | 0.475 | 0.475 | 0.475 | 0.475 | 0.475 | 0.475 | 0.500 | 0.525 | 0.550 | 0.575 | 0.600 | 0.600 | 0.575 | 0.575 | 0.625 | 0.700 | 0.700 | 0.750 | 0.750 | 0.750 | 0.750 | **0.775** | 0.750 | 0.750 | 0.750 | 0.750 | 0.725 | 0.700 |

**Table 6.27: R-Range average accuracies for Gutenberg, Concatenated Dynamic. The lower range value is shown across the columns and the upper range value shown across the rows.**



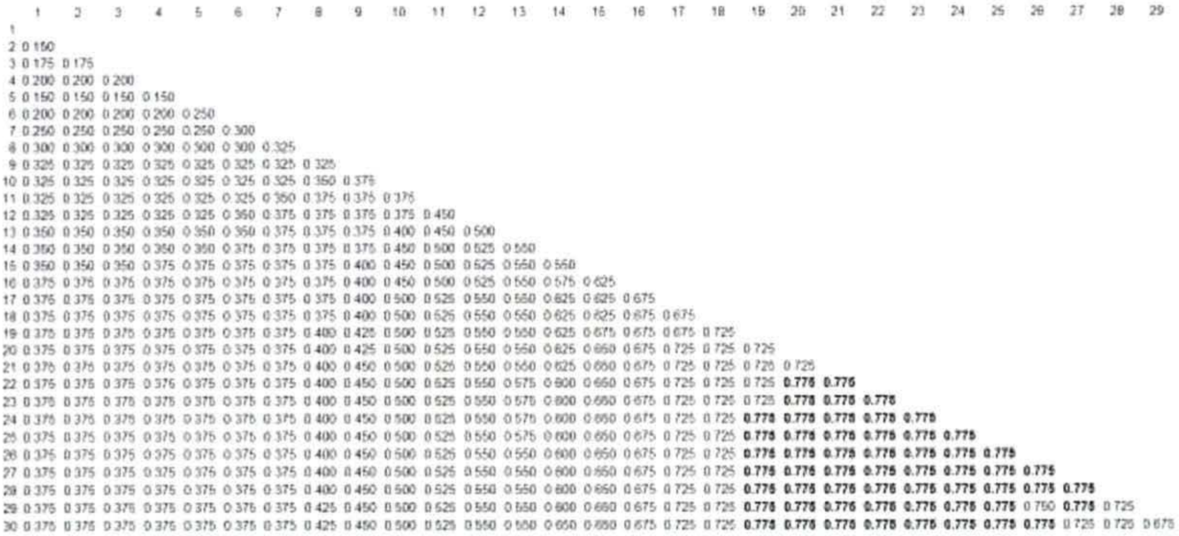
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 2 | 0.100 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 3 | 0.167 | 0.167 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 4 | 0.233 | 0.233 | 0.233 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 5 | 0.333 | 0.333 | 0.333 | 0.367 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 6 | 0.400 | 0.400 | 0.400 | 0.433 | 0.433 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 7 | 0.433 | 0.433 | 0.433 | 0.433 | 0.433 | 0.433 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 8 | 0.467 | 0.467 | 0.467 | 0.467 | 0.467 | 0.467 | 0.500 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 9 | 0.467 | 0.467 | 0.467 | 0.467 | 0.467 | 0.500 | 0.567 | 0.567 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 10 | 0.533 | 0.533 | 0.533 | 0.533 | 0.533 | 0.567 | 0.567 | 0.567 | 0.567 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 11 | 0.567 | 0.567 | 0.567 | 0.567 | 0.567 | 0.567 | 0.567 | 0.567 | 0.600 | 0.600 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 12 | 0.567 | 0.567 | 0.567 | 0.567 | 0.567 | 0.567 | 0.567 | 0.567 | 0.600 | 0.600 | 0.600 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 13 | 0.567 | 0.567 | 0.567 | 0.567 | 0.567 | 0.567 | 0.567 | 0.600 | 0.600 | 0.600 | 0.600 | 0.633 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 14 | 0.567 | 0.567 | 0.567 | 0.567 | 0.567 | 0.567 | 0.600 | 0.600 | 0.600 | 0.600 | 0.600 | 0.633 | 0.633 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 15 | 0.567 | 0.567 | 0.567 | 0.567 | 0.567 | 0.567 | 0.600 | 0.600 | 0.600 | 0.600 | 0.633 | 0.633 | 0.633 | 0.633 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 16 | 0.567 | 0.567 | 0.567 | 0.567 | 0.567 | 0.567 | 0.600 | 0.600 | 0.600 | 0.600 | 0.633 | 0.633 | 0.633 | 0.633 | 0.633 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 17 | 0.567 | 0.567 | 0.567 | 0.567 | 0.567 | 0.600 | 0.600 | 0.600 | 0.600 | 0.600 | 0.633 | 0.633 | 0.633 | 0.633 | 0.633 | 0.633 |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 18 | 0.567 | 0.567 | 0.567 | 0.567 | 0.567 | 0.600 | 0.600 | 0.600 | 0.600 | 0.633 | 0.633 | 0.633 | 0.633 | 0.633 | 0.633 | 0.633 | 0.633 |  |  |  |  |  |  |  |  |  |  |  |  |
| 19 | 0.567 | 0.567 | 0.567 | 0.567 | 0.567 | 0.600 | 0.600 | 0.600 | 0.600 | 0.633 | 0.633 | 0.633 | 0.633 | 0.633 | 0.633 | 0.633 | 0.700 | **0.767** |  |  |  |  |  |  |  |  |  |  |  |
| 20 | 0.567 | 0.567 | 0.567 | 0.567 | 0.567 | 0.600 | 0.600 | 0.600 | 0.600 | 0.633 | 0.633 | 0.633 | 0.633 | 0.633 | 0.633 | 0.667 | 0.733 | 0.733 | 0.733 |  |  |  |  |  |  |  |  |  |  |
| 21 | 0.567 | 0.567 | 0.567 | 0.567 | 0.567 | 0.600 | 0.600 | 0.600 | 0.600 | 0.633 | 0.633 | 0.633 | 0.633 | 0.633 | 0.633 | 0.667 | 0.700 | 0.733 | 0.733 | 0.733 |  |  |  |  |  |  |  |  |  |
| 22 | 0.567 | 0.567 | 0.567 | 0.567 | 0.600 | 0.600 | 0.600 | 0.600 | 0.600 | 0.633 | 0.633 | 0.633 | 0.633 | 0.633 | 0.633 | 0.667 | 0.733 | 0.733 | 0.733 | **0.767** | 0.733 |  |  |  |  |  |  |  |  |
| 23 | 0.567 | 0.567 | 0.567 | 0.567 | 0.600 | 0.600 | 0.600 | 0.600 | 0.600 | 0.633 | 0.633 | 0.633 | 0.633 | 0.633 | 0.633 | 0.700 | 0.733 | 0.733 | 0.733 | 0.733 | 0.733 | 0.733 |  |  |  |  |  |  |  |
| 24 | 0.567 | 0.567 | 0.567 | 0.567 | 0.600 | 0.600 | 0.600 | 0.600 | 0.600 | 0.633 | 0.633 | 0.633 | 0.633 | 0.633 | 0.633 | 0.700 | 0.733 | 0.733 | **0.767** | 0.733 | 0.733 | 0.733 | 0.733 |  |  |  |  |  |  |
| 25 | 0.567 | 0.567 | 0.567 | 0.567 | 0.600 | 0.600 | 0.600 | 0.600 | 0.600 | 0.633 | 0.633 | 0.633 | 0.633 | 0.633 | 0.633 | 0.700 | 0.733 | 0.733 | **0.767** | 0.733 | 0.733 | 0.733 | 0.733 | 0.733 |  |  |  |  |  |
| 26 | 0.567 | 0.567 | 0.567 | 0.567 | 0.600 | 0.600 | 0.600 | 0.600 | 0.600 | 0.633 | 0.633 | 0.633 | 0.633 | 0.633 | 0.667 | 0.700 | 0.733 | 0.733 | **0.767** | 0.733 | 0.733 | 0.733 | 0.733 | 0.733 | 0.733 |  |  |  |  |
| 27 | 0.567 | 0.567 | 0.567 | 0.567 | 0.600 | 0.600 | 0.600 | 0.600 | 0.600 | 0.633 | 0.633 | 0.633 | 0.633 | 0.633 | 0.633 | 0.700 | 0.733 | 0.733 | 0.733 | 0.733 | 0.733 | 0.733 | 0.733 | 0.733 | 0.733 | 0.733 |  |  |  |
| 28 | 0.567 | 0.567 | 0.567 | 0.567 | 0.600 | 0.600 | 0.600 | 0.600 | 0.600 | 0.633 | 0.633 | 0.633 | 0.633 | 0.633 | 0.633 | 0.700 | 0.733 | 0.733 | 0.733 | 0.733 | 0.733 | 0.733 | 0.733 | 0.733 | 0.733 | 0.733 | 0.700 |  |  |
| 29 | 0.567 | 0.567 | 0.567 | 0.567 | 0.600 | 0.600 | 0.600 | 0.600 | 0.600 | 0.633 | 0.633 | 0.633 | 0.633 | 0.633 | 0.633 | 0.700 | 0.733 | 0.733 | 0.733 | 0.733 | 0.733 | 0.733 | 0.733 | 0.733 | 0.733 | 0.733 | 0.700 | 0.700 |  |
| 30 | 0.567 | 0.567 | 0.567 | 0.567 | 0.600 | 0.600 | 0.600 | 0.600 | 0.600 | 0.633 | 0.633 | 0.633 | 0.633 | 0.633 | 0.633 | 0.700 | 0.733 | 0.733 | 0.733 | 0.733 | 0.733 | 0.733 | 0.733 | 0.733 | 0.733 | 0.700 | 0.700 | 0.700 | 0.633 |

**Table 6.28: R-Range average accuracies for Gutenberg, Concatenated Static. The lower range value is shown across the columns and the upper range value shown across the rows.**



| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 2 | 0.250 | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 3 | 0.200 | 0.175 | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 4 | 0.175 | 0.175 | 0.175 | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 5 | 0.250 | 0.250 | 0.225 | 0.250 | | | | | | | | | | | | | | | | | | | | | | | | | |
| 6 | 0.275 | 0.275 | 0.275 | 0.275 | 0.250 | | | | | | | | | | | | | | | | | | | | | | | | |
| 7 | 0.275 | 0.275 | 0.275 | 0.275 | 0.325 | 0.375 | | | | | | | | | | | | | | | | | | | | | | | |
| 8 | 0.325 | 0.325 | 0.300 | 0.325 | 0.375 | 0.375 | 0.400 | | | | | | | | | | | | | | | | | | | | | | |
| 9 | 0.375 | 0.375 | 0.375 | 0.375 | 0.375 | 0.425 | 0.450 | 0.450 | | | | | | | | | | | | | | | | | | | | | |
| 10 | 0.375 | 0.375 | 0.375 | 0.375 | 0.400 | 0.425 | 0.450 | 0.450 | 0.475 | | | | | | | | | | | | | | | | | | | | |
| 11 | 0.400 | 0.400 | 0.400 | 0.400 | 0.425 | 0.450 | 0.475 | 0.475 | 0.475 | 0.500 | | | | | | | | | | | | | | | | | | | |
| 12 | 0.400 | 0.400 | 0.400 | 0.425 | 0.475 | 0.475 | 0.475 | 0.475 | 0.500 | 0.500 | 0.475 | | | | | | | | | | | | | | | | | | |
| 13 | 0.475 | 0.475 | 0.475 | 0.475 | 0.475 | 0.475 | 0.475 | 0.475 | 0.500 | 0.475 | 0.475 | 0.550 | | | | | | | | | | | | | | | | | |
| 14 | 0.475 | 0.475 | 0.475 | 0.475 | 0.475 | 0.475 | 0.475 | 0.500 | 0.500 | 0.475 | 0.525 | 0.550 | 0.550 | | | | | | | | | | | | | | | | |
| 15 | 0.475 | 0.475 | 0.475 | 0.475 | 0.475 | 0.475 | 0.475 | 0.500 | 0.500 | 0.475 | 0.550 | 0.550 | 0.575 | 0.600 | | | | | | | | | | | | | | | |
| 16 | 0.475 | 0.475 | 0.475 | 0.475 | 0.475 | 0.475 | 0.475 | 0.500 | 0.500 | 0.475 | 0.550 | 0.550 | 0.600 | 0.600 | 0.625 | | | | | | | | | | | | | | |
| 17 | 0.475 | 0.475 | 0.475 | 0.475 | 0.475 | 0.475 | 0.475 | 0.500 | 0.500 | 0.525 | 0.550 | 0.550 | 0.600 | 0.625 | 0.625 | 0.650 | | | | | | | | | | | | | |
| 18 | 0.475 | 0.475 | 0.475 | 0.475 | 0.475 | 0.475 | 0.475 | 0.500 | 0.500 | 0.525 | 0.550 | 0.550 | 0.600 | 0.625 | 0.625 | 0.650 | 0.625 | | | | | | | | | | | | |
| 19 | 0.475 | 0.475 | 0.475 | 0.475 | 0.475 | 0.475 | 0.500 | 0.500 | 0.500 | 0.525 | 0.550 | 0.550 | 0.600 | 0.625 | 0.625 | 0.650 | 0.675 | 0.675 | | | | | | | | | | | |
| 20 | 0.475 | 0.475 | 0.475 | 0.475 | 0.475 | 0.475 | 0.500 | 0.500 | 0.500 | 0.525 | 0.550 | 0.550 | 0.625 | 0.625 | 0.650 | 0.625 | 0.650 | 0.650 | 0.650 | | | | | | | | | | |
| 21 | 0.475 | 0.475 | 0.475 | 0.475 | 0.475 | 0.475 | 0.500 | 0.500 | 0.500 | 0.550 | 0.550 | 0.575 | 0.625 | 0.625 | 0.650 | 0.625 | 0.650 | 0.650 | 0.675 | 0.700 | | | | | | | | | |
| 22 | 0.475 | 0.475 | 0.475 | 0.475 | 0.475 | 0.475 | 0.500 | 0.500 | 0.500 | 0.550 | 0.550 | 0.575 | 0.625 | 0.625 | 0.625 | 0.625 | 0.650 | 0.675 | 0.700 | 0.700 | 0.725 | | | | | | | | |
| 23 | 0.475 | 0.475 | 0.475 | 0.475 | 0.475 | 0.475 | 0.500 | 0.500 | 0.500 | 0.550 | 0.550 | 0.575 | 0.625 | 0.625 | 0.625 | 0.625 | 0.650 | 0.675 | 0.700 | 0.725 | 0.725 | **0.775** | | | | | | | |
| 24 | 0.475 | 0.475 | 0.475 | 0.475 | 0.475 | 0.475 | 0.500 | 0.500 | 0.500 | 0.550 | 0.550 | 0.575 | 0.625 | 0.625 | 0.625 | 0.625 | 0.650 | 0.675 | 0.700 | 0.725 | 0.750 | **0.775** | 0.750 | | | | | | |
| 25 | 0.475 | 0.475 | 0.475 | 0.475 | 0.475 | 0.475 | 0.500 | 0.500 | 0.500 | 0.550 | 0.550 | 0.575 | 0.625 | 0.625 | 0.625 | 0.625 | 0.650 | 0.675 | 0.700 | 0.725 | 0.750 | 0.750 | 0.750 | 0.750 | | | | | |
| 26 | 0.475 | 0.475 | 0.475 | 0.475 | 0.475 | 0.475 | 0.500 | 0.500 | 0.500 | 0.550 | 0.550 | 0.575 | 0.625 | 0.625 | 0.625 | 0.625 | 0.650 | 0.700 | 0.700 | 0.725 | 0.750 | 0.750 | 0.750 | 0.750 | 0.750 | | | | |
| 27 | 0.500 | 0.500 | 0.500 | 0.500 | 0.500 | 0.500 | 0.525 | 0.525 | 0.525 | 0.550 | 0.550 | 0.575 | 0.625 | 0.600 | 0.625 | 0.625 | 0.650 | 0.700 | 0.700 | 0.750 | 0.750 | 0.750 | 0.750 | 0.750 | 0.750 | 0.750 | | | |
| 28 | 0.500 | 0.500 | 0.500 | 0.500 | 0.500 | 0.500 | 0.525 | 0.525 | 0.525 | 0.550 | 0.550 | 0.575 | 0.625 | 0.600 | 0.625 | 0.625 | 0.650 | 0.675 | 0.725 | 0.750 | 0.750 | 0.750 | 0.750 | 0.750 | 0.750 | 0.750 | 0.725 | | |
| 29 | 0.500 | 0.500 | 0.500 | 0.500 | 0.500 | 0.500 | 0.525 | 0.525 | 0.525 | 0.550 | 0.550 | 0.575 | 0.625 | 0.600 | 0.625 | 0.650 | 0.675 | 0.700 | 0.750 | **0.775** | 0.750 | 0.750 | 0.750 | 0.750 | 0.750 | 0.725 | 0.725 | 0.650 | |
| 30 | 0.525 | 0.525 | 0.525 | 0.525 | 0.525 | 0.525 | 0.550 | 0.550 | 0.550 | 0.575 | 0.575 | 0.600 | 0.650 | 0.625 | 0.625 | 0.650 | 0.675 | 0.700 | 0.750 | 0.750 | 0.750 | 0.750 | 0.750 | 0.750 | **0.775** | 0.725 | 0.700 | 0.650 | 0.600 |

**Table 6.29: R-Range average accuracies for Gutenberg, Non-concatenated Static. The lower range value is shown across the columns and the upper range value shown across the rows.**



| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 2 | 0.150 | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 3 | 0.175 | 0.175 | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 4 | 0.200 | 0.200 | 0.200 | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 5 | 0.150 | 0.150 | 0.150 | 0.150 | | | | | | | | | | | | | | | | | | | | | | | | | |
| 6 | 0.200 | 0.200 | 0.200 | 0.200 | 0.250 | | | | | | | | | | | | | | | | | | | | | | | | |
| 7 | 0.250 | 0.250 | 0.250 | 0.250 | 0.250 | 0.300 | | | | | | | | | | | | | | | | | | | | | | | |
| 8 | 0.300 | 0.300 | 0.300 | 0.300 | 0.300 | 0.300 | 0.325 | | | | | | | | | | | | | | | | | | | | | | |
| 9 | 0.325 | 0.325 | 0.325 | 0.325 | 0.325 | 0.325 | 0.325 | 0.325 | | | | | | | | | | | | | | | | | | | | | |
| 10 | 0.325 | 0.325 | 0.325 | 0.325 | 0.325 | 0.325 | 0.325 | 0.350 | 0.375 | | | | | | | | | | | | | | | | | | | | |
| 11 | 0.325 | 0.325 | 0.325 | 0.325 | 0.325 | 0.325 | 0.350 | 0.375 | 0.375 | 0.375 | | | | | | | | | | | | | | | | | | | |
| 12 | 0.325 | 0.325 | 0.325 | 0.325 | 0.325 | 0.350 | 0.375 | 0.375 | 0.375 | 0.375 | 0.450 | | | | | | | | | | | | | | | | | | |
| 13 | 0.350 | 0.350 | 0.350 | 0.350 | 0.350 | 0.350 | 0.375 | 0.375 | 0.375 | 0.400 | 0.450 | 0.500 | | | | | | | | | | | | | | | | | |
| 14 | 0.350 | 0.350 | 0.350 | 0.350 | 0.350 | 0.375 | 0.375 | 0.375 | 0.375 | 0.450 | 0.500 | 0.525 | 0.550 | | | | | | | | | | | | | | | | |
| 15 | 0.350 | 0.350 | 0.350 | 0.375 | 0.375 | 0.375 | 0.375 | 0.375 | 0.400 | 0.450 | 0.500 | 0.525 | 0.550 | 0.550 | | | | | | | | | | | | | | | |
| 16 | 0.375 | 0.375 | 0.375 | 0.375 | 0.375 | 0.375 | 0.375 | 0.375 | 0.400 | 0.450 | 0.500 | 0.525 | 0.550 | 0.575 | 0.625 | | | | | | | | | | | | | | |
| 17 | 0.375 | 0.375 | 0.375 | 0.375 | 0.375 | 0.375 | 0.375 | 0.375 | 0.400 | 0.500 | 0.525 | 0.550 | 0.550 | 0.625 | 0.625 | 0.675 | | | | | | | | | | | | | |
| 18 | 0.375 | 0.375 | 0.375 | 0.375 | 0.375 | 0.375 | 0.375 | 0.375 | 0.400 | 0.500 | 0.525 | 0.550 | 0.550 | 0.625 | 0.625 | 0.675 | 0.675 | | | | | | | | | | | | |
| 19 | 0.375 | 0.375 | 0.375 | 0.375 | 0.375 | 0.375 | 0.375 | 0.400 | 0.425 | 0.500 | 0.525 | 0.550 | 0.550 | 0.625 | 0.675 | 0.675 | 0.675 | 0.725 | | | | | | | | | | | |
| 20 | 0.375 | 0.375 | 0.375 | 0.375 | 0.375 | 0.375 | 0.375 | 0.400 | 0.425 | 0.500 | 0.525 | 0.550 | 0.550 | 0.625 | 0.650 | 0.675 | 0.725 | 0.725 | 0.725 | | | | | | | | | | |
| 21 | 0.375 | 0.375 | 0.375 | 0.375 | 0.375 | 0.375 | 0.375 | 0.400 | 0.450 | 0.500 | 0.525 | 0.550 | 0.550 | 0.625 | 0.650 | 0.675 | 0.725 | 0.725 | 0.725 | 0.725 | | | | | | | | | |
| 22 | 0.375 | 0.375 | 0.375 | 0.375 | 0.375 | 0.375 | 0.375 | 0.400 | 0.450 | 0.500 | 0.525 | 0.550 | 0.575 | 0.600 | 0.650 | 0.675 | 0.725 | 0.725 | 0.725 | **0.775** | **0.775** | | | | | | | | |
| 23 | 0.375 | 0.375 | 0.375 | 0.375 | 0.375 | 0.375 | 0.375 | 0.400 | 0.450 | 0.500 | 0.525 | 0.550 | 0.575 | 0.600 | 0.650 | 0.675 | 0.725 | 0.725 | 0.725 | **0.775** | **0.775** | **0.775** | | | | | | | |
| 24 | 0.375 | 0.375 | 0.375 | 0.375 | 0.375 | 0.375 | 0.375 | 0.400 | 0.450 | 0.500 | 0.525 | 0.550 | 0.575 | 0.600 | 0.650 | 0.675 | 0.725 | 0.725 | **0.775** | **0.775** | **0.775** | **0.775** | **0.775** | | | | | | |
| 25 | 0.375 | 0.375 | 0.375 | 0.375 | 0.375 | 0.375 | 0.375 | 0.400 | 0.450 | 0.500 | 0.525 | 0.550 | 0.575 | 0.600 | 0.650 | 0.675 | 0.725 | 0.725 | **0.775** | **0.775** | **0.775** | **0.775** | **0.775** | **0.775** | | | | | |
| 26 | 0.375 | 0.375 | 0.375 | 0.375 | 0.375 | 0.375 | 0.375 | 0.400 | 0.450 | 0.500 | 0.525 | 0.550 | 0.550 | 0.600 | 0.650 | 0.675 | 0.725 | 0.725 | **0.775** | **0.775** | **0.775** | **0.775** | **0.775** | **0.775** | **0.775** | | | | |
| 27 | 0.375 | 0.375 | 0.375 | 0.375 | 0.375 | 0.375 | 0.375 | 0.400 | 0.450 | 0.500 | 0.525 | 0.550 | 0.550 | 0.600 | 0.650 | 0.675 | 0.725 | 0.725 | **0.775** | **0.775** | **0.775** | **0.775** | **0.775** | **0.775** | **0.775** | **0.775** | | | |
| 28 | 0.375 | 0.375 | 0.375 | 0.375 | 0.375 | 0.375 | 0.375 | 0.400 | 0.450 | 0.500 | 0.525 | 0.550 | 0.550 | 0.600 | 0.650 | 0.675 | 0.725 | 0.725 | **0.775** | **0.775** | **0.775** | **0.775** | **0.775** | **0.775** | **0.775** | **0.775** | **0.775** | | |
| 29 | 0.375 | 0.375 | 0.375 | 0.375 | 0.375 | 0.375 | 0.375 | 0.425 | 0.450 | 0.500 | 0.525 | 0.550 | 0.550 | 0.600 | 0.650 | 0.675 | 0.725 | 0.725 | **0.775** | **0.775** | **0.775** | **0.775** | **0.775** | **0.775** | **0.775** | 0.750 | **0.775** | 0.725 | |
| 30 | 0.375 | 0.375 | 0.375 | 0.375 | 0.375 | 0.375 | 0.375 | 0.425 | 0.450 | 0.500 | 0.525 | 0.550 | 0.550 | 0.650 | 0.650 | 0.675 | 0.725 | 0.725 | **0.775** | **0.775** | **0.775** | **0.775** | **0.775** | **0.775** | **0.775** | **0.775** | 0.725 | 0.725 | 0.675 |

**Table 6.30: R-Range average accuracies for Gutenberg, Non-concatenated Dynamic. The lower range value is shown across the columns and the upper range value shown across the rows.**

| Concatenated Dynamic | Concatenated Static | NonConcatenated Dynamic | NonConcatenated Static |
|---|---|---|---|
| 0.475 | 0.525 | 0.475 | 0.475 |

**Table 6.31: $r^{max}$ average accuracies for Gutenberg.**

The R-Range result within Table 6.27-6.30 all indicate that for Gutenberg higher substring lengths improve the categorization of its documents. This suggests that no matter what protocol is used, when categorizing corpora of this type using R-Ranges, ranges typically over 20 achieve the best results. The finding that longer suffixes are more successful for categorizing this corpus than for others are consistent with the results reported for C-Measure against the same corpus.

Table 6.25 shows that for $R_{\leq q}$-Measure, concatenated models performed better than non-concatenated but unlike with 6.2.3.1, static models outperform. The results suggest that longer substrings improve accuracy or possibly that shorter ones hinder for $R_{\leq q}$-Measure as the accuracies continue to improve as we increase minimum substring length. A value of between 15 and 16 for q is optimal for concatenated models but a much larger value for non-concatenated, with $q = 30$ achieving the highest accuracy.

Table 6.26 shows that for $R_{\geq q}$-Measure it is difficult to clearly state that one model consistently performs better than another but the highest accuracy is achieved by non-concatenated dynamic with a value of 0.775. Unlike 20Newsgroups (Table 6.20) the optimal minimum substring lengths are similar across all four protocols at around 21.

Table 6.31 indicates that for $r^{max}$there is little difference between the accuracies achieved by each protocol, concatenated static did however achieve the highest for $r^{max}$ with an accuracy of 0.525.

#### 6.2.3.3 RCV1-Author

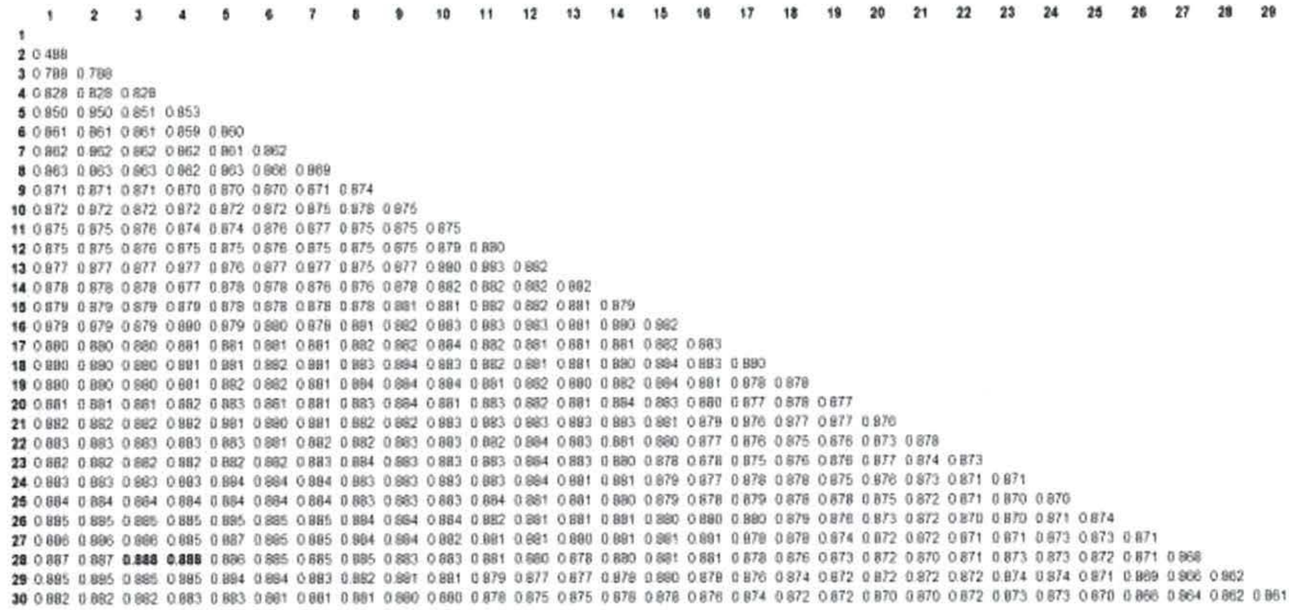
The results for the $R_{\leq q}$-Measure experiments are shown in Table 6.32, $R_{\geq q}$-Measure in Table 6.33, $r^{max}$ in 6.36 and Tables 6.34 and 6.35 show results for R-Ranges. Tables 6.32 and 6.33 display accuracies for both protocols with the leftmost column indicating the lower substring limit for each algorithm. Tables 6.34-6.35 show the accuracy for each range with a single table displaying results for a single protocol. The highest accuracies are again highlighted in bold font.

|  | Concatenated Dynamic | Concatenated Static |
|---|---|---|
| 1 | 0.025 | 0.025 |
| 2 | 0.470 | 0.481 |
| 3 | 0.787 | 0.792 |
| 4 | 0.830 | 0.834 |
| 5 | 0.850 | 0.853 |
| 6 | 0.862 | 0.858 |
| 7 | 0.863 | 0.862 |
| 8 | 0.864 | 0.863 |
| 9 | 0.871 | 0.871 |
| 10 | 0.872 | 0.875 |
| 11 | 0.875 | **0.877** |
| 12 | 0.875 | **0.877** |
| 13 | **0.877** | 0.879 |
| 14 | 0.878 | 0.880 |
| 15 | 0.879 | 0.881 |
| 16 | 0.879 | 0.885 |
| 17 | 0.880 | 0.885 |
| 18 | 0.880 | 0.885 |
| 19 | 0.880 | 0.885 |
| 20 | 0.881 | 0.885 |
| 21 | 0.882 | 0.882 |
| 22 | 0.883 | 0.884 |
| 23 | 0.882 | 0.884 |
| 24 | 0.883 | 0.884 |
| 25 | 0.884 | 0.884 |
| 26 | 0.884 | **0.887** |
| 27 | 0.886 | 0.886 |
| 28 | **0.887** | 0.886 |
| 29 | 0.885 | 0.884 |
| 30 | 0.882 | 0.881 |

**Table 6.32: Accuracies achieved by applying $R_{\leq q}$-Measure to RCV1-Author for each protocol.**

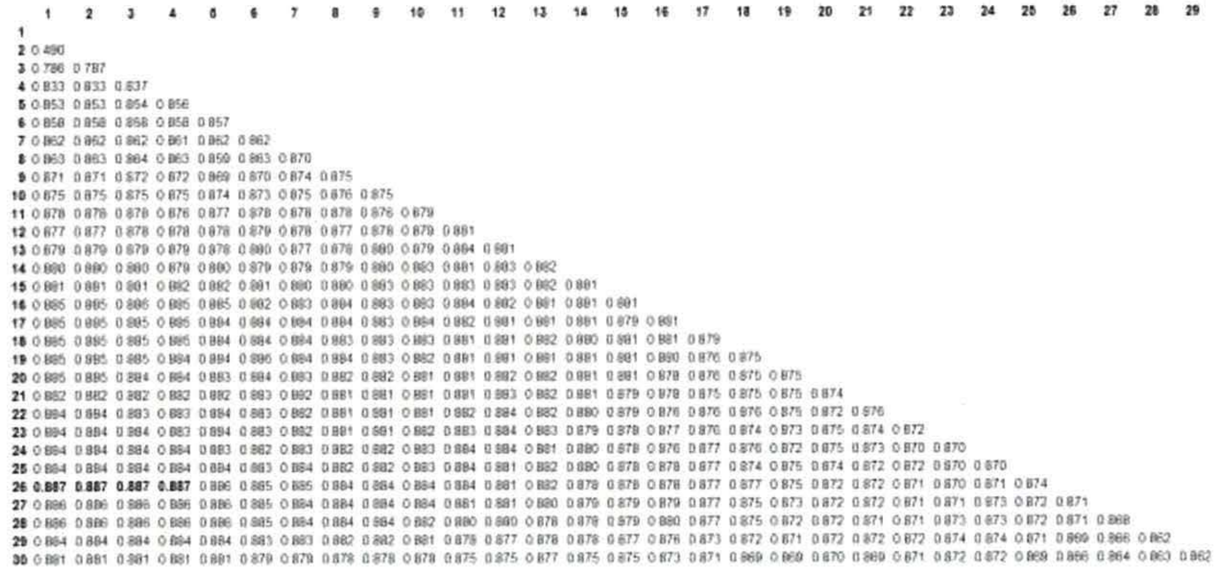
| | Concatenated Dynamic | Concatenated Static |
|---|---|---|
| 1 | 0.879 | 0.879 |
| 2 | 0.879 | 0.879 |
| 3 | 0.879 | 0.879 |
| 4 | 0.880 | 0.879 |
| 5 | 0.880 | 0.879 |
| 6 | 0.879 | 0.879 |
| 7 | 0.880 | 0.879 |
| 8 | 0.880 | 0.881 |
| 9 | 0.879 | 0.882 |
| 10 | 0.881 | **0.883** |
| 11 | 0.880 | 0.881 |
| 12 | 0.881 | 0.882 |
| 13 | 0.882 | **0.883** |
| 14 | **0.884** | **0.883** |
| 15 | 0.881 | 0.878 |
| 16 | 0.877 | 0.878 |
| 17 | 0.875 | 0.875 |
| 18 | 0.875 | 0.872 |
| 19 | 0.872 | 0.872 |
| 20 | 0.869 | 0.870 |
| 21 | 0.867 | 0.867 |
| 22 | 0.868 | 0.869 |
| 23 | 0.865 | 0.865 |
| 24 | 0.862 | 0.862 |
| 25 | 0.859 | 0.859 |
| 26 | 0.856 | 0.856 |
| 27 | 0.853 | 0.853 |
| 28 | 0.851 | 0.851 |
| 29 | 0.847 | 0.847 |
| 30 | 0.845 | 0.845 |

**Table 6.33: Accuracies achieved by applying $R_{\geq q}$-Measure to RCV1-Author for each protocol.**



| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **1** | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| **2** | 0.488 | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| **3** | 0.788 | 0.788 | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| **4** | 0.828 | 0.828 | 0.828 | | | | | | | | | | | | | | | | | | | | | | | | | | |
| **5** | 0.850 | 0.850 | 0.851 | 0.853 | | | | | | | | | | | | | | | | | | | | | | | | | |
| **6** | 0.861 | 0.861 | 0.861 | 0.859 | 0.860 | | | | | | | | | | | | | | | | | | | | | | | | |
| **7** | 0.862 | 0.862 | 0.862 | 0.862 | 0.861 | 0.862 | | | | | | | | | | | | | | | | | | | | | | | |
| **8** | 0.863 | 0.863 | 0.863 | 0.862 | 0.863 | 0.866 | 0.869 | | | | | | | | | | | | | | | | | | | | | | |
| **9** | 0.871 | 0.871 | 0.871 | 0.870 | 0.870 | 0.870 | 0.871 | 0.874 | | | | | | | | | | | | | | | | | | | | | |
| **10** | 0.872 | 0.872 | 0.872 | 0.872 | 0.872 | 0.872 | 0.875 | 0.878 | 0.876 | | | | | | | | | | | | | | | | | | | | |
| **11** | 0.875 | 0.875 | 0.876 | 0.874 | 0.874 | 0.876 | 0.877 | 0.875 | 0.875 | 0.875 | | | | | | | | | | | | | | | | | | | |
| **12** | 0.875 | 0.875 | 0.876 | 0.875 | 0.875 | 0.876 | 0.875 | 0.875 | 0.875 | 0.879 | 0.880 | | | | | | | | | | | | | | | | | | |
| **13** | 0.877 | 0.877 | 0.877 | 0.877 | 0.876 | 0.877 | 0.877 | 0.875 | 0.877 | 0.880 | 0.883 | 0.882 | | | | | | | | | | | | | | | | | |
| **14** | 0.878 | 0.878 | 0.878 | 0.877 | 0.878 | 0.878 | 0.876 | 0.876 | 0.878 | 0.882 | 0.882 | 0.882 | 0.882 | | | | | | | | | | | | | | | | |
| **15** | 0.879 | 0.879 | 0.879 | 0.879 | 0.878 | 0.878 | 0.878 | 0.878 | 0.881 | 0.881 | 0.882 | 0.882 | 0.881 | 0.879 | | | | | | | | | | | | | | | |
| **16** | 0.879 | 0.879 | 0.879 | 0.880 | 0.879 | 0.880 | 0.878 | 0.881 | 0.882 | 0.883 | 0.883 | 0.883 | 0.881 | 0.880 | 0.882 | | | | | | | | | | | | | | |
| **17** | 0.880 | 0.880 | 0.880 | 0.881 | 0.881 | 0.881 | 0.881 | 0.882 | 0.882 | 0.884 | 0.882 | 0.881 | 0.881 | 0.881 | 0.882 | 0.883 | | | | | | | | | | | | | |
| **18** | 0.880 | 0.880 | 0.880 | 0.881 | 0.881 | 0.882 | 0.881 | 0.883 | 0.884 | 0.883 | 0.882 | 0.881 | 0.881 | 0.880 | 0.884 | 0.883 | 0.880 | | | | | | | | | | | | |
| **19** | 0.880 | 0.880 | 0.880 | 0.881 | 0.882 | 0.882 | 0.881 | 0.884 | 0.884 | 0.884 | 0.881 | 0.882 | 0.880 | 0.882 | 0.884 | 0.881 | 0.878 | 0.878 | | | | | | | | | | | |
| **20** | 0.881 | 0.881 | 0.881 | 0.882 | 0.883 | 0.881 | 0.881 | 0.883 | 0.884 | 0.881 | 0.883 | 0.882 | 0.881 | 0.884 | 0.883 | 0.880 | 0.877 | 0.878 | 0.877 | | | | | | | | | | |
| **21** | 0.882 | 0.882 | 0.882 | 0.882 | 0.881 | 0.880 | 0.881 | 0.882 | 0.882 | 0.883 | 0.883 | 0.883 | 0.883 | 0.883 | 0.881 | 0.879 | 0.876 | 0.877 | 0.877 | 0.876 | | | | | | | | | |
| **22** | 0.883 | 0.883 | 0.883 | 0.883 | 0.883 | 0.881 | 0.882 | 0.882 | 0.883 | 0.883 | 0.882 | 0.884 | 0.883 | 0.881 | 0.880 | 0.877 | 0.876 | 0.875 | 0.876 | 0.873 | 0.878 | | | | | | | | |
| **23** | 0.882 | 0.882 | 0.882 | 0.882 | 0.882 | 0.882 | 0.883 | 0.884 | 0.883 | 0.883 | 0.883 | 0.884 | 0.883 | 0.880 | 0.878 | 0.878 | 0.875 | 0.876 | 0.876 | 0.877 | 0.874 | 0.873 | | | | | | | |
| **24** | 0.883 | 0.883 | 0.883 | 0.883 | 0.884 | 0.884 | 0.884 | 0.883 | 0.883 | 0.883 | 0.883 | 0.884 | 0.881 | 0.881 | 0.879 | 0.877 | 0.878 | 0.878 | 0.875 | 0.876 | 0.873 | 0.871 | 0.871 | | | | | | |
| **25** | 0.884 | 0.884 | 0.884 | 0.884 | 0.884 | 0.884 | 0.884 | 0.883 | 0.883 | 0.883 | 0.884 | 0.881 | 0.881 | 0.880 | 0.879 | 0.878 | 0.879 | 0.876 | 0.878 | 0.875 | 0.872 | 0.871 | 0.870 | 0.870 | | | | | |
| **26** | 0.885 | 0.885 | 0.885 | 0.885 | 0.885 | 0.885 | 0.885 | 0.884 | 0.884 | 0.884 | 0.882 | 0.881 | 0.881 | 0.881 | 0.880 | 0.880 | 0.880 | 0.879 | 0.876 | 0.873 | 0.872 | 0.870 | 0.870 | 0.871 | 0.874 | | | | |
| **27** | 0.886 | 0.886 | 0.886 | 0.885 | 0.887 | 0.885 | 0.885 | 0.884 | 0.884 | 0.882 | 0.881 | 0.881 | 0.880 | 0.881 | 0.881 | 0.881 | 0.878 | 0.878 | 0.874 | 0.872 | 0.872 | 0.871 | 0.871 | 0.873 | 0.873 | 0.871 | | | |
| **28** | 0.887 | 0.887 | **0.888** | **0.888** | 0.886 | 0.885 | 0.885 | 0.885 | 0.883 | 0.883 | 0.881 | 0.880 | 0.878 | 0.880 | 0.881 | 0.881 | 0.878 | 0.876 | 0.873 | 0.872 | 0.870 | 0.871 | 0.873 | 0.873 | 0.872 | 0.871 | 0.868 | | |
| **29** | 0.885 | 0.885 | 0.885 | 0.885 | 0.884 | 0.884 | 0.883 | 0.882 | 0.881 | 0.881 | 0.879 | 0.877 | 0.877 | 0.878 | 0.880 | 0.878 | 0.876 | 0.874 | 0.872 | 0.872 | 0.872 | 0.872 | 0.874 | 0.874 | 0.871 | 0.869 | 0.866 | 0.862 | |
| **30** | 0.882 | 0.882 | 0.882 | 0.883 | 0.883 | 0.881 | 0.881 | 0.881 | 0.880 | 0.880 | 0.878 | 0.875 | 0.875 | 0.878 | 0.878 | 0.876 | 0.874 | 0.872 | 0.872 | 0.870 | 0.870 | 0.872 | 0.873 | 0.873 | 0.870 | 0.866 | 0.864 | 0.862 | 0.861 |

**Table 6.34: R-Range average accuracies for RCV1-Author, Concatenated Dynamic. The lower range value is shown across the columns and the upper range value shown across the rows.**



| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **1** | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| **2** | 0.490 | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| **3** | 0.786 | 0.787 | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| **4** | 0.833 | 0.833 | 0.837 | | | | | | | | | | | | | | | | | | | | | | | | | | |
| **5** | 0.853 | 0.853 | 0.854 | 0.856 | | | | | | | | | | | | | | | | | | | | | | | | | |
| **6** | 0.858 | 0.858 | 0.858 | 0.858 | 0.857 | | | | | | | | | | | | | | | | | | | | | | | | |
| **7** | 0.862 | 0.862 | 0.862 | 0.861 | 0.862 | 0.862 | | | | | | | | | | | | | | | | | | | | | | | |
| **8** | 0.863 | 0.863 | 0.864 | 0.863 | 0.859 | 0.863 | 0.870 | | | | | | | | | | | | | | | | | | | | | | |
| **9** | 0.871 | 0.871 | 0.872 | 0.872 | 0.869 | 0.870 | 0.874 | 0.875 | | | | | | | | | | | | | | | | | | | | | |
| **10** | 0.875 | 0.875 | 0.875 | 0.875 | 0.874 | 0.873 | 0.875 | 0.876 | 0.875 | | | | | | | | | | | | | | | | | | | | |
| **11** | 0.878 | 0.878 | 0.878 | 0.876 | 0.877 | 0.878 | 0.878 | 0.878 | 0.876 | 0.879 | | | | | | | | | | | | | | | | | | | |
| **12** | 0.877 | 0.877 | 0.878 | 0.878 | 0.878 | 0.879 | 0.878 | 0.877 | 0.878 | 0.879 | 0.881 | | | | | | | | | | | | | | | | | | |
| **13** | 0.879 | 0.879 | 0.879 | 0.879 | 0.878 | 0.880 | 0.877 | 0.878 | 0.880 | 0.879 | 0.884 | 0.881 | | | | | | | | | | | | | | | | | |
| **14** | 0.880 | 0.880 | 0.880 | 0.879 | 0.880 | 0.879 | 0.879 | 0.879 | 0.880 | 0.883 | 0.881 | 0.883 | 0.882 | | | | | | | | | | | | | | | | |
| **15** | 0.881 | 0.881 | 0.881 | 0.882 | 0.882 | 0.881 | 0.880 | 0.880 | 0.883 | 0.883 | 0.883 | 0.883 | 0.882 | 0.881 | | | | | | | | | | | | | | | |
| **16** | 0.885 | 0.885 | 0.886 | 0.885 | 0.885 | 0.882 | 0.883 | 0.884 | 0.883 | 0.883 | 0.884 | 0.882 | 0.881 | 0.881 | 0.881 | | | | | | | | | | | | | | |
| **17** | 0.885 | 0.885 | 0.885 | 0.885 | 0.884 | 0.884 | 0.884 | 0.884 | 0.883 | 0.884 | 0.882 | 0.881 | 0.881 | 0.881 | 0.879 | 0.881 | | | | | | | | | | | | | |
| **18** | 0.885 | 0.885 | 0.885 | 0.885 | 0.884 | 0.884 | 0.884 | 0.883 | 0.883 | 0.883 | 0.881 | 0.881 | 0.882 | 0.880 | 0.881 | 0.881 | 0.879 | | | | | | | | | | | | |
| **19** | 0.885 | 0.885 | 0.885 | 0.884 | 0.884 | 0.886 | 0.884 | 0.884 | 0.883 | 0.882 | 0.881 | 0.881 | 0.881 | 0.881 | 0.881 | 0.880 | 0.876 | 0.875 | | | | | | | | | | | |
| **20** | 0.885 | 0.885 | 0.884 | 0.884 | 0.883 | 0.884 | 0.883 | 0.882 | 0.882 | 0.881 | 0.881 | 0.882 | 0.882 | 0.881 | 0.881 | 0.878 | 0.876 | 0.875 | 0.875 | | | | | | | | | | |
| **21** | 0.882 | 0.882 | 0.882 | 0.882 | 0.882 | 0.883 | 0.882 | 0.881 | 0.881 | 0.881 | 0.881 | 0.883 | 0.882 | 0.881 | 0.879 | 0.878 | 0.875 | 0.875 | 0.875 | 0.874 | | | | | | | | | |
| **22** | 0.884 | 0.884 | 0.883 | 0.883 | 0.884 | 0.883 | 0.882 | 0.881 | 0.881 | 0.881 | 0.882 | 0.884 | 0.882 | 0.880 | 0.879 | 0.876 | 0.876 | 0.876 | 0.875 | 0.872 | 0.876 | | | | | | | | |
| **23** | 0.884 | 0.884 | 0.884 | 0.883 | 0.884 | 0.883 | 0.882 | 0.881 | 0.881 | 0.882 | 0.883 | 0.884 | 0.883 | 0.879 | 0.878 | 0.877 | 0.876 | 0.874 | 0.873 | 0.875 | 0.874 | 0.872 | | | | | | | |
| **24** | 0.884 | 0.884 | 0.884 | 0.884 | 0.883 | 0.882 | 0.883 | 0.882 | 0.882 | 0.883 | 0.884 | 0.884 | 0.881 | 0.880 | 0.878 | 0.876 | 0.877 | 0.876 | 0.872 | 0.875 | 0.873 | 0.870 | 0.870 | | | | | | |
| **25** | 0.884 | 0.884 | 0.884 | 0.884 | 0.884 | 0.883 | 0.884 | 0.882 | 0.882 | 0.883 | 0.884 | 0.881 | 0.882 | 0.880 | 0.878 | 0.878 | 0.877 | 0.874 | 0.875 | 0.874 | 0.872 | 0.872 | 0.870 | 0.870 | | | | | |
| **26** | **0.887** | **0.887** | **0.887** | **0.887** | 0.886 | 0.885 | 0.885 | 0.884 | 0.884 | 0.884 | 0.884 | 0.881 | 0.882 | 0.878 | 0.878 | 0.878 | 0.877 | 0.877 | 0.875 | 0.872 | 0.872 | 0.871 | 0.870 | 0.871 | 0.874 | | | | |
| **27** | 0.886 | 0.886 | 0.886 | 0.886 | 0.886 | 0.885 | 0.884 | 0.884 | 0.884 | 0.884 | 0.881 | 0.881 | 0.880 | 0.879 | 0.879 | 0.879 | 0.877 | 0.875 | 0.873 | 0.872 | 0.872 | 0.871 | 0.871 | 0.873 | 0.872 | 0.871 | | | |
| **28** | 0.886 | 0.886 | 0.886 | 0.886 | 0.886 | 0.885 | 0.884 | 0.884 | 0.884 | 0.882 | 0.880 | 0.880 | 0.878 | 0.879 | 0.879 | 0.880 | 0.877 | 0.875 | 0.872 | 0.872 | 0.871 | 0.871 | 0.873 | 0.873 | 0.872 | 0.871 | 0.868 | | |
| **29** | 0.884 | 0.884 | 0.884 | 0.884 | 0.884 | 0.883 | 0.883 | 0.882 | 0.882 | 0.881 | 0.878 | 0.877 | 0.878 | 0.878 | 0.877 | 0.876 | 0.873 | 0.872 | 0.871 | 0.872 | 0.872 | 0.872 | 0.874 | 0.874 | 0.871 | 0.869 | 0.866 | 0.862 | |
| **30** | 0.881 | 0.881 | 0.881 | 0.881 | 0.881 | 0.879 | 0.879 | 0.878 | 0.878 | 0.878 | 0.875 | 0.875 | 0.877 | 0.875 | 0.875 | 0.873 | 0.871 | 0.869 | 0.869 | 0.870 | 0.869 | 0.871 | 0.872 | 0.872 | 0.869 | 0.866 | 0.864 | 0.863 | 0.862 |

**Table 6.35: R-Range average accuracies for RCV1-Author, Concatenated Static. The lower range value is shown across the columns and the upper range value shown across the rows.**

| Concatenated Dynamic | Concatenated Static |
|---|---|
| 0.879 | 0.879 |

**Table 6.36: $r^{max}$ average accuracies for RCV1-Author.**

As with 20Newsgroups there does not appear to be a great difference between static or dynamic models. The best performing range is this time achieved between 1-26 and 4-26, an extremely high range as with that of the concatenated 20Newsgroups models. R-Ranges outperformed the C-Measure results as seen in 6.3.1.3 with accuracies of up to 0.888 and 0.887 for the dynamic and static models respectively.

#### 6.2.3.3.4 Reuters-10

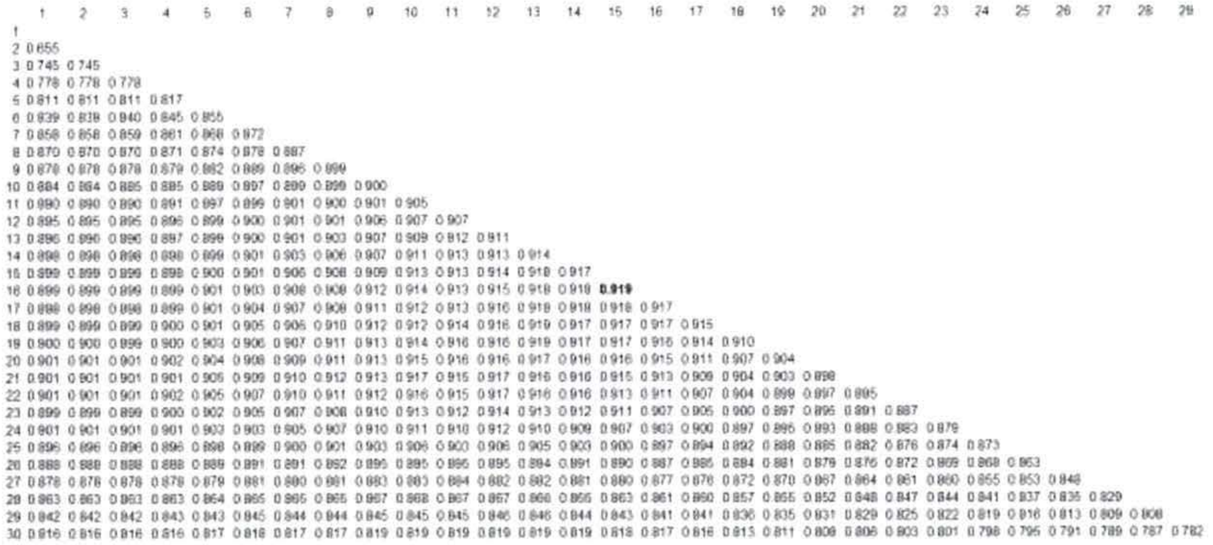
The results for the $R_{\leq q}$-Measure experiments are shown in Table 6.37, $R_{\geq q}$-Measure in Table 6.38, $r^{max}$ in 6.43 and Tables 6.39-6.42 show results for R-Ranges. Tables 6.37 and 6.38 display accuracies for all protocols with the leftmost column indicating the lower substring limit for each algorithm. Tables 6.39-6.42 show the accuracy for each range with a single table displaying results for a single protocol. The highest accuracies are again highlighted in bold font.

| | **Concatenated Dynamic** | **Concatenated Static** | **NonConcatenated Dynamic** | **NonConcatenated Static** |
|---|---|---|---|---|
| 1 | 0.311 | 0.311 | 0.420 | 0.418 |
| 2 | 0.655 | 0.655 | 0.588 | 0.623 |
| 3 | 0.745 | 0.746 | 0.644 | 0.686 |
| 4 | 0.778 | 0.783 | 0.727 | 0.760 |
| 5 | 0.811 | 0.819 | 0.776 | 0.808 |
| 6 | 0.839 | 0.841 | 0.814 | 0.837 |
| 7 | 0.858 | 0.861 | 0.833 | 0.854 |
| 8 | 0.870 | 0.872 | 0.847 | 0.858 |
| 9 | 0.878 | 0.882 | 0.852 | 0.865 |
| 10 | 0.884 | 0.887 | 0.859 | 0.867 |
| 11 | 0.890 | 0.893 | 0.862 | 0.872 |
| 12 | 0.895 | 0.897 | 0.867 | 0.873 |
| 13 | 0.896 | 0.899 | 0.870 | 0.879 |
| 14 | 0.898 | 0.900 | 0.873 | 0.879 |
| 15 | 0.899 | 0.899 | 0.874 | 0.882 |
| 16 | 0.899 | 0.902 | **0.876** | **0.886** |
| 17 | 0.898 | 0.900 | **0.876** | **0.886** |
| 18 | 0.899 | 0.901 | 0.874 | 0.881 |
| 19 | 0.900 | 0.903 | 0.874 | 0.879 |
| 20 | **0.901** | 0.904 | 0.873 | 0.878 |
| 21 | **0.901** | 0.903 | 0.873 | 0.875 |
| 22 | **0.901** | **0.905** | **0.876** | 0.878 |
| 23 | 0.899 | **0.905** | 0.872 | 0.873 |
| 24 | **0.901** | 0.903 | 0.872 | 0.869 |
| 25 | 0.896 | 0.898 | 0.866 | 0.865 |
| 26 | 0.888 | 0.890 | 0.857 | 0.855 |
| 27 | 0.878 | 0.879 | 0.849 | 0.848 |
| 28 | 0.863 | 0.864 | 0.836 | 0.834 |
| 29 | 0.842 | 0.840 | 0.812 | 0.809 |
| 30 | 0.816 | 0.813 | 0.789 | 0.787 |

**Table 6.37: Accuracies achieved by applying $R_{\leq q}$-Measure to Reuters-10 for each protocol.**

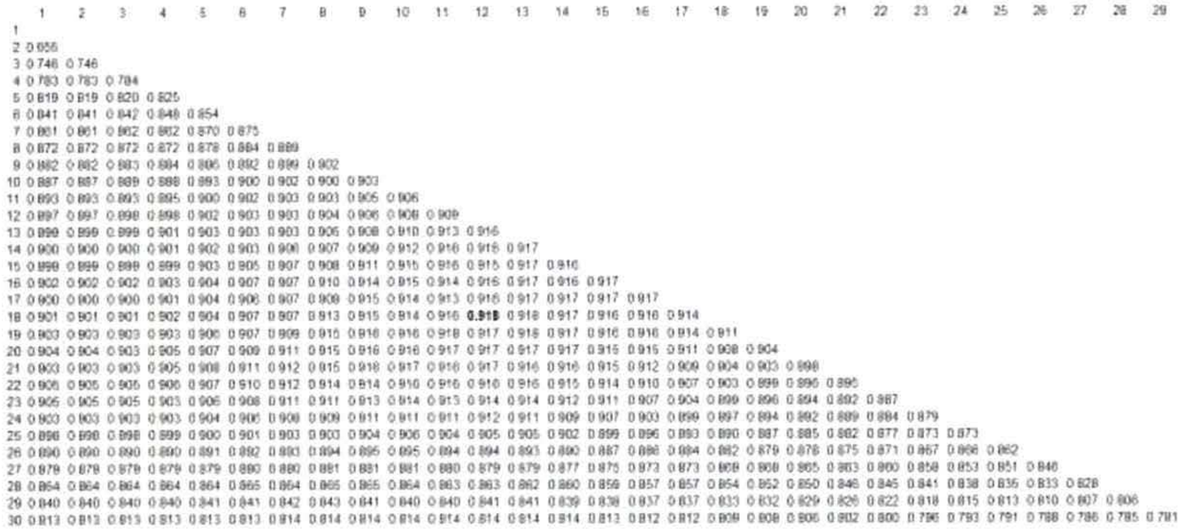
| | **Concatenated Dynamic** | **Concatenated Static** | **NonConcatenated Dynamic** | **NonConcatenated Static** |
|---|---|---|---|---|
| 1 | 0.908 | 0.910 | 0.879 | 0.886 |
| 2 | 0.908 | 0.910 | 0.878 | 0.885 |
| 3 | 0.908 | 0.911 | 0.881 | 0.886 |
| 4 | 0.909 | 0.911 | 0.883 | 0.887 |
| 5 | 0.911 | 0.914 | 0.887 | **0.890** |
| 6 | 0.914 | 0.915 | **0.889** | 0.887 |
| 7 | 0.916 | 0.918 | 0.882 | 0.883 |
| 8 | 0.918 | 0.919 | 0.878 | 0.879 |
| 9 | **0.920** | **0.920** | 0.875 | 0.873 |
| 10 | 0.919 | 0.918 | 0.869 | 0.869 |
| 11 | **0.920** | **0.920** | 0.865 | 0.867 |
| 12 | 0.919 | 0.919 | 0.864 | 0.865 |
| 13 | 0.916 | 0.916 | 0.860 | 0.861 |
| 14 | 0.913 | 0.915 | 0.857 | 0.857 |
| 15 | 0.909 | 0.910 | 0.856 | 0.857 |
| 16 | 0.907 | 0.907 | 0.851 | 0.852 |
| 17 | 0.903 | 0.903 | 0.851 | 0.851 |
| 18 | 0.893 | 0.893 | 0.849 | 0.848 |
| 19 | 0.891 | 0.891 | 0.848 | 0.847 |
| 20 | 0.886 | 0.886 | 0.847 | 0.847 |
| 21 | 0.880 | 0.881 | 0.846 | 0.846 |
| 22 | 0.875 | 0.875 | 0.846 | 0.846 |
| 23 | 0.871 | 0.871 | 0.842 | 0.842 |
| 24 | 0.865 | 0.865 | 0.842 | 0.842 |
| 25 | 0.857 | 0.857 | 0.837 | 0.837 |
| 26 | 0.848 | 0.848 | 0.831 | 0.831 |
| 27 | 0.834 | 0.834 | 0.821 | 0.821 |
| 28 | 0.819 | 0.819 | 0.810 | 0.810 |
| 29 | 0.797 | 0.797 | 0.789 | 0.789 |
| 30 | 0.777 | 0.777 | 0.770 | 0.770 |

**Table 6.38: Accuracies achieved by applying $R_{\geq q}$-Measure to Reuters-10 for each protocol.**



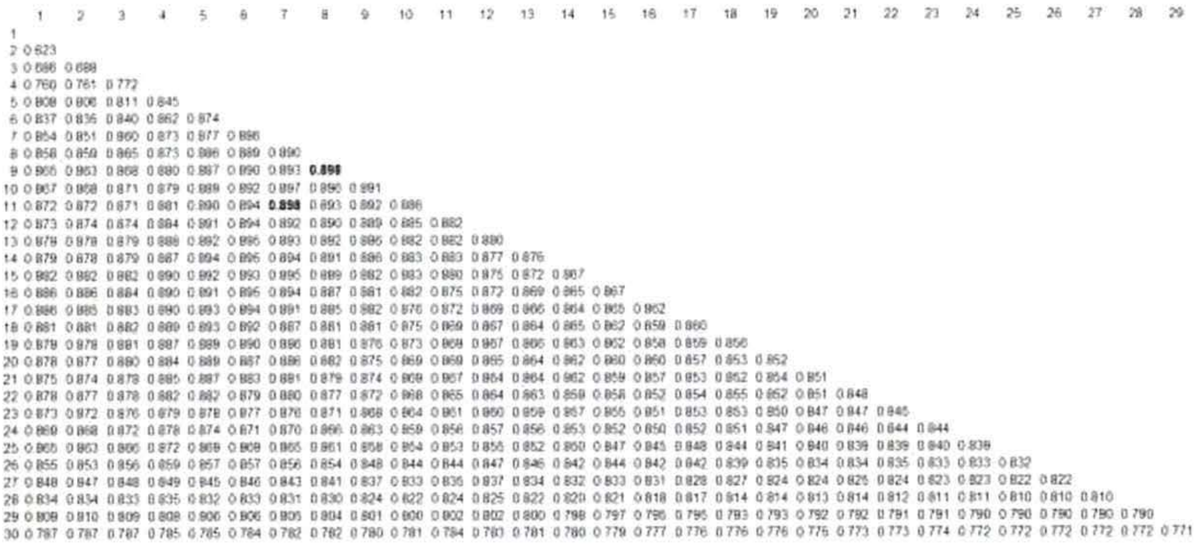
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 2 | 0.655 | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 3 | 0.745 | 0.745 | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 4 | 0.778 | 0.778 | 0.778 | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 5 | 0.811 | 0.811 | 0.811 | 0.817 | | | | | | | | | | | | | | | | | | | | | | | | | |
| 6 | 0.839 | 0.838 | 0.840 | 0.845 | 0.855 | | | | | | | | | | | | | | | | | | | | | | | | |
| 7 | 0.858 | 0.858 | 0.859 | 0.861 | 0.868 | 0.872 | | | | | | | | | | | | | | | | | | | | | | | |
| 8 | 0.870 | 0.870 | 0.870 | 0.871 | 0.874 | 0.878 | 0.887 | | | | | | | | | | | | | | | | | | | | | | |
| 9 | 0.878 | 0.878 | 0.878 | 0.879 | 0.882 | 0.889 | 0.896 | 0.899 | | | | | | | | | | | | | | | | | | | | | |
| 10 | 0.884 | 0.884 | 0.885 | 0.885 | 0.889 | 0.897 | 0.899 | 0.899 | 0.900 | | | | | | | | | | | | | | | | | | | | |
| 11 | 0.890 | 0.890 | 0.890 | 0.891 | 0.897 | 0.899 | 0.901 | 0.900 | 0.901 | 0.905 | | | | | | | | | | | | | | | | | | | |
| 12 | 0.895 | 0.895 | 0.895 | 0.896 | 0.899 | 0.900 | 0.901 | 0.901 | 0.906 | 0.907 | 0.907 | | | | | | | | | | | | | | | | | | |
| 13 | 0.896 | 0.896 | 0.896 | 0.897 | 0.899 | 0.900 | 0.901 | 0.903 | 0.907 | 0.909 | 0.912 | 0.911 | | | | | | | | | | | | | | | | | |
| 14 | 0.898 | 0.898 | 0.898 | 0.898 | 0.899 | 0.901 | 0.903 | 0.906 | 0.907 | 0.911 | 0.913 | 0.913 | 0.914 | | | | | | | | | | | | | | | | |
| 15 | 0.899 | 0.899 | 0.899 | 0.898 | 0.900 | 0.901 | 0.906 | 0.908 | 0.909 | 0.913 | 0.913 | 0.914 | 0.918 | 0.917 | | | | | | | | | | | | | | | |
| 16 | 0.899 | 0.899 | 0.899 | 0.899 | 0.901 | 0.903 | 0.908 | 0.908 | 0.912 | 0.914 | 0.913 | 0.915 | 0.918 | 0.918 | **0.919** | | | | | | | | | | | | | | |
| 17 | 0.898 | 0.898 | 0.898 | 0.899 | 0.901 | 0.904 | 0.907 | 0.908 | 0.911 | 0.912 | 0.913 | 0.916 | 0.918 | 0.918 | 0.918 | 0.917 | | | | | | | | | | | | | |
| 18 | 0.899 | 0.899 | 0.899 | 0.900 | 0.901 | 0.905 | 0.906 | 0.910 | 0.912 | 0.912 | 0.914 | 0.916 | 0.919 | 0.917 | 0.917 | 0.917 | 0.915 | | | | | | | | | | | | |
| 19 | 0.900 | 0.900 | 0.899 | 0.900 | 0.903 | 0.906 | 0.907 | 0.911 | 0.913 | 0.914 | 0.916 | 0.916 | 0.919 | 0.917 | 0.917 | 0.916 | 0.914 | 0.910 | | | | | | | | | | | |
| 20 | 0.901 | 0.901 | 0.901 | 0.902 | 0.904 | 0.908 | 0.909 | 0.911 | 0.913 | 0.915 | 0.916 | 0.916 | 0.917 | 0.916 | 0.916 | 0.915 | 0.911 | 0.907 | 0.904 | | | | | | | | | | |
| 21 | 0.901 | 0.901 | 0.901 | 0.901 | 0.905 | 0.909 | 0.910 | 0.912 | 0.913 | 0.917 | 0.915 | 0.917 | 0.916 | 0.916 | 0.915 | 0.913 | 0.909 | 0.904 | 0.903 | 0.898 | | | | | | | | | |
| 22 | 0.901 | 0.901 | 0.901 | 0.902 | 0.905 | 0.907 | 0.910 | 0.911 | 0.912 | 0.916 | 0.915 | 0.917 | 0.916 | 0.916 | 0.913 | 0.911 | 0.907 | 0.904 | 0.899 | 0.897 | 0.895 | | | | | | | | |
| 23 | 0.899 | 0.899 | 0.899 | 0.900 | 0.902 | 0.905 | 0.907 | 0.908 | 0.910 | 0.913 | 0.912 | 0.914 | 0.913 | 0.912 | 0.911 | 0.907 | 0.905 | 0.900 | 0.897 | 0.895 | 0.891 | 0.887 | | | | | | | |
| 24 | 0.901 | 0.901 | 0.901 | 0.901 | 0.903 | 0.903 | 0.905 | 0.907 | 0.910 | 0.911 | 0.910 | 0.912 | 0.910 | 0.909 | 0.907 | 0.903 | 0.900 | 0.897 | 0.896 | 0.893 | 0.888 | 0.883 | 0.879 | | | | | | |
| 25 | 0.896 | 0.896 | 0.896 | 0.896 | 0.898 | 0.899 | 0.900 | 0.901 | 0.903 | 0.906 | 0.900 | 0.906 | 0.905 | 0.903 | 0.900 | 0.897 | 0.894 | 0.892 | 0.888 | 0.885 | 0.882 | 0.876 | 0.874 | 0.873 | | | | | |
| 26 | 0.888 | 0.888 | 0.888 | 0.888 | 0.889 | 0.891 | 0.891 | 0.892 | 0.895 | 0.895 | 0.896 | 0.895 | 0.894 | 0.891 | 0.890 | 0.887 | 0.885 | 0.884 | 0.881 | 0.879 | 0.876 | 0.872 | 0.869 | 0.868 | 0.863 | | | | |
| 27 | 0.878 | 0.878 | 0.878 | 0.878 | 0.879 | 0.881 | 0.880 | 0.881 | 0.883 | 0.883 | 0.884 | 0.882 | 0.882 | 0.881 | 0.880 | 0.877 | 0.876 | 0.872 | 0.870 | 0.867 | 0.864 | 0.861 | 0.860 | 0.855 | 0.853 | 0.848 | | | |
| 28 | 0.863 | 0.863 | 0.863 | 0.863 | 0.864 | 0.865 | 0.865 | 0.865 | 0.867 | 0.868 | 0.867 | 0.867 | 0.866 | 0.866 | 0.863 | 0.861 | 0.860 | 0.857 | 0.855 | 0.852 | 0.848 | 0.847 | 0.844 | 0.841 | 0.837 | 0.835 | 0.829 | | |
| 29 | 0.842 | 0.842 | 0.842 | 0.843 | 0.843 | 0.845 | 0.844 | 0.844 | 0.845 | 0.845 | 0.845 | 0.846 | 0.846 | 0.844 | 0.843 | 0.841 | 0.841 | 0.836 | 0.835 | 0.831 | 0.829 | 0.825 | 0.822 | 0.819 | 0.816 | 0.813 | 0.809 | 0.808 | |
| 30 | 0.816 | 0.816 | 0.816 | 0.816 | 0.817 | 0.818 | 0.817 | 0.817 | 0.819 | 0.819 | 0.819 | 0.819 | 0.819 | 0.819 | 0.818 | 0.817 | 0.816 | 0.815 | 0.811 | 0.808 | 0.806 | 0.803 | 0.801 | 0.798 | 0.796 | 0.791 | 0.789 | 0.787 | 0.782 |

**Table 6.39: R-Range average accuracies for Reuters-10, Concatenated Dynamic. The lower range value is shown across the columns and the upper range value shown across the rows.**



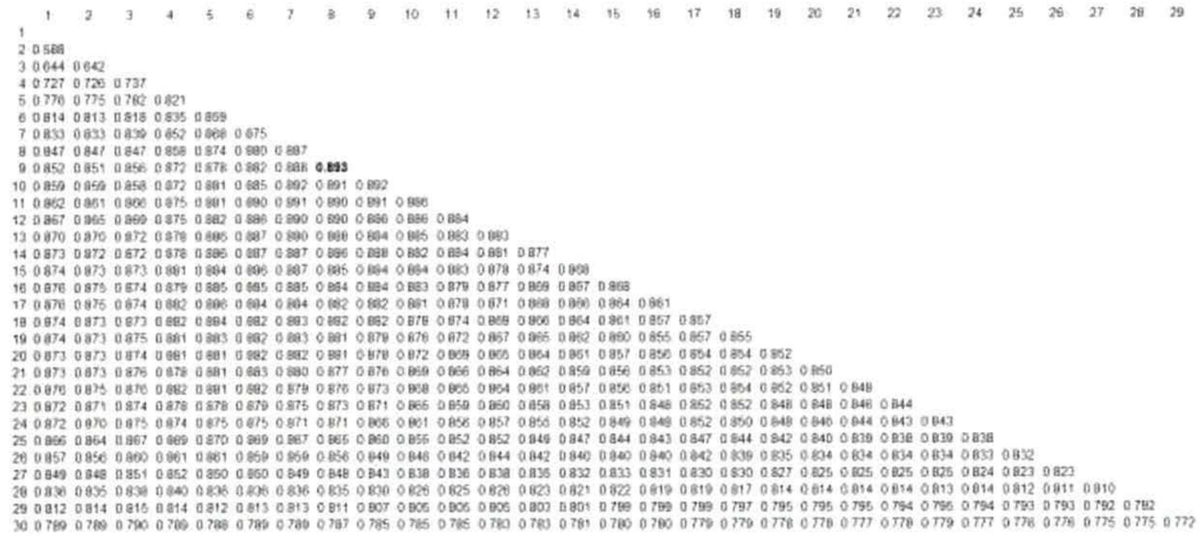
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 2 | 0.656 | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 3 | 0.746 | 0.746 | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 4 | 0.783 | 0.783 | 0.784 | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 5 | 0.819 | 0.819 | 0.820 | 0.825 | | | | | | | | | | | | | | | | | | | | | | | | | |
| 6 | 0.841 | 0.841 | 0.842 | 0.848 | 0.854 | | | | | | | | | | | | | | | | | | | | | | | | |
| 7 | 0.861 | 0.861 | 0.862 | 0.862 | 0.870 | 0.875 | | | | | | | | | | | | | | | | | | | | | | | |
| 8 | 0.872 | 0.872 | 0.872 | 0.872 | 0.878 | 0.884 | 0.889 | | | | | | | | | | | | | | | | | | | | | | |
| 9 | 0.882 | 0.882 | 0.883 | 0.884 | 0.886 | 0.892 | 0.899 | 0.902 | | | | | | | | | | | | | | | | | | | | | |
| 10 | 0.887 | 0.887 | 0.889 | 0.888 | 0.893 | 0.900 | 0.902 | 0.900 | 0.903 | | | | | | | | | | | | | | | | | | | | |
| 11 | 0.893 | 0.893 | 0.893 | 0.895 | 0.900 | 0.902 | 0.903 | 0.903 | 0.905 | 0.906 | | | | | | | | | | | | | | | | | | | |
| 12 | 0.897 | 0.897 | 0.898 | 0.898 | 0.902 | 0.903 | 0.903 | 0.904 | 0.906 | 0.908 | 0.909 | | | | | | | | | | | | | | | | | | |
| 13 | 0.899 | 0.899 | 0.899 | 0.901 | 0.903 | 0.903 | 0.903 | 0.906 | 0.908 | 0.910 | 0.913 | 0.916 | | | | | | | | | | | | | | | | | |
| 14 | 0.900 | 0.900 | 0.900 | 0.901 | 0.902 | 0.903 | 0.906 | 0.907 | 0.909 | 0.912 | 0.916 | 0.916 | 0.917 | | | | | | | | | | | | | | | | |
| 15 | 0.898 | 0.899 | 0.898 | 0.899 | 0.903 | 0.905 | 0.907 | 0.908 | 0.911 | 0.915 | 0.916 | 0.915 | 0.917 | 0.916 | | | | | | | | | | | | | | | |
| 16 | 0.902 | 0.902 | 0.902 | 0.903 | 0.904 | 0.907 | 0.907 | 0.910 | 0.914 | 0.915 | 0.914 | 0.916 | 0.917 | 0.916 | 0.917 | | | | | | | | | | | | | | |
| 17 | 0.900 | 0.900 | 0.900 | 0.901 | 0.904 | 0.906 | 0.907 | 0.908 | 0.915 | 0.914 | 0.913 | 0.916 | 0.917 | 0.917 | 0.917 | 0.917 | | | | | | | | | | | | | |
| 18 | 0.901 | 0.901 | 0.901 | 0.902 | 0.904 | 0.907 | 0.907 | 0.913 | 0.915 | 0.914 | 0.916 | **0.918** | 0.918 | 0.917 | 0.916 | 0.916 | 0.914 | | | | | | | | | | | | |
| 19 | 0.903 | 0.903 | 0.903 | 0.903 | 0.906 | 0.907 | 0.909 | 0.915 | 0.916 | 0.916 | 0.918 | 0.917 | 0.916 | 0.917 | 0.916 | 0.916 | 0.914 | 0.911 | | | | | | | | | | | |
| 20 | 0.904 | 0.904 | 0.903 | 0.905 | 0.907 | 0.909 | 0.911 | 0.915 | 0.916 | 0.916 | 0.917 | 0.917 | 0.917 | 0.917 | 0.915 | 0.915 | 0.911 | 0.908 | 0.904 | | | | | | | | | | |
| 21 | 0.903 | 0.903 | 0.903 | 0.905 | 0.908 | 0.911 | 0.912 | 0.915 | 0.918 | 0.917 | 0.916 | 0.917 | 0.916 | 0.916 | 0.915 | 0.912 | 0.909 | 0.904 | 0.903 | 0.898 | | | | | | | | | |
| 22 | 0.906 | 0.905 | 0.905 | 0.906 | 0.907 | 0.910 | 0.912 | 0.914 | 0.914 | 0.916 | 0.916 | 0.916 | 0.916 | 0.915 | 0.914 | 0.910 | 0.907 | 0.903 | 0.899 | 0.896 | 0.895 | | | | | | | | |
| 23 | 0.906 | 0.905 | 0.905 | 0.903 | 0.906 | 0.908 | 0.911 | 0.911 | 0.913 | 0.914 | 0.913 | 0.914 | 0.914 | 0.912 | 0.911 | 0.907 | 0.904 | 0.899 | 0.896 | 0.894 | 0.892 | 0.887 | | | | | | | |
| 24 | 0.903 | 0.903 | 0.903 | 0.903 | 0.904 | 0.906 | 0.908 | 0.909 | 0.911 | 0.911 | 0.911 | 0.912 | 0.911 | 0.909 | 0.907 | 0.903 | 0.899 | 0.897 | 0.894 | 0.892 | 0.889 | 0.884 | 0.879 | | | | | | |
| 25 | 0.898 | 0.898 | 0.898 | 0.899 | 0.900 | 0.901 | 0.903 | 0.903 | 0.904 | 0.906 | 0.904 | 0.905 | 0.905 | 0.902 | 0.899 | 0.896 | 0.893 | 0.890 | 0.887 | 0.885 | 0.882 | 0.877 | 0.873 | 0.873 | | | | | |
| 26 | 0.890 | 0.890 | 0.890 | 0.890 | 0.891 | 0.892 | 0.893 | 0.894 | 0.895 | 0.895 | 0.894 | 0.894 | 0.893 | 0.890 | 0.887 | 0.886 | 0.884 | 0.882 | 0.879 | 0.878 | 0.875 | 0.871 | 0.867 | 0.866 | 0.862 | | | | |
| 27 | 0.879 | 0.879 | 0.879 | 0.879 | 0.879 | 0.880 | 0.880 | 0.881 | 0.881 | 0.881 | 0.880 | 0.879 | 0.879 | 0.877 | 0.875 | 0.873 | 0.873 | 0.868 | 0.868 | 0.865 | 0.863 | 0.860 | 0.858 | 0.853 | 0.851 | 0.846 | | | |
| 28 | 0.864 | 0.864 | 0.864 | 0.864 | 0.864 | 0.865 | 0.864 | 0.865 | 0.865 | 0.864 | 0.863 | 0.863 | 0.862 | 0.860 | 0.859 | 0.857 | 0.857 | 0.854 | 0.852 | 0.850 | 0.846 | 0.845 | 0.841 | 0.838 | 0.836 | 0.833 | 0.828 | | |
| 29 | 0.840 | 0.840 | 0.840 | 0.840 | 0.841 | 0.841 | 0.842 | 0.843 | 0.841 | 0.840 | 0.840 | 0.841 | 0.841 | 0.839 | 0.838 | 0.837 | 0.837 | 0.833 | 0.832 | 0.829 | 0.826 | 0.822 | 0.818 | 0.815 | 0.813 | 0.810 | 0.807 | 0.806 | |
| 30 | 0.813 | 0.813 | 0.813 | 0.813 | 0.813 | 0.813 | 0.814 | 0.814 | 0.814 | 0.814 | 0.814 | 0.814 | 0.814 | 0.814 | 0.813 | 0.812 | 0.812 | 0.809 | 0.808 | 0.806 | 0.802 | 0.800 | 0.796 | 0.793 | 0.791 | 0.788 | 0.786 | 0.785 | 0.781 |

**Table 6.40: R-Range average accuracies for Reuters-10, Concatenated Static. The lower range value is shown across the columns and the upper range value shown across the rows.**



| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 2 | 0.623 | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 3 | 0.686 | 0.688 | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 4 | 0.760 | 0.761 | 0.772 | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 5 | 0.808 | 0.806 | 0.811 | 0.845 | | | | | | | | | | | | | | | | | | | | | | | | | |
| 6 | 0.837 | 0.836 | 0.840 | 0.862 | 0.874 | | | | | | | | | | | | | | | | | | | | | | | | |
| 7 | 0.854 | 0.851 | 0.860 | 0.873 | 0.877 | 0.886 | | | | | | | | | | | | | | | | | | | | | | | |
| 8 | 0.858 | 0.859 | 0.865 | 0.873 | 0.886 | 0.889 | 0.890 | | | | | | | | | | | | | | | | | | | | | | |
| 9 | 0.866 | 0.863 | 0.868 | 0.880 | 0.887 | 0.890 | 0.893 | **0.898** | | | | | | | | | | | | | | | | | | | | | |
| 10 | 0.867 | 0.868 | 0.871 | 0.879 | 0.889 | 0.892 | 0.897 | 0.896 | 0.891 | | | | | | | | | | | | | | | | | | | | |
| 11 | 0.872 | 0.872 | 0.871 | 0.881 | 0.890 | 0.894 | **0.898** | 0.893 | 0.892 | 0.886 | | | | | | | | | | | | | | | | | | | |
| 12 | 0.873 | 0.874 | 0.874 | 0.884 | 0.891 | 0.894 | 0.892 | 0.890 | 0.889 | 0.885 | 0.882 | | | | | | | | | | | | | | | | | | |
| 13 | 0.878 | 0.878 | 0.879 | 0.888 | 0.892 | 0.896 | 0.893 | 0.892 | 0.886 | 0.882 | 0.882 | 0.880 | | | | | | | | | | | | | | | | | |
| 14 | 0.879 | 0.878 | 0.879 | 0.887 | 0.894 | 0.896 | 0.894 | 0.891 | 0.886 | 0.883 | 0.883 | 0.877 | 0.876 | | | | | | | | | | | | | | | | |
| 15 | 0.882 | 0.882 | 0.882 | 0.890 | 0.892 | 0.893 | 0.895 | 0.889 | 0.882 | 0.883 | 0.880 | 0.875 | 0.872 | 0.867 | | | | | | | | | | | | | | | |
| 16 | 0.886 | 0.886 | 0.884 | 0.890 | 0.891 | 0.896 | 0.894 | 0.887 | 0.881 | 0.882 | 0.875 | 0.872 | 0.869 | 0.865 | 0.867 | | | | | | | | | | | | | | |
| 17 | 0.886 | 0.885 | 0.883 | 0.890 | 0.893 | 0.894 | 0.891 | 0.885 | 0.882 | 0.876 | 0.872 | 0.869 | 0.866 | 0.864 | 0.865 | 0.862 | | | | | | | | | | | | | |
| 18 | 0.881 | 0.881 | 0.882 | 0.889 | 0.893 | 0.892 | 0.887 | 0.881 | 0.881 | 0.875 | 0.869 | 0.867 | 0.864 | 0.865 | 0.862 | 0.859 | 0.860 | | | | | | | | | | | | |
| 19 | 0.879 | 0.878 | 0.881 | 0.887 | 0.889 | 0.890 | 0.886 | 0.881 | 0.876 | 0.873 | 0.869 | 0.867 | 0.866 | 0.863 | 0.862 | 0.858 | 0.859 | 0.856 | | | | | | | | | | | |
| 20 | 0.878 | 0.877 | 0.880 | 0.884 | 0.889 | 0.887 | 0.886 | 0.882 | 0.875 | 0.869 | 0.869 | 0.865 | 0.864 | 0.862 | 0.860 | 0.860 | 0.857 | 0.853 | 0.852 | | | | | | | | | | |
| 21 | 0.875 | 0.874 | 0.878 | 0.885 | 0.887 | 0.883 | 0.881 | 0.879 | 0.874 | 0.869 | 0.867 | 0.864 | 0.864 | 0.862 | 0.859 | 0.857 | 0.853 | 0.852 | 0.854 | 0.851 | | | | | | | | | |
| 22 | 0.878 | 0.877 | 0.878 | 0.882 | 0.882 | 0.879 | 0.880 | 0.877 | 0.872 | 0.868 | 0.865 | 0.864 | 0.863 | 0.859 | 0.858 | 0.852 | 0.854 | 0.855 | 0.852 | 0.851 | 0.848 | | | | | | | | |
| 23 | 0.873 | 0.872 | 0.876 | 0.879 | 0.878 | 0.877 | 0.876 | 0.871 | 0.868 | 0.864 | 0.861 | 0.860 | 0.859 | 0.857 | 0.855 | 0.851 | 0.853 | 0.853 | 0.850 | 0.847 | 0.847 | 0.845 | | | | | | | |
| 24 | 0.869 | 0.868 | 0.872 | 0.878 | 0.874 | 0.871 | 0.870 | 0.866 | 0.863 | 0.859 | 0.856 | 0.857 | 0.856 | 0.853 | 0.852 | 0.850 | 0.852 | 0.851 | 0.847 | 0.846 | 0.846 | 0.844 | 0.844 | | | | | | |
| 25 | 0.865 | 0.863 | 0.866 | 0.872 | 0.869 | 0.869 | 0.865 | 0.861 | 0.858 | 0.854 | 0.853 | 0.855 | 0.852 | 0.850 | 0.847 | 0.845 | 0.848 | 0.844 | 0.841 | 0.840 | 0.839 | 0.839 | 0.840 | 0.838 | | | | | |
| 26 | 0.855 | 0.853 | 0.856 | 0.859 | 0.857 | 0.857 | 0.856 | 0.854 | 0.848 | 0.844 | 0.844 | 0.847 | 0.846 | 0.842 | 0.844 | 0.842 | 0.842 | 0.839 | 0.835 | 0.834 | 0.834 | 0.835 | 0.833 | 0.833 | 0.832 | | | | |
| 27 | 0.848 | 0.847 | 0.848 | 0.849 | 0.845 | 0.846 | 0.843 | 0.841 | 0.837 | 0.833 | 0.835 | 0.837 | 0.834 | 0.832 | 0.833 | 0.831 | 0.828 | 0.827 | 0.824 | 0.824 | 0.825 | 0.824 | 0.823 | 0.823 | 0.822 | 0.822 | | | |
| 28 | 0.834 | 0.834 | 0.833 | 0.835 | 0.832 | 0.833 | 0.831 | 0.830 | 0.824 | 0.822 | 0.824 | 0.825 | 0.822 | 0.820 | 0.821 | 0.818 | 0.817 | 0.814 | 0.814 | 0.813 | 0.814 | 0.812 | 0.811 | 0.811 | 0.810 | 0.810 | 0.810 | | |
| 29 | 0.809 | 0.810 | 0.809 | 0.808 | 0.806 | 0.806 | 0.805 | 0.804 | 0.801 | 0.800 | 0.802 | 0.802 | 0.800 | 0.798 | 0.797 | 0.796 | 0.795 | 0.793 | 0.793 | 0.792 | 0.792 | 0.791 | 0.791 | 0.790 | 0.790 | 0.790 | 0.790 | 0.790 | |
| 30 | 0.787 | 0.787 | 0.787 | 0.785 | 0.785 | 0.784 | 0.782 | 0.782 | 0.780 | 0.781 | 0.784 | 0.780 | 0.781 | 0.780 | 0.779 | 0.777 | 0.776 | 0.776 | 0.776 | 0.775 | 0.773 | 0.773 | 0.774 | 0.772 | 0.772 | 0.772 | 0.772 | 0.772 | 0.771 |

**Table 6.41: R-Range average accuracies for Reuters-10, Non-concatenated Static. The lower range value is shown across the columns and the upper range value shown across the rows.**



| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 2 | 0.588 | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 3 | 0.644 | 0.642 | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 4 | 0.727 | 0.726 | 0.737 | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 5 | 0.776 | 0.775 | 0.782 | 0.821 | | | | | | | | | | | | | | | | | | | | | | | | | |
| 6 | 0.814 | 0.813 | 0.818 | 0.835 | 0.859 | | | | | | | | | | | | | | | | | | | | | | | | |
| 7 | 0.833 | 0.833 | 0.839 | 0.852 | 0.868 | 0.875 | | | | | | | | | | | | | | | | | | | | | | | |
| 8 | 0.847 | 0.847 | 0.847 | 0.858 | 0.874 | 0.880 | 0.887 | | | | | | | | | | | | | | | | | | | | | | |
| 9 | 0.852 | 0.851 | 0.856 | 0.872 | 0.878 | 0.882 | 0.888 | **0.893** | | | | | | | | | | | | | | | | | | | | | |
| 10 | 0.859 | 0.859 | 0.858 | 0.872 | 0.881 | 0.885 | 0.892 | 0.891 | 0.892 | | | | | | | | | | | | | | | | | | | | |
| 11 | 0.862 | 0.861 | 0.866 | 0.875 | 0.881 | 0.890 | 0.891 | 0.890 | 0.891 | 0.886 | | | | | | | | | | | | | | | | | | | |
| 12 | 0.867 | 0.865 | 0.869 | 0.875 | 0.882 | 0.886 | 0.890 | 0.890 | 0.886 | 0.886 | 0.884 | | | | | | | | | | | | | | | | | | |
| 13 | 0.870 | 0.870 | 0.872 | 0.878 | 0.885 | 0.887 | 0.890 | 0.888 | 0.884 | 0.885 | 0.883 | 0.883 | | | | | | | | | | | | | | | | | |
| 14 | 0.873 | 0.872 | 0.872 | 0.878 | 0.886 | 0.887 | 0.887 | 0.886 | 0.888 | 0.882 | 0.884 | 0.881 | 0.877 | | | | | | | | | | | | | | | | |
| 15 | 0.874 | 0.873 | 0.873 | 0.881 | 0.884 | 0.886 | 0.887 | 0.885 | 0.884 | 0.884 | 0.883 | 0.878 | 0.874 | 0.868 | | | | | | | | | | | | | | | |
| 16 | 0.876 | 0.875 | 0.874 | 0.879 | 0.885 | 0.885 | 0.885 | 0.884 | 0.884 | 0.883 | 0.879 | 0.877 | 0.869 | 0.867 | 0.868 | | | | | | | | | | | | | | |
| 17 | 0.876 | 0.876 | 0.874 | 0.882 | 0.886 | 0.884 | 0.884 | 0.882 | 0.882 | 0.881 | 0.878 | 0.871 | 0.868 | 0.866 | 0.864 | 0.861 | | | | | | | | | | | | | |
| 18 | 0.874 | 0.873 | 0.873 | 0.882 | 0.884 | 0.882 | 0.883 | 0.882 | 0.882 | 0.878 | 0.874 | 0.869 | 0.866 | 0.864 | 0.861 | 0.857 | 0.857 | | | | | | | | | | | | |
| 19 | 0.874 | 0.873 | 0.875 | 0.881 | 0.883 | 0.882 | 0.883 | 0.881 | 0.879 | 0.876 | 0.872 | 0.867 | 0.865 | 0.862 | 0.860 | 0.855 | 0.857 | 0.855 | | | | | | | | | | | |
| 20 | 0.873 | 0.873 | 0.874 | 0.881 | 0.881 | 0.882 | 0.882 | 0.881 | 0.878 | 0.872 | 0.869 | 0.865 | 0.864 | 0.861 | 0.857 | 0.856 | 0.854 | 0.854 | 0.852 | | | | | | | | | | |
| 21 | 0.873 | 0.873 | 0.876 | 0.878 | 0.881 | 0.883 | 0.880 | 0.877 | 0.876 | 0.869 | 0.866 | 0.864 | 0.862 | 0.859 | 0.856 | 0.853 | 0.852 | 0.852 | 0.853 | 0.850 | | | | | | | | | |
| 22 | 0.876 | 0.875 | 0.876 | 0.882 | 0.881 | 0.882 | 0.879 | 0.876 | 0.873 | 0.868 | 0.865 | 0.864 | 0.861 | 0.857 | 0.856 | 0.851 | 0.853 | 0.854 | 0.852 | 0.851 | 0.848 | | | | | | | | |
| 23 | 0.872 | 0.871 | 0.874 | 0.878 | 0.878 | 0.879 | 0.875 | 0.873 | 0.871 | 0.865 | 0.859 | 0.860 | 0.858 | 0.853 | 0.851 | 0.848 | 0.852 | 0.852 | 0.848 | 0.848 | 0.846 | 0.844 | | | | | | | |
| 24 | 0.872 | 0.870 | 0.875 | 0.874 | 0.875 | 0.875 | 0.871 | 0.871 | 0.866 | 0.861 | 0.856 | 0.857 | 0.856 | 0.852 | 0.849 | 0.848 | 0.852 | 0.850 | 0.848 | 0.846 | 0.844 | 0.843 | 0.843 | | | | | | |
| 25 | 0.866 | 0.864 | 0.867 | 0.869 | 0.870 | 0.869 | 0.867 | 0.865 | 0.860 | 0.856 | 0.852 | 0.852 | 0.849 | 0.847 | 0.844 | 0.843 | 0.847 | 0.844 | 0.842 | 0.840 | 0.839 | 0.838 | 0.839 | 0.838 | | | | | |
| 26 | 0.857 | 0.856 | 0.860 | 0.861 | 0.861 | 0.859 | 0.859 | 0.856 | 0.849 | 0.846 | 0.842 | 0.844 | 0.842 | 0.840 | 0.840 | 0.840 | 0.842 | 0.839 | 0.835 | 0.834 | 0.834 | 0.834 | 0.834 | 0.833 | 0.832 | | | | |
| 27 | 0.849 | 0.848 | 0.851 | 0.852 | 0.850 | 0.850 | 0.849 | 0.848 | 0.843 | 0.838 | 0.836 | 0.838 | 0.836 | 0.832 | 0.833 | 0.831 | 0.830 | 0.830 | 0.827 | 0.825 | 0.825 | 0.825 | 0.826 | 0.824 | 0.823 | 0.823 | | | |
| 28 | 0.838 | 0.835 | 0.838 | 0.840 | 0.836 | 0.836 | 0.836 | 0.835 | 0.830 | 0.826 | 0.825 | 0.828 | 0.823 | 0.821 | 0.822 | 0.819 | 0.819 | 0.817 | 0.814 | 0.814 | 0.814 | 0.814 | 0.813 | 0.814 | 0.812 | 0.811 | 0.810 | | |
| 29 | 0.812 | 0.814 | 0.816 | 0.814 | 0.812 | 0.813 | 0.813 | 0.811 | 0.807 | 0.805 | 0.806 | 0.805 | 0.803 | 0.801 | 0.799 | 0.799 | 0.799 | 0.797 | 0.795 | 0.795 | 0.795 | 0.794 | 0.796 | 0.794 | 0.793 | 0.793 | 0.792 | 0.792 | |
| 30 | 0.789 | 0.789 | 0.790 | 0.789 | 0.788 | 0.789 | 0.789 | 0.787 | 0.785 | 0.785 | 0.785 | 0.780 | 0.783 | 0.781 | 0.780 | 0.780 | 0.779 | 0.779 | 0.778 | 0.778 | 0.777 | 0.778 | 0.779 | 0.777 | 0.776 | 0.776 | 0.775 | 0.775 | 0.772 |

**Table 6.42: R-Range average accuracies for Reuters-10, Non-concatenated Dynamic. The lower range value is shown across the columns and the upper range value shown across the rows.**

| Concatenated Dynamic | Concatenated Static | NonConcatenated Dynamic | NonConcatenated Static |
|---|---|---|---|
| 0.908 | 0.910 | 0.879 | 0.886 |

**Table 6.43: $r^{max}$ average accuracies for Reuters-10.**

## 6.3 Execution times

The time taken to compute results varies substantially depending on a number of factors including the number of comparisons (i.e. number of testing and training documents), the size of these texts in addition to the algorithms and protocol used. As 20Newsgroups had the greatest number of training and testing files, the computations took a significantly greater amount of time compared to the other corpora and so it is the timing for this corpus which shall be investigated.

### 6.3.1 C-Measure

| NonConc Dyn | NonConc Stat | Conc Dyn | Conc Stat |
|---|---|---|---|
| 398529 | 204591 | 79697 | 402 |

**Table 6.44: Average timings in seconds to calculate C-Measure on 20Newsgroups for each split.**

Table 6.44 shows the average number of seconds taken to compute accuracies for a single cross for each protocol. It is clear that we have great variance for the timings across each of the protocols, the minimum being for concatenated static with timings typically under seven minutes and the longest being for non-concatenated dynamic with timings of over six days. Concatented models are of course much larger than their counterparts and take longer to load into memory but the number of comparisons is drastically reduced to 20 x 3759 = 75,180 in most cases compared to 15036 x 3792 = 57,016,512. It is also worth noting that this means even with our longest case, a training document was compared against a testing stream, with results written to disk on average 0.009 seconds each. Even though each comparison can be executed quickly, unfortunately it still takes a long time due to the overwhelming number to be computed.

The reason static models are much quicker than the dynamic ones is that because the training model is dynamically modified, we are forced to reload the model again when we are to compare against the next testing stream. Speed ups for the concatenated case relating to the order of comparisons was discussed in 5.3.5 and though what is discussed there is true for the concatenated static case, with dynamic we must still reload the original training model. The time taken to do this was drastically reduced by holding the original model in memory and modifying a copy of the object rather than the original.

### 6.3.2 PPM

| | NonConc Dyn | | NonConc Stat | | Conc Dyn | | Conc Stat | |
|---|---|---|---|---|---|---|---|---|
| | With Exclusions | Without Exclusions | With Exclusions | Without Exclusions | With Exclusions | Without Exclusions | With Exclusions | Without Exclusions |
| 2 | 442797 | 330100 | 614554 | 230035 | 2417 | 1915 | 2498 | 1858 |
| 3 | 512327 | 461451 | 790613 | 280094 | 5817 | 5835 | 6234 | 4897 |

**Table 6.45: Average timings in seconds to calculate PPMC on 20Newsgroups for each split.**

| | NonConc Dyn | | NonConc Stat | | Conc Dyn | | Conc Stat | |
|---|---|---|---|---|---|---|---|---|
| | With Exclusions | Without Exclusions | With Exclusions | Without Exclusions | With Exclusions | Without Exclusions | With Exclusions | Without Exclusions |
| 2 | 514562 | 337758 | 983581 | 252895 | 2891 | 3758 | 4073 | 2792 |
| 3 | 595362 | 472157 | 1265359 | 307929 | 8912 | 8195 | 9248 | 8095 |

**Table 6.46: Average timings in seconds to calculate PPMD on 20Newsgroups for each split.**

Results for PPM took much longer to collate than for C-Measure due to the fact that for every protocol, two computations were required for with update exclusions and without. It is clear that again concatenated results were much quicker, as previously mentioned this is because of the very few number of training documents to be compared against each testing document. As expected with update exclusions take longer to process than without and the length of the order dramatically increases execution time.

## 6.4 Chapter Discussion

The best results from each method and protocol for each corpus is listed in Table 6.47 and the best results for each corpus is marked in bold font. The table shows that for Gutenberg and Reuters-10, PPM achieves the highest accuracies and each of these corpora has been shown to be quite different in the number of files and the sizes of each. R-Ranges also achieved extremely high accuracies and outperformed PPM for both 20Newsgroups and RCV1-Author and the results suggest that R-Ranges is the best performing variant of R-Measure.

PPMC has been shown to outperform PPMD overall and also with exclusions achieved higher accuracies on more occasions than without. As mentioned in 6.2.2 the computational overheads involved with investigating high order PPM models for large corpora meant that high orders could not be investigated in all cases, however, interestingly Table 6.47 indicates that in some cases low order models can outperform higher ones. Table 6.19 shows that as with low order models performing well, in some cases substring lengths as low as 1 are important in categorizing texts, especially when concatenated models are being used.

With regards to the performance of each protocol, Table 6.47 shows that the concatenated dynamic protocol achieved the highest accuracies for each of the corpora and concatenated models outperformed non-concatenated in nearly every experiment of all methods for all corpora.

Section 6.3 shows that the amount of time required to gather results for each protocol vary considerably. Dynamic models take slightly longer than static ones due to the need to reload the models or undo changes after each comparison. Non-concatenated models take considerably longer to compute results for due to the large amounts of comparisons required, making it seem very convenient that concatenated models performed best.

| Corpus | Protocol | Method | Accuracy |
|---|---|---|---|
| 20Newsgroups | Conc, Dyn | C-Measure 10 | 0.9070 |
| 20Newsgroups | Conc, Stat | C-Measure 9,10 | 0.9066 |
| 20Newsgroups | NonConc, Dyn | C-Measure 15 | 0.8520 |
| 20Newsgroups | NonConc, Stat | C-Measure 15 | 0.8400 |
| **20Newsgroups** | **Conc, Dyn** | **R-Ranges, 4-15** | **0.9147** |
| 20Newsgroups | Conc, Stat | R-Ranges, 4-15 | 0.9136 |
| 20Newsgroups | NonConc, Dyn | R-Ranges, 11-19 | 0.8523 |
| 20Newsgroups | NonConc, Stat | R-Ranges, 12-13 | 0.8325 |
| 20Newsgroups | Conc, Dyn | PPMD, Order 2, no exclusions | 0.8955 |
| 20Newsgroups | Conc, Stat | PPMD, Order 2, no exclusions | 0.8910 |
| 20Newsgroups | NonConc, Dyn | PPMC, Order 2, with exclusions | 0.7828 |
| 20Newsgroups | NonConc, Stat | PPMD, Order 2, with exclusions | 0.7537 |
| Gutenberg | Conc, Dyn | C-Measure, 24,25 | 0.78 |
| Gutenberg | Conc, Stat | C-Measure, 21 | 0.78 |
| Gutenberg | NonConc, Dyn | C-Measure, 22-27 | 0.78 |
| Gutenberg | NonConc, Stat | C-Measure, 23 | 0.78 |
| Gutenberg | Conc, Dyn | R-Ranges, numerous | 0.775 |
| Gutenberg | Conc, Stat | R-Ranges, numerous | 0.767 |
| Gutenberg | NonConc, Dyn | $R_{\geq q}$ 20-24, R-Ranges 19-24,27-29 | 0.775 |
| Gutenberg | NonConc, Stat | R-Ranges, numerous | 0.775 |
| **Gutenberg** | **Conc, Dyn** | **PPMC 3 no excl, PPMD 3 with/no excl** | **0.95** |
| Gutenberg | Conc, Stat | PPMC 2 with excl, PPMD 2/3/4 with excl | 0.75 |
| Gutenberg | NonConc, Dyn | PPMC 3 with excl, PPMD 5 with excl | 0.925 |
| Gutenberg | NonConc, Stat | PPMC 6/7 with excl, PPMD 6/7 with excl | 0.6 |
| RCV1-Author | Conc Dyn | C-Measure, 16 | 0.8830 |
| RCV1-Author | Conc Stat | C-Measure, 12 | 0.8837 |
| **RCV1-Author** | **Conc Dyn** | **R-Ranges, 3-28, 4-28** | **0.888** |
| RCV1-Author | Conc Stat | $R_{\geq q}$ 11,12,26, R-Ranges numerous | 0.887 |
| RCV1-Author | Conc Dyn | PPMD, 3 no exclusions | 0.8533 |
| RCV1-Author | Conc Stat | PPMD, 3 no exclusions | 0.8518 |
| R-10 | Conc, Dyn | C-Measure, 15 | 0.9177 |
| R-10 | Conc, Stat | C-Measure, 14 | 0.9173 |
| R-10 | NonConc, Dyn | C-Measure, 9 | 0.8927 |
| R-10 | NonConc, Stat | C-Measure, 9 | 0.8954 |
| R-10 | Conc, Dyn | $R_{\geq q}$, 9,11 | 0.920 |
| R-10 | Conc, Stat | $R_{\geq q}$, 9,11 | 0.920 |
| R-10 | NonConc, Dyn | R-Ranges, 8-9 | 0.893 |
| R-10 | NonConc, Stat | R-Ranges, 7-11, 8-9 | 0.898 |
| **R-10** | **Conc, Dyn** | **PPMC, 3, with exclusions** | **0.9531** |
| R-10 | Conc, Stat | PPMC, 3, with exclusions | 0.9455 |
| R-10 | NonConc, Dyn | PPMC, 2, with exclusions | 0.8556 |
| R-10 | NonConc, Stat | PPMC, 4, no exclusions | 0.9080 |

**Table 6.47: Highest achieved accuracies for each method, for each protocol against each corpus.**

| Corpus | Method | Acc. | R | P | BEP | Citation Number |
|---|---|---|---|---|---|---|
| Gutenberg | Markov chains | 0.7472 | | | | Khmelev, D. 2001 |
| Gutenberg | RAR | 0.82 | | | | Marton. et al. 2005 |
| Gutenberg | LZW | 0.83 | | | | Marton. et al. 2005 |
| Gutenberg | GZIP | 0.67 | | | | Marton. et al.2005 |
| R-10 | Word | | | | 0.32 | Yang. 1999 |
| R-10 | kNN | | | | 0.85 | Yang. 1999 |
| R-10 | LLSF | | | | 0.85 | Yang. 1999 |
| R-10 | CLASSI | | | | 0.80 | Yang. 1999 |
| R-10 | RIPPER | | | | 0.80 | Yang. 1999 |
| R-10 | SWAP-1 | | | | 0.79 | Yang. 1999 |
| R-10 | DTree C4.5 | | | | 0.79 | Yang. 1999 |
| R-10 | CHARADE | | | | 0.78 | Yang. 1999 |
| R-10 | EXPERTS (n-gram) | | | | 0.76 | Yang. 1999 |
| R-10 | Rocchio | | | | 0.75 | Yang. 1999 |
| R-10 | NaiveBayes | | | | 0.71 | Yang. 1999 |
| R-10 | Action algorithm | | 0.8691 | 0.8949 | 0.895 | D'Alessio. 1998 |
| R-10 | SVM | | 0.8120 | 0.9137 | | Jimin, L. et al. 2001 |
| R-10 | KNN | | 0.8339 | 0.8807 | | Jimin, L. et al. 2001 |
| R-10 | LSF | | 0.8507 | 0.8489 | | Jimin, L. et al. 2001 |
| R-10 | NNet | | 0.7842 | 0.8785 | | Jimin, L. et al. 2001 |
| R-10 | Naive Bayes | | 0.7688 | 0.8245 | | Jimin, L. et al. 2001 |
| R-10 | Naive Bayes | 0.848 | | | | Teahan, et al. 2001 |
| R-10 | LSVM | 0.919 | | | | Teahan, et al. 2001 |
| R-10 | PPMC Order 2 | 0.863 | | | | Teahan, et al. 2001 |
| R-10 | PPMD Order 3 with excl | 0.910 | | | | Teahan, et al. 2001 |
| R-10 | PPMD Order 3, no excl | 0.902 | | | | Teahan, et al. 2001 |
| R-10 | Findsim | 0.646 | | | | Dumais, S. et al. 1998 |
| R-10 | Naive Bayes | 0.815 | | | | Dumais, S. et al. 1998 |
| R-10 | BayesNets | 0.85 | | | | Dumais, S. et al. 1998 |
| R-10 | Decision Trees | 0.884 | | | | Dumais, S. et al. 1998 |
| R-10 | LinearSVM | 0.92 | | | | Dumais, S. et al. 1998 |
| R-10 | RAR | 0.87 | | | | Marton et al. 2005 |
| R-10 | LZW | 0.84 | | | | Marton et al. 2005 |
| R-10 | GZIP | 0.83 | | | | Marton et al. 2005 |
| R-10 | SVM + IB (information bottleneck) clustering | 0.916 | | | | Bekkerman, R. 2001 |
| 20News | Naive Bayes | 0.85 | | | | McCallum, A. et al. 1998 |
| 20News | PPMD | 0.821 | | | | Teahan, et al. 2001 |
| 20News | Multivariate Bernoulli event model | 0.74 | | | | Teahan, et al. 2001 |
| 20News | Multinomial model | 0.85 | | | | Teahan, et al. 2001 |
| 20News | PrTFIDF | 0.918 | | | | Joachims, T. 1997 |
| 20News | Naiev Bayes | 0.896 | | | | Joachims, T. 1997 |
| 20News | TFIDF | 0.863 | | | | Joachims, T. 1997 |
| 20News | SVM + IB (information bottleneck) clustering | 0.895 | | | | Bekkerman, R. 2001 |
| 20News | RAR | 0.90 | | | | Marton et al. 2005 |
| 20News | LZW | 0.66 | | | | Marton et al. 2005 |
| 20News | GZIP | 0.47 | | | | Marton et al. 2005 |
| RCV1-Author | PPMD Order 8 with excl | 0.8876 | | | | Hunnisett, D. Et al. 2004 |
| RCV1-Author | PPMD Order 7,8 no excl | 0.8978 | | | | Hunnisett, D. Et al. 2004 |
| RCV1-Author | C-Measure Order 13 | 0.9038 | | | | Hunnisett, D. Et al. 2004 |
| RCV1-Author | SVM | 0.85 | | | | Hunnisett, D. Et al. 2004 |
| RCV1-Author | R-Measure | 0.89 | | | | Hunnisett, D. Et al. 2004 |
| RCV1-Author | RAR | 0.894 | | | | Hunnisett, D. Et al. 2004 |
| RCV1-Author | Multi-SVM | 0.85 | | | | Khmelev, D. Et al. 2003 |
| RCV1-Author | Bxip2 | 0.482 | | | | Khmelev, D. Et al. 2003 |

|  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|
| RCV1-Author | Gzip | 0.594 |  |  |  | Khmelev, D. Et al. 2003 |
| RCV1-Author | Markov chains order 1 | 0.661 |  |  |  | Khmelev, D. Et al. 2003 |
| RCV1-Author | Markov chains order 2 | 0.645 |  |  |  | Khmelev, D. Et al. 2003 |
| RCV1-Author | Markov chains order 3 | 0.633 |  |  |  | Khmelev, D. Et al. 2003 |
| RCV1-Author | PPMD Order 2 | 0.813 |  |  |  | Khmelev, D. Et al. 2003 |
| RCV1-Author | PPMD Order 3 | 0.864 |  |  |  | Khmelev, D. Et al. 2003 |
| RCV1-Author | PPMD Order 4 | 0.884 |  |  |  | Khmelev, D. Et al. 2003 |
| RCV1-Author | PPMD Order 5 | 0.892 |  |  |  | Khmelev, D. Et al. 2003 |
| RCV1-Author | RAR | 0.78 |  |  |  | Marton et al. 2005 |
| RCV1-Author | LZW | 0.66 |  |  |  | Marton et al. 2005 |
| RCV1-Author | GZIP | 0.79 |  |  |  | Marton et al. 2005 |

**Table 6.48: Best Results from other text categorization methods.**

Table 6.48 lists results from past experiments for each of the studied corpora. The evaluation techniques of some are different and so columns are provided for accuracy (Acc.), recall (R), precision (P) and also breakeven point (BEP). The citation number pointing to the reference is included in the rightmost column. By comparing the results for stream-based methods against those in Table 6.48 we can see that for 20Newsgroups, R-Ranges outperforms all other methods apart from one. Other methods exist that outperform C-Measure and R-Measure for Gutenberg, however, PPM has clearly outperformed any other method with an accuracy of 0.95.

None of the new results were able to reach the performance achieved of past results for RCV1-Author but the best performing was found to be C-Measure by Hunnisett & Teahan (2004). Table 6.48 shows that there has been substantial experimentation performed against Reuters-21578 and our experimentation of PPM was found to achieve the best results, having accuracy of 0.9531.

For each corpus in turn, Table 6.47 allows us to easily evaluate the performance of each algorithm but it is more difficult to evaluate the performance of each measure in turn against each of the corpora and this is the reasoning for Tables 6.49 – 6.53. It has been shown that the concatenated dynamic protocol performs better than any other overall therefore the following results have been tabulated for this protocol only.

| Order | 20-Newsgroups | Gutenberg | RCV1-Author | Reuters-10 |
|---|---|---|---|---|
| 1 | 0.043 | 0.080 | 0.018 | 0.311 |
| 2 | 0.238 | 0.180 | 0.487 | 0.655 |
| 3 | 0.788 | 0.180 | 0.793 | 0.747 |
| 4 | 0.879 | 0.230 | 0.834 | 0.779 |
| 5 | 0.898 | 0.300 | 0.856 | 0.834 |
| 6 | 0.905 | 0.300 | 0.861 | 0.862 |
| 7 | 0.904 | 0.300 | 0.866 | 0.878 |
| 8 | 0.905 | 0.350 | 0.872 | 0.894 |
| 9 | 0.906 | 0.400 | 0.875 | 0.899 |
| 10 | **0.907** | 0.450 | 0.875 | 0.902 |
| 11 | 0.906 | 0.480 | 0.878 | 0.906 |
| 12 | 0.904 | 0.550 | 0.882 | 0.912 |
| 13 | 0.902 | 0.550 | 0.882 | 0.915 |
| 14 | 0.898 | 0.580 | 0.882 | 0.917 |
| 15 | 0.892 | 0.600 | 0.879 | **0.918** |
| 16 | 0.889 | 0.630 | **0.883** | 0.917 |
| 17 | 0.885 | 0.630 | 0.882 | 0.915 |
| 18 | 0.879 | 0.630 | 0.878 | 0.913 |
| 19 | 0.875 | 0.700 | 0.876 | 0.907 |
| 20 | 0.871 | 0.700 | 0.878 | 0.903 |
| 21 | 0.868 | 0.750 | 0.875 | 0.895 |
| 22 | 0.864 | 0.750 | 0.875 | 0.891 |
| 23 | 0.861 | 0.750 | 0.869 | 0.884 |
| 24 | 0.857 | **0.780** | 0.870 | 0.878 |
| 25 | 0.855 | **0.780** | 0.872 | 0.872 |
| 26 | 0.852 | 0.750 | 0.873 | 0.861 |
| 27 | 0.848 | 0.750 | 0.870 | 0.844 |
| 28 | 0.845 | 0.750 | 0.864 | 0.827 |
| 29 | 0.842 | 0.730 | 0.861 | 0.800 |
| 30 | 0.839 | 0.680 | 0.860 | 0.781 |
| 31 | 0.835 | 0.630 | 0.856 | 0.760 |
| 32 | 0.832 | 0.580 | 0.851 | 0.728 |
| 33 | 0.830 | 0.500 | 0.849 | 0.699 |
| 34 | 0.827 | 0.450 | 0.847 | 0.667 |
| 35 | 0.824 | 0.430 | 0.842 | 0.645 |
| 36 | 0.821 | 0.430 | 0.839 | 0.619 |
| 37 | 0.819 | 0.450 | 0.834 | 0.583 |
| 38 | 0.817 | 0.450 | 0.827 | 0.558 |
| 39 | 0.814 | 0.400 | 0.822 | 0.525 |
| 40 | 0.813 | 0.380 | 0.815 | 0.499 |

**Table 6.49: C-Measure results for each of the corpora for the concatenated dynamic protocol.**

| Order | 20-Newsgroups | Gutenberg | RCV1-Author | Reuters-10 |
|---|---|---|---|---|
| 1 | **0.907** | 0.475 | 0.879 | 0.908 |
| 2 | **0.907** | 0.475 | 0.879 | 0.908 |
| 3 | **0.907** | 0.475 | 0.879 | 0.908 |
| 4 | 0.9066 | 0.475 | 0.88 | 0.909 |
| 5 | 0.9063 | 0.475 | 0.88 | 0.911 |
| 6 | 0.9048 | 0.475 | 0.879 | 0.914 |
| 7 | 0.9023 | 0.475 | 0.88 | 0.916 |
| 8 | 0.8994 | 0.5 | 0.88 | 0.918 |
| 9 | 0.8969 | 0.525 | 0.879 | **0.92** |
| 10 | 0.8932 | 0.525 | 0.881 | 0.919 |
| 11 | 0.8896 | 0.55 | 0.88 | **0.92** |
| 12 | 0.8858 | 0.525 | 0.881 | 0.919 |
| 13 | 0.8807 | 0.525 | 0.882 | 0.916 |
| 14 | 0.8761 | 0.55 | **0.884** | 0.913 |
| 15 | 0.8721 | 0.55 | 0.881 | 0.909 |
| 16 | 0.8684 | 0.6 | 0.877 | 0.907 |
| 17 | 0.8649 | 0.675 | 0.875 | 0.903 |
| 18 | 0.8627 | 0.7 | 0.875 | 0.893 |
| 19 | 0.8595 | 0.725 | 0.872 | 0.891 |
| 20 | 0.8569 | 0.725 | 0.869 | 0.886 |
| 21 | 0.8548 | 0.725 | 0.867 | 0.88 |
| 22 | 0.8521 | 0.725 | 0.868 | 0.875 |
| 23 | 0.8495 | **0.75** | 0.865 | 0.871 |
| 24 | 0.846 | 0.725 | 0.862 | 0.865 |
| 25 | 0.8432 | 0.725 | 0.859 | 0.857 |
| 26 | 0.8407 | 0.725 | 0.856 | 0.848 |
| 27 | 0.8378 | 0.675 | 0.853 | 0.834 |
| 28 | 0.8358 | 0.625 | 0.851 | 0.819 |
| 29 | 0.8331 | 0.575 | 0.847 | 0.797 |
| 30 | 0.8303 | 0.525 | 0.845 | 0.777 |

**Table 6.50: $R_{\geq q}$-Measure results for each of the corpora for the concatenated dynamic protocol.**

| Order | 20-Newsgroups | Gutenberg | RCV1-Author | Reuters-10 |
|---|---|---|---|---|
| 1 | 0.0427 | 0.075 | 0.025 | 0.311 |
| 2 | 0.2379 | 0.175 | 0.47 | 0.655 |
| 3 | 0.7805 | 0.15 | 0.787 | 0.745 |
| 4 | 0.8727 | 0.2 | 0.83 | 0.778 |
| 5 | 0.8939 | 0.3 | 0.85 | 0.811 |
| 6 | 0.9032 | 0.275 | 0.862 | 0.839 |
| 7 | 0.9056 | 0.3 | 0.863 | 0.858 |
| 8 | 0.9084 | 0.3 | 0.864 | 0.87 |
| 9 | 0.9094 | 0.3 | 0.871 | 0.878 |
| 10 | 0.9107 | 0.35 | 0.872 | 0.884 |
| 11 | 0.9113 | 0.35 | 0.875 | 0.89 |
| 12 | 0.913 | 0.4 | 0.875 | 0.895 |
| 13 | 0.9133 | 0.45 | **0.877** | 0.896 |
| 14 | 0.9141 | 0.45 | 0.878 | 0.898 |
| 15 | **0.9145** | 0.45 | 0.879 | 0.899 |
| 16 | 0.914 | 0.45 | 0.879 | 0.899 |
| 17 | 0.9135 | 0.45 | 0.88 | 0.898 |
| 18 | 0.9126 | 0.45 | 0.88 | 0.899 |
| 19 | 0.9119 | 0.45 | 0.88 | 0.9 |
| 20 | 0.9099 | 0.45 | 0.881 | **0.901** |
| 21 | 0.9072 | 0.45 | 0.882 | **0.901** |
| 22 | 0.9047 | **0.475** | 0.883 | **0.901** |
| 23 | 0.9008 | **0.475** | 0.882 | 0.899 |
| 24 | 0.8949 | **0.475** | 0.883 | **0.901** |
| 25 | 0.8904 | **0.475** | 0.884 | 0.896 |
| 26 | 0.8853 | **0.475** | 0.884 | 0.888 |
| 27 | 0.8806 | **0.475** | 0.886 | 0.878 |
| 28 | 0.8766 | **0.475** | **0.887** | 0.863 |
| 29 | 0.8726 | **0.475** | 0.885 | 0.842 |
| 30 | 0.8697 | **0.475** | 0.882 | 0.816 |

**Table 6.51: $R_{\leq q}$-Measure results for each of the corpora for the concatenated dynamic protocol.**

| 20-Newsgroups | Gutenberg | RCV1-Author | Reuters-10 |
|---|---|---|---|
| 0.907 | 0.475 | 0.879 | 0.908 |

**Table 6.52: $r^{max}$ results for each of the corpora for the concatenated dynamic protocol.**

|  | PPMC |  | PPMD |  |
|---|---|---|---|---|
|  | With exclusions | Without exclusions | With exclusions | Without exclusions |
| **20-Newsgroups** | 0.8886 | 0.893 | 0.892 | **0.8955** |
| **Gutenberg** | **0.8** | **0.8** | 0.75 | 0.775 |
| **RCV1-Author** | 0.7994 | 0.8055 | 0.8062 | **0.8146** |
| **Reuters-10** | **0.9477** | 0.9437 | 0.9455 | 0.945 |

**Table 6.53: PPMC and PPMD results for each of the corpora for the concatenated dynamic protocol.**

$r^{max}$ and the new $R_{\leq q}$-Measure is always outperformed by either the $R_{\geq q}$-Measure or the $R_{p..q}$-Measure. Overall, the new $R_{\geq q}$ and $R_{p..q}$ measures compare favourably with the normalised C-Measure and PPM results and with the best results previously published, including feature-based results.

Interestingly, neither PPMC nor PPMD dominates, unlike compression experiments where PPMD usually leads to better compression. The remarkable aspect of the PPM results is that no one order is clearly better across the protocols; and interestingly, models that do not use exclusions in some cases are better than those that do. This is counter-intuitive from an information theoretic perspective where one would expect that the model that performs better at compression (usually PPMD order 5 with exclusions) would also dominate for text categorization. An explanation might be that the categorization process requires optimizing for the best class decision, not best compression. That is, the information concerning the class is not being encoded, so is not being factored into the optimization. Also, performing no exclusions penalizes the classifier by adding an avoidable coding cost, but this only occurs when an escape has occurred to a lower context, strong evidence that the class may be invalid; so the extra coding cost is aiding the classification decision.

Table 6.49 illustrates that the results vary markedly between Gutenberg and the other three corpora. For 20 Newsgroups, RCV1-Author and Reuters-10 $C_k$-Measures using lower values of $k$ perform better, with peaks occurring for $C$ lengths between $10 \leq k \geq 16$. For Gutenberg, in contrast, peaks occur for $24 \leq k \geq 25$ and this may be due to the substantially larger documents found in that collection providing much greater training data for relatively few authors.

It was conjectured in section 3.1 that with natural language text, the shortest substrings would be poor discriminants since these short substrings are common across all strings. This has been borne out in the results, with the lowest $R$ measure ranges not featuring in the best performing methods – for example, for 20 Newsgroups, the best performing method is $R_{4..15}$ (where the substrings less than length 4 are ignored), for Reuters-10 the best performing methods are $R_{\geq 9}$ and $R_{\geq 11}$ (where the substrings less than length 9 or 11 are ignored) and for RCV1-Author the best performing methods are $R_{3..28}$ and $R_{4..28}$. It has been shown that for Gutenberg peaks occurred for much longer substring lengths. It is not yet known if this is due to the style of text (the included Gutenberg texts are novels) or the large amounts of training data for only a few authors. From these results it must be recommended that anyone attempting to categorise new streams exclude shorter substrings and to use a longer minimum shortest substring length for streams similar to the Gutenberg corpus.

## Chapter 7

# Conclusions & future work

### 7.1 Discussion

There were two problems that motivated the work within this thesis. The first was that although stream based methods for text categorization have been shown to perform well in some experiments, no thorough study of their performance has ever been performed on a number of major corpora and their results have not been thoroughly compared against the current state-of-the-art feature based techniques. This is an important problem as the merit of the techniques cannot be fully established until a thorough study has been performed. A number of new stream based methods have been detailed within the thesis and one of these new techniques, R-Ranges, has been shown to outperform all other methods for two of the corpora.

The concept of protocols and how each affects categorization results has also not been studied thoroughly across a number of methods for several corpora. The experimentation performed within chapter 6 showed that the protocol does indeed affect the accuracies of each method and the concatenated dynamic protocol was found to outperform all others on most occasions and performs consistently well across all methods, for all corpora. This study has now conclusively shown that the method used to categorise text must not be the only one, the selection of protocol is also just as important.

From the experimentation, a third problem was identified. It has been highlighted by Yang (1999) that it is often difficult to recreate the exact experimentation conditions of previous studies. One reason for this is that the training and testing splits often differ. To ensure that all methods and protocols were fairly compared, a toolkit was developed to offer a single location from which all methods could be ran, for all protocols, on the same data. This is important as all experiments can now be accurately recreated and any new techniques can then fairly compare its results against all found from within this study.

### 7.2 Summary of chapters

Chapter 2 reviewed important concepts within the field of text categorization and difficulties in comparing results among techniques were mentioned. Several applications of text categorization were discussed as were the most common feature based approaches to categorization. The more common feature based approaches perform pre-processing techniques which consumes both time and resources, but it has been shown that stream based approaches do not. Previous stream based methods were discussed and the fact that their research was sparse was noted.

Chapter 3 introduced new stream based categorization techniques. Improved performance to the C-Measure has meant that longer substring lengths have been examined and several new variants of the R-Measure have been introduced. The chapter also showed how these new variants of R-Measure, C-Measure and PPM could be calculated through the use of the suffix tree data structure which has not been previously performed.

After introducing suffix trees as powerful data structures that allow fast searching, chapter 4 showed that it is possible to compute the stream-based methods in reasonable time and space and detailed the implementation of the stream based techniques.

Chapter 5 details jSCat, a toolkit created to facilitate the text categorization experiments and to allow the calculation of several techniques all at once. As well as making it simple to run experiments for a number of techniques, the toolkit has been shown to be extensible in order to allow the introduction of new techniques and also handles tasks common to categorization. Chapter 5 has also shown that optimisations can be found that drastically affect processing times and we have now been able to analyse stream based substring lengths that are much longer than previous research. One problem with comparing the performance of previous studies in Table 6.48 to that of the results found within this study, shown in Table 6.47 is the inconsistency. The same subsets will not be often used and the evaluation techniques will also differ between experiments. This problem was highlighted by Yang (1999) but the use of the toolkit to perform the experiments for all methods meant that all were performed in a consistent manner, on the same subsets of corpora and evaluated in the same way and this is what is needed for all future research.

Chapter 6 described the experimental results for text categorization using stream-based methods and compared these against a number of feature based techniques. Results obtained for C-Measure, PPMC, PPMD and all R-Measure variants showed that stream-based methods are able to match the performance of state-of-the-art techniques. PPMC has been shown to outperform PPMD overall and also with exclusions achieved higher accuracies on more occasions than without. The computational overheads involved with investigating high order PPM models for large corpora meant that high orders could not be investigated in all cases. However, interestingly Table 6.47 indicates that in some cases low order models can outperform higher ones. Table 6.19 shows that as with low order models performing well, in some cases substring lengths as low as 1 are important in categorizing texts, especially when concatenated models are being used. Concatenated models were found to achieve better accuracies than non-concatenated and concatenated dynamic was the best performing protocol overall. The best performing substring length for C-Measure varies between corpora with lengths of 21-27 achieving high accuracies for Gutenberg; however, lengths of between 9 and 16 achieved the best results for the other three corpora. For R-Ranges it is difficult from the results to say which ranges perform best, however, it is clear that although the best ranges vary greatly for each corpora, for each protocol of each corpus the best ranges are very close.

## 7.3 Contributions

Previous to this study there had been no complete and comparative study on the stream based approaches to text categorization. Within chapter 2 it was shown that in the limited study that had been performed, the methods performed well and so there was a need for these methods to be investigated thoroughly against some well known corpora. Chapter 2 discussed protocols and how their variants have previously been used within the study of text categorization but again the investigation has been limited and provided a further motivation for gathering the results within chapter 6.

New stream based methods have been developed within the study, namely variants of the R-Measure algorithm. The study has shown how these new methods and also existing ones could be implemented using suffix trees, a data structure allowing for very faster searching of substrings between models. It was shown how PPMC, PPMD both with and without update exclusions can be implemented using suffix trees and a C-Measure implementation was developed that allowed us to investigate longer substring lengths than was possible previously.

The results that have been found further support the fact that stream based classifiers can perform as well as current leading techniques, beating them in some cases. In chapter 6, a new method R-Ranges was found to achieve the highest results on a number of occasions, beating well supported methods such as PPM for 20Newsgroups and RCV1-Author. 20Newsgroups is known to be a robust measure used for comparing standard text categorization and RCV1-Author is good for authorship ascription. The fact that this new technique has been found to outperform other state-of-the-art techniques such as PPM justifies the work that has been done.

The results also showed that the choice of protocol does in fact have a major bearing on the successfulness of the results, with concatenated dynamic found to outperform all others on most occasions across all corpora. Interestingly the highest results of all methods for each corpus were all found to have concatenated dynamic as its protocol.

Chapter 6 also showed that there are major differences between the computational times of each protocol with static models taking less time to construct than dynamic ones. Concatenated models were shown to take longer to construct than non-concatenated ones but it was noted that for many comparisons the models could be stored in memory and compared against each testing model yet be loaded only once, this coupled with the fact that generally the number of categories is much smaller than the number of training documents means that using concatenated models for experimentation if often the quickest method.

## 7.4 Review of aims & objectives

The first objective was to further investigate and perform a comparative study of stream based approaches. The results shown in chapter 6 has shown that stream based methods are able to outperform current leading techniques and that they should be considered in any future text categorization study.

The second objective was to discover which stream based approaches perform best in which situations. It was hoped to show that for certain corpora or document lengths that certain approaches and protocols should be used. The results in chapter 6 were able to show that the effectiveness of the techniques varied with each corpus but similarities of corpora usually related to similarity in effectiveness of each technique. One interesting discovery was that the concatenated dynamic protocol was shown to outperform the other protocols on almost all occasions across all the data sets. It would have been desirable to have computed many more results, in particular higher order models for the PPM variants but the computation overheads meant that this was not possible. An attempt was made to compute the results at an early stage but after difficulties in gathering the required subsets of each corpus, much time was lost and ideally the experimentation for each would have been performed much earlier in the study.

The final objective was to show that the suffix tree data structure could be used to implement each of the stream based algorithms. The complexities of processing each of the techniques using suffix trees showed that it is indeed a suitable data structure. However, some of the timings shown in chapter 6 highlight the fact that although an individual comparison may be quick, some corpora require vast amounts of calculations to be performed and this can in some cases take a long time. It was shown in chapter 5 that there are a number of techniques available to further improve the time required to load the models and also to limit the amount of memory required, such techniques are pruning the tree and storing representations of the loaded model within text files so that a node may be added to another quicker after analyzing how one node relates to another.

## 7.5 Future work

As mentioned in 7.2 it was felt that the result for the PPM variants against some of the corpora was limited and although it has been shown that higher order models do not always guarantee improved classification, the results that have been attained perform well and warrant further investigation. One of the biggest problems found during the stages of collecting the results was the amount of time it took to compute all of the results. The current implementation within the toolkit is for all models and results to be stored within text files and loaded as required. In order to greatly increase the flexibility for anyone wishing to take forward this work, it would be suggested to instead use a database to store the models and results so that the required data could be found without having to load entire files.

It was hoped that by running experiments of all variants of each algorithm against each corpora that it would be possible to state that a particular algorithm performs better on certain corpora than others, and to state that this is somehow linked to the sizes and/or type of files contained within. It has been shown that for any dataset, short substring lengths should be omitted when using C-Measure or R-Measure variants. The desired length of the shortest substring to be included does appear to differ between the types of text. For texts similar to those within the Gutenberg dataset, which are novels, this study has found that longer substrings of length 21-27 achieving the highest accuracies. However, lengths of between roughly 4 and 28 achieving the highest accuracies for the others. In addition to the findings of the best performing substring lengths, and that concatenated dynamic models are likely to achieve the highest accuracies it has also been found that for some experimentation some cleanup of the input stream is likely to be required. It is beneficial to retrieve as much training data as possible, however, corpora have been shown to include duplicate files and also disclaimer text, both of which should be searched for and removed. These findings serve as recommendations for any new study. However, there are types of text not covered within this study as it focused on well known corpora i.e. it would be interesting to investigate whether our recommendations achieve the highest accuracies for categorizing tweets (twitter updates), Facebook status updates or blogs.

The process of doing background research on the current implementations of stream based methods brought with it a number of questions to mind. There was always a concern that a training document containing the same word repeatedly would achieve high counts if the word existing within the testing stream. It was this thought that brought about the idea that the counts of each node could be reduced once they have been matched, essentially the opposite process to constructing dynamic models. This reduction could continue until the count reaches zero, at which point the node is removed. This would allow similarities to be matched whilst removing the possibility that a single word could have such a weighting, essentially ensuring that a broader range of nodes are matched for documents to achieve high counts.

Another process that was performed was cleansing of the datasets, for instance Gutenberg had lots of disclaimer text that was not part of the original document and does nothing to improve the categorization of it. During this process of studying the contents of the corpora it was found on a number of occasions that there exists a lot of white space in order to break up sections or separate emails and so on. It is possible this could possibly stop substrings of high lengths being matched (unless the substring itself was whitespace of course). A class was therefore written that removes any extra whitespace from within each document before it is categorized but unfortunately due to the overwhelming number of computations to be performed for the existing experiments, this experimentation was never performed and could well improve the categorization results of each.

# 8 References

Argamon, S., Whitelaw, C., Chase, P., Hota, S. R., Dhawle, S., Garg, N., Levitan, S. *Stylistic Text Classification using Functional Lexical Features*. Journal of the American Society for Information Science and Technology (JASIST), pp. 802-822, 2005.

Baker, L. D. and McCallum, A. (1998). *Distributional clustering of words for text classification.* In Proceedings of the Twenty-first ACM International Conference on Research and Development in Information Retrieval (SIGIR98), pp. 96–103, 1998.

Bekkerman, R. *On feature distributional clustering for text categorization*. 2001.

Biber, D. *Variation across speech and writing*. Cambridge: Cambridge University Press. 1988.

Biber, D. *Dimensions of Register Variation: A Cross-linguistic Comparison*. Cambridge University Press, Cambridge. 1995.

Biebricher, Peter, Fuhr, Norbert, Lustig, Gerhard, Schwantner, Michael and Knorz, Gerhard: *The Automatic Indexing System AIR/PHYS - From Research to Application*. In: Proceedings of the Eleventh Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. pp. 333-342, 1988.

Bieganski, J. R. P. and Carlis, J. V. *"Generalized suffix trees for biological sequence data: Application and implantation"*. In Proceedings of 27th Annual Hawaii International Conference on System Sciences, pp. 35-44, 1994.

Boggess, L., Argawal, R. and Davies, R. *Disambiguation of prepositional phrases in automatically labelled technical text*. AAAI'91, 155-159, 1991.

Boser, B. E., Guyon, I. M. and Vapnik, V. N. *A training algorithm for optimal margin classifiers*. In D. Haussler, editor, 5th Annual ACM Workshop on COLT, pp. 144-152, Pittsburgh, PA, 1992.

Branner, David Prager. *Problems in Comparative Chinese Dialectology: The Classification of Miin and Hakka.* Berlin and New York: Mouton de Gruyter. 2000.

Bratko, A., G. V. Cormack, B. Filipic, T. R. Lynam, and B. Zupan. *Spam filtering using statistical data compression models*. Journal of Machine Learning Research 7 Dec, pp. 2673-2698. 2006.

Bratko, A., Filipic, B. and Zupan, B. *Towards Practical PPM Spam Filtering: Experiments for the TREC 2006 Spam Track*. In Proc. 15th Text REtrieval Conference, TREC 2006, NIST, Gaithersburg, MD, November 2006.

Bratko, A. *Fighting Spam With Data Compression Models*. Virus Bulletin, March 2006, pp S2-S4.

Bratko, A. and Filipic, B. *Spam Filtering Using Compression Models*. Technical Report IJS-DP-9227, Department of Intelligent Systems, Jozef Stefan Institute, November 2005.

Bratko, A. and Filipic, B. *Spam Filtering using Character-level Markov Models: Experiments for the TREC 2005 Spam Track*. In Proc. 14th Text REtrieval Conference, TREC 2005, NIST, Gaithersburg, MD, November 2005.

Cameron, Deborah. *Language: Person, number, gender*. Critical Quarterly, Volume 46, Number 4, December 2004, pp. 131-135. 2004.

Chiang, D., Diab, M., Habash, N., Rambow, O. and Sharif, S. *Parsing Arabic dialects*. In Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics, 2006.

Colley, A. and Todd, Z. *Gender-linked differences in the style and content of e-mails to friends*. Journal of Language and Social Psychology, pp. 380–392. 2002.

Corney, M., De Val, O., Anderson, A. and Mohay, G. *Gender-Preferential Text Mining of E-mail Discourse*. In 18th Annual Computer Security Applications Conference. 2002. San Diego California. 2002.

Crowston, K. and Williams, M. *Reproduced and emergent genres of communication on the World Wide Web*. The Information Society, 16, pp. 201-215. 2000.

David Chang et al, *Parsing Arabic Dialects*

Creecy, H., Masand, M., Smith, J., and Waltz, D. *Trading Mips and Memory for Knowledge Engineering*. Communications of the ACM, 35(8):48-63, 1992.

D'Alessio, S. Hierarchical Text Categorization. 1998.

Dahlgren, K., Lord, C., Wada, H., McDowell, J., and Stabler, E. P. *ITP: Description of the Interpretext System as used for MUC-3*. Proceedings of the 3rd Conference on Message Understanding, San Diego, CA, Association for Computational Linguistics, 163-170, 1991.

Dave, K., Lawrence, S., and Pennock, D. M. *Mining the Peanut Gallery: Opinion Extraction and Semantic Classification of Product Reviews*. In Proceedings of the International World Wide Web Conference, Budapest, Hungary. 2003.

de Vel, O., Corney, M., Anderson, A. and Mohay, G. *Language and Gender Author Cohort Analysis of E-mail for Computer Forensics*. Digital Forensic Research Workshop, August 7 – 9, 2002, Syracuse, NY. 2002.

Drucker, H. D., Wu, D., and V., V. *Support Vector Machines for spam categorization*. IEEE Transactions On Neural Networks 10, 5, 1048-1054, 1999.

Dumais, S., Platt, J., Heckerman, D. and Sahami, M. *Inductive Learning Algorithms And Representations For Text Categorization*. In Proceedings of ACM Conference on Information and Knowledge Management (CIKM98), Nov. 1998, pp. 148-155, 1998.

Ehrenfeucht, A. and Haussler, D. "*A new distance metric on strings computable in linear time*". Discrete Applied Math, 40, 1988.

Finn, A. and Kushmerick, N. *Learning to classify documents according to genre*. JASIST, Special Issue on Computational Analysis of Style, Vol. 57, N. 11, September 2006.

Francis, W. N. and Kucera, H. *Frequency analysis of English usage: lexicon and grammar*. Boston: Houghton Mifflin. 1982.

Fung, G. (2003). *The disputed federalist papers: SVM feature selection via concave minimization*. New York City, ACM Press, 2003.

Gamon, M. and Aue, A. *Automatic identification of sentiment vocabulary: Exploiting low association with known sentiment terms*. In Proceedings of the ACL Workshop on Feature Engineering for Machine Learning in Natural Language Processing, Ann Arbor, Michigan. 2005.

Giegerich, R. and S. Kurtz, S. *"From Ukkonen to McCreight and Weiner: A Unifying View of Linear-Time Suffix Tree Construction"*. Algorithmica 19 (3): pp. 331-353. 1997.

Grishman, R., Sterling, J. and Macleod, C. *New York University description of the PROTEUS system as used for MUC-3*. In Proceedings of the Third Message Understanding Evaluation and Conference, Los Altos, CA: Morgan Kaufmann, May 1991.

Groom, C. J. and Pennebaker, J. W. *The language of love: sex, sexual orientation, and language use in online personal advertisements*. Sex Roles: A Journal of Research, 52 (7-8), pp. 447-461. 2005.

Hardt, S. *On recognizing planned deception*. AAAI-88 Workshop on Plan Recognition, 1988.

Hayes, Philip J. and Weinstein, Steven P. *CONSTRUE/TIS: a system for content-based indexing of a database of news stories*. In Second Annual Conference on Innovative Applications of Artificial Intelligence, 1990.

Hidalgo, J. G. and Lopez, M. M. *Combining text and heuristics for cost-sensitive spam filtering*. In Proceedings of the 4th Computational Natural Language Learning Workshop. Lisbon, Portugal, 99-102, 2000.

Hobbs, Jerry R. SRI *International: Description of the TACITUS system as used for MUC-3*. In Proceedings of the Third Message Understanding Evaluation and Conference , Los Altos, CA: Morgan Kaufmann, May 1991.

Holmes and Forsyth. *The Federalist Revisited: New Directions in Authorship Attribution. Lit Linguist Computing.*1995; 10: 111-127. 1995.

House, A. S., and Neuberg, E. P. (1977). *Toward automatic identification of the language of an utterance.* Preliminary methodological considerations. *Journal of the Acoustical Society of America,* 62(3):708--713, 1977.

Huang, R. and  Hansen, J.H.L. *Dialect Classification on Printed Text using Perplexity Measure and Conditional Random Fields*. Acoustics, Speech and Signal Processing, 2007. ICASSP 2007. IEEE International Conference on, Volume: 4, On page(s): IV-993-IV-996. 2007.

Hunnisett, D. 2010. *Categorizing Human Computer Interaction.* PhD thesis. Bangor University

Hunnisett, D. and Teahan. *Context-based methods for text categorization.* In Proceedings of the SIGIR Conference on Information Retrieval (SIGIR 2004), Sheffield, UK, July 25-29, pp. 578-579. 2004.

Jimin, L. and Chua, T. *Building Semantic Perceptron Net for Topic Spotting.* 2001.

Joachims, T. *Learning to classify Text Using Support Vector Machines: Methods, Theory, and Algorithms.* Kluwer, 2002.

Joachims, T. *A Probabilistic Analysis of the Rocchio Algorithm with TFIDF for Text Categorization.* Proc. of the 14th International Conference on Machine Learning ICML97, pp. 143---151, 1997.

Johansson, S., Atwell, E., Garside, R. and Leech, G. *The tagged LOB Corpus.* Bergen: Norwegian Computing Centre for the Humanities. 1986

Karlgren, J. and Cutting, D. *Recognizing text Genres with Simple Metrics Using Discriminant Analysis.* In Proc. of the 15th International Conference on Computational Linguistics (COLING '94), pp. 1071-1075. 1994.

Kennedy, A. and Inkpen, D. *Sentiment Classification of Movie and Product Reviews Using Contextual Valence Shifters*, Proceedings of Workshop on the Analysis of Informal and Formal Information Exchange during Negotiations, Ottawa, CA, 2005.

Kessler, B., Numberg, G. and Schutze, H. *Automatic Detection of Text Genr*e. Proceedings of the 35th Annual Meeting of the ACL and 8th Conference of the EACL. 1997.

Khmelev, D. *Using Markov Chains for Identification of Writers.* 2001

Khmelev, D. and Teahan. *A repetition based measure for verification of text collections and for text categorization.* In Proceedings of the SIGIR Conference on Information Retrieval (SIGIR 2003), Toronto, pp. 104-110. 2003.

Kim, Y. and Ross, S. *Variations of word frequencies in Genre classification tasks*. In Proceedings DELOS conference on Digital Libraries, Tirrenia, Italy. 2007.

Koppel, M., Argamon, S. and Shimoni, A. R. *Automatically determining the gender of a text's author*. Bar-Ilan University Technical Report BIU-TR-01-32. 2001.

Kwasnik, B. H. and Crowston, K. *Introduction to the special issue: Genres of digital documents*. Information Technology & People. 18(2), pp. 76-88. 2005.

Labov, W. *The intersection of sex and social class in the course of linguistic change*. Language Variation and Change 2. 1990.

Lakoff, R. T. *Language and Women's Place*. Harper Colophon Books, New York. 1975.

Lamel, L. F. and Gauvain, J-L. S. *Language identification using phone-based acoustic likelihoods*, In Proceedings IEEE International Conference on Acoustics, Speech, and Signal Processing 94, pp. 293-296, Adelaide, Australia, April 1994.

Larsson, N. J. *"Structures of string matching and data compression"*. Ph.D. thesis, Dept. of Computer Science, Lund University. 1999.

Layton, R., Watters, P., Dazeley, R. *Authorship attribution for twitter in 140 characters or less*. In Workshop Cybercrime and Trustworthy Computing, pp. 1-8. 2010

Lee, Y. and Myaeng, S. (2004). *Automatic Identification of Text Genres and Their Roles in Subject-Based Categorization*. Proceedings of the 37th Hawaii International Conference on System Sciences.

Lewis, D. *Evaluating Text Categorization*. Proceedings of the Speech and Natural Language Workshop, Asilomar, February pp. 312-318. 1991.

Lewis, D. *Representation and Learning in Information Retrieval*. Phd thesis, Computer Science Department, Univ. of Massachusetts. 1992.

Maron, M. E. (1961). *Automatic indexing: An experimental inquiry*. Journal of the Association for Computing Machinery, 8, 404-417.

Marton, Y., Wu, N. and Hellerstein, L. *On compression-based text classification*. In Proceedings of the 27th European Conference on IR Research (ECIR), pp. 300--314, Santiago de Compostela, Spain, 2005.

McCallum, A. and Nigam, K. *A comparison of event models for naïve bayes text classification*. In Proceedings of AAAI-98 Workshop on Learning for Text Categorization, pp. 41–48. 1998.

McCreight, E. M. *"A Space-Economical Suffix Tree Construction Algorithm"*. Journal of the ACM 23 (2): pp. 262-272. 1976.

Miyoshi, T. and Nakagami, Y. *Sentiment classification of customer reviews on electric products*. Systems, Man and Cybernetics, 2007. ISIC. IEEE International Conference on, pp. 2028-2033. 2007.

Mosteller, F. and Wallace, D. L. (1984). *Applied Bayesian and Classical Inference: The Case of the Federalist Papers*, Springer Series in Statistics.
Mulac, A., Bradac, J. J., and Gibbons, P. *Empirical support for the gender-as-culture hypothesis. An intercultural analysis of male/female language differences*. Human Communication Research, 27, pp. 121-152. 2001.

Mullen, T. and Collier, N. *Sentiment analysis using support vector machines with diverse information sources*. In Dekang Lin and Dekai Wu, editors, Proceedings of EMNLP-2004, pp. 412-418, Barcelona, Spain, July 2004. Association for Computational Linguistics. 2004.

Nagy, N., Zhang, X., Nagy, G. and Schneider, E. *A quantitative categorization of phonemic dialect features in context*. In A. Dey (Ed.), CONTEXT 2005 lecture notes in artificial intelligence 3554, 326–338. Berlin Heidelberg: Springer-Verlag. 2005.

Nerbonne, J., Heeringa, W., and Kleiweg, P. *Comparison and Classification of Dialects*. In Proceedings of the 9th Meeting of the European Chapter of the Association for Computational Linguistics, Bergen, pp. 281-282, 1999.

Nigam, K., McCallum, A., Thrun, S. and Mitchell, T. *Text Classification from Labeled and Unlabeled Documents using EM*. In: Machine Learning 39 (2/3), pp. 103-134. 2000.

Pampapathi, B. M. R. and Levene, M. *"A suffix tree approach to anti-spam email filtering"*. Machine Learning, 65, 2006.

Pang, B., Lee, L., and Vaithyanathan, S. *Thumbs up? Sentiment Classification using Machine Learning Techniques*. Proceedings of EMNLP 2002, pp. 79-86. 2002.

Pantel, P. and Lin, D. *SpamCop: a spam classification and organization program*. In Learning for Text Categorization - Papers from the AAAI Workshop. Madison, Wisconsin, 95-98, 1998.

Pennebaker, J. W., Mehl, M. R. and Niederhoffer, K. G. *Psychological aspects of natural language use: Our words, our selves*. Annual Review of Psychology, 2003. 54: pp. 547-577. 2003.

Rayson, P., Leech, G. and Hodges, M. *Social differentiation in the use of English vocabulary: Some analyses of the conversational component of the British National Corpus*. International Journal of Corpus Linguistics 2 (1), pp. 133-152. 1997.

Read, J. *Using emoticons to reduce dependency in machine learning techniques for sentiment classification*. In Proceedings of ACL-05, 43nd Meeting of the Association for Computational Linguistics, Ann Arbor, US, 2005. Association for Computational Linguistics. 2005.

Rosso, M. *Using Genre to Improve Web Search*, Thesis submitted for the degree of PhD, University of North Carolina at Chapel Hill, USA. 2005.

Sahami, M., Dumais, S., Heckerman, D., and Horvitz, E. *A Bayesian Approach to Filtering Junk EMail*. In Learning for Text Categorization – Papers from the AAAI Workshop, pp. 55–62, Madison Wisconsin. AAAI Technical Report WS-98-05, 1998.

Salton, G., Yang, C. S. and Yu, C. T. *A theory of term importance in automatic text analysis*. Journal of the American Society for Information Science, pp. 33-44, 1975.

Santini, M. *Automatic Genre Identification: Towards a Flexible Classification Scheme*. BCS IRSG Symposium: Future Directions in Information Access 2007 (FDIA 2007). Held in conjunction with the European Summer School on IR (ESSIR 2007). 2007.

Santini, M. *Characterizing Genres of Web Pages: Genre Hybridism and Individualization*. 40th Annual Hawaii International Conference on System Sciences (HICSS'07). 2007.

Schiffman, H. *Bibliography of Gender and Language*. http://ccat.sas.upenn.edu/~haroldfs/popcult/bibliogs/gender/gendbibs.html. 2002.

Sebastiani, F. *Machine Learning in Automated Text Categorization*. ACM Computing Surveys, 34(1):1-47. 2002.

Sebastiani, F. *Text Categorization*. In Alessandro Zanasi, editor, Text Mining and its Applications to Intelligence, CRM and Knowledge Management, pp. 109-129. WIT Press, Southampton, UK, 2005.

Spertus, E. *Smokey: Automatic Recognition of Hostile Messages*. In Proceedings of the Innovative Applications of Artificial Intelligence. 1997.

Sriram, B., Fuhry, D., Demir, E., Ferhatosmanoglu, H. and Demirbas, M. *Short text classification in twitter to improve information filtering*. In SIGIR, pp. 841–842. 2010.

Stamatatos, E., Fakotakis, N. and Kokkinakis, G. *Text Genre Detection Using Common Word Frequencies*. Proceedings of COLING 2000, Saarbrucken, Germany. 2000.

Teahan. *Modelling English text*. D.Phil. thesis, University of Waikato, New Zealand. 1998.

Teahan and Harper, D. J. *Using compression-based language models for text categorization*. Proc. Of the Workshop on Language Modeling and Information Retrieval. 2001.

Thaper, N. *Using Compression For Source Based Classification of Text*. Master's Thesis, Massachusetts Institute of Technology. 2001.

Thomas, D. L. and Teahan. *Text categorization for streams*. Annual ACM Conference on Research and Development in Information Retrieval Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval, Demonstration Session, pp. 907 – 907. 2007.

Toman, M., Tesar, R. and Jezek, K. *Influence of Word Normalization on Text Classification*. In Proceedings of InSciT 2006, pp. 354–358, Merida, Spain, 2006. ISBN 84-611-3105-3.

Trudgill, P. *Sex, covert prestige and linguistic change in the urban British English of Norwich*. Language in Society 1, pp. 179-96, 1969.

Turney, P. D. *Thumbs up or thumbs down? Semantic orientation applied to unsupervised classification reviews*. In Proceedings of the 40$^{th}$ Annual Meeting of the Association for Computational Linguistics (ACL '02), pp. 417-424. 2002.
Ukkonen, E. *"On-line construction of suffix trees"*. Algorithmica 14 (3): 249-260. 1995.

Van Rijsbergen. *Information retrieval (second edition)*, in London: Butterworths. 1979.

Watt, S. *Text categorisation and genre in information retrieval*. In A Goker & J Davies (eds), Information retrieval: Searching in the 21st Century, John Wiley & Sons. 2009.

Weiner, P. *"Linear pattern matching algorithm"*. 14th Annual IEEE Symposium on Switching and Automata Theory: 1-11. 1973.

Yang, Y. *An Evaluation of Statistical Approaches to Text Categorization*. Information Retrieval, 1(1/2), pp. 67-88. 1999.

Yu, B. *An Evaluation of Text Classification Methods for Literary Study*. 2008

Frakes, W. B. *Stemming Algorithms*. In William B. Frakes and Ricardo Baeza-Yates, editors, Information Retrieval Data Structures and Algorithms, pp. 131-160. 1992.

Porter, M. F. *An algorithm for suffix stripping*. Program, Vol. 14, No. 3. pp. 313-316, 1997.

## Web References

Alessandro Moschitti, Information Engineering and Computer Science Department, University of Trento, 01/07/2008. http://dit.unitn.it/~moschitt/corpora.htm

Corpus Linguistics. http://www.essex.ac.uk/linguistics/external/clmt/w3c/corpus_ling/content/corpora/list/index2.html

ERPAePRINTS, The Electronic Resource Preservation and Access Network (ERPANET) and the Digital Curation Centre (DCC), 03/03/2009. http://eprints.erpanet.org/

Europa press release, Data protection: "Junk" e-mail costs internet users €10 billion a year worldwide - Commission study, 02/02/2001. http://europa.eu/rapid/pressReleasesAction.do?reference=IP/01/154

Martin Porter, The Porter Stemming Algorithm, Jan 2006. http://tartarus.org/~martin/PorterStemmer/

Scott Hazen Mueller, spam.abuse.net, 03/03/2009. http://spam.abuse.net/

Spam Laws, California business and professions code, division 7, part 3, chapter 1. Article 1.8. Restrictions On Unsolicited Commercial E-mail Advertisers, 2003. http://www.spamlaws.com/state/ca.shtml

Symantec, Case Study: Symantec Brightmail AntiSpam™ Gives TelstraClear The Advantage In The War Against Spam, 2004. http://www.symantec.com/region/reg_ap/promo/es/docs/TelstraClear_Final.pdf

The Gender Genie. http://bookblog.net/gender/genie.php

The UCI KDD Archive, University of California, Irvine, Feb 1999. http://kdd.ics.uci.edu/databases/reuters21578/reuters21578.html

Wikipedia, Spam (electronic), 03/03/2009. http://en.wikipedia.org/wiki/Spam_(electronic)

Word Splitter, Cognitive Computation Group, University of Illinois at Urbana, 11/03/2009. http://l2r.cs.uiuc.edu/~cogcomp/atool.php?tkey=WS