Learning and Inference of Probabilistic Finite State Machines using MML and Applications to Classification Problem

by

Vidya Saikrishna

Thesis

Submitted by Vidya Saikrishna

for fulfillment of the Requirements for the Degree of

Doctor of Philosophy (0190)

Supervisor: Sid Ray

Associate Supervisor: David Dowe

Faculty of Information Technology
Monash University

June, 2017

I luv youse all

Contents

List of Figures . . . . . **ix**

List of Tables . . . . . **xiii**

Abstract . . . . . **xv**

Acknowledgments . . . . . **xix**

1 Introduction **1**

1.1 Text classification using MML 5

1.1.1 Proposed Method-1 6

1.1.2 Proposed Method-2 7

1.1.3 Proposed Method-3 8

1.2 Contributions 10

1.3 Thesis Outline 11

2 Finite State Machines (FSMs) and Probabilistic FSMs **15**

2.1 Introduction 15

2.2 Finite State Machines 16

2.2.1 Mathematical Definition of a Finite State Machine 17

2.2.2 Representation of an FSM in a state diagram 18

2.3 Probabilistic Finite State Machines 21

2.3.1 Mathematical Definition of a Probabilistic Finite State Machine 22

2.3.2 Representation of a PFSM in a state diagram 22

2.4 Equivalence of FSMs and PFSMs 25

2.5 Summary 26

3 Minimum Message Length (MML) 29

3.1 Introduction 29

3.2 Bayes' Theorem 29

3.2.1 An example 31

3.2.2 Bayesian Interpretation 31

3.3 Bayesian Inference 32

3.4 Coding Probabilities 34

3.4.1 Prefix Code Tree 35

3.5 Minimum Message Length (MML) principle 38

3.5.1 More on MML and Applications 40

3.6 Code length calculation for a Binomial distribution case 42

3.7 Code length calculation for a Multinomial distribution case 44

3.8 Summary 45

4 MML Encoding and Inference of PFSMs 47

4.1 Introduction 47

4.2 Modelling an Inductive Hypothesis 48

4.3 Modelling a PFSM 49

4.3.1 Assertion code for hypothesis H 50

4.3.2 Assertion code for data D generated by hypothesis H 54

4.3.2.1 Adaptive Coding of data D with a uniform prior . . 55

4.3.3 A Few Examples 57

4.4 Inference of PFSM using MML 61

4.4.1 Inference of PFSM by Ordered Merging (OM) 62

4.4.1.1 First Stage Merge 62

4.4.1.2 Second Stage Merge 63

4.4.1.3 Third Stage Merge 64

4.4.1.4 Ordered Merging (OM) algorithm 64

4.4.2 Inference of PFSM using Simulated Annealing (SA) 66
4.4.2.1 Simulated Annealing (SA) 67
4.4.2.2 Simulated Annealing (SA) algorithm 68
4.5 Experiments and Discussion 68
4.6 Summary 73

5 Learning Two-Machine PFSM models 77
5.1 Introduction 77
5.2 Design of the two-machine PFSM models 78
5.3 Classification using the two-machine PFSM models 80
5.4 Experiments 82
5.4.1 Experimental set-up A 83
5.4.1.1 Model Training and Feature Selection 84
5.4.1.2 Evaluation Procedure 87
5.4.1.3 Testing and Results 87
5.4.1.4 Minimum Description Length (MDL) classifier 97
5.4.1.5 Comparison of Two-Machine Model approach using MML with the MDL approach 98
5.4.2 Experimental set-up B 100
5.4.2.1 Description of Datasets 100
5.4.2.2 Conversion of the ADL datasets for PFSM modelling 102
5.4.2.3 Building PFSM models 103
5.4.2.4 Prediction of Individuals 104
5.5 Summary 106

6 Learning Single-Machine PFSM model 109
6.1 Introduction 109
6.2 Design of the single-machine PFSM model 110
6.3 Classification using the single-machine PFSM model 113

6.4 Differences between the Single-Machine PFSM model and the Two-Machine PFSM models . . . 116
6.5 Experiments . . . 117
6.5.1 Naive Bayes Classification methods . . . 121
6.5.1.1 Basic Naive Bayes . . . 122
6.5.1.2 Multinomial Term Frequency Naive Bayes . . . 123
6.5.1.3 Multinomial Boolean Naive Bayes . . . 123
6.5.1.4 Multivariate Bernoulli Naive Bayes . . . 123
6.5.1.5 Boolean Naive Bayes . . . 124
6.5.1.6 Multivariate Gauss Naive Bayes . . . 124
6.5.1.7 Flexible Bayes . . . 125
6.5.1.8 Support Vector Machines . . . 125
6.5.2 Comparison of the Single-Machine model with the Naive Bayes Classifiers . . . 126
6.6 Summary . . . 127

7 Learning Hierarchical PFSM (HPFSM) model . . . 131
7.1 Introduction . . . 131
7.2 Hierarchical Probabilistic Finite State Machine (HPFSM) . . . 132
7.2.1 Defining an HPFSM . . . 134
7.2.2 MML assertion code for the hypothesis H of HPFSM . . . 135
7.2.3 Encoding the transitions of HPFSM . . . 138
7.3 Experiments . . . 139
7.3.1 Experiments on Artificial datasets . . . 139
7.3.1.1 Example-1 . . . 139
7.3.1.2 Example-2 . . . 143
7.3.2 Experiments on ADL datasets . . . 146
7.4 Summary . . . 150

8 Conclusions and Future Work . . . 153

8.1 Summary and Conclusions 153
8.2 Future Work 157

Publications **160**

Bibliography **162**

List of Figures

1.1 Probabilistic Finite State Machines generating L 3

2.1 Behaviour of a stopwatch represented as an FSM 16

2.2 Finite State Machine accepting the above mentioned language L . . . 19

2.3 Hypothetical Finite State Machine generating L 21

2.4 Probabilistic Finite State Machine generating L 23

2.5 Finite State Machine M1 25

2.6 Finite State Machine M2 25

3.1 Venn Diagram illustrating Bayes's theorem 30

3.2 Tree for prefix code 35

3.3 Tree for prefix code with weights and levels 36

4.1 PFSM (PTA) of Gaines data D 51

4.2 1-state PFSM explaining Gaines data D 57

4.3 2-state PFSM explaining Gaines data D 57

4.4 3-state PFSM explaining Gaines data D 57

4.5 4-state PFSM explaining Gaines data D 58

4.6 5-state PFSM explaining Gaines data D 58

4.7 6-state PFSM explaining Gaines data D 58

4.8 PFSMs explaining the sequence $\{ABCD\#\}$ 60

4.9 PFSM with final states merged from Figure 4.1 63

4.10 PFSM with Second Stage states merged from Figure 4.9 64

4.11 Two-part Code Length comparison between the true PFSM model of Figure 4.5, the inferred PFSM model by OM and the inferred PFSM model by SA 69
4.12 Error measured while inferring PFSMs on test strings using SA and OM methods of inference 70
4.13 Sample PFSM 71
4.14 Two-part Code Length comparison between the true PFSM model of Figure 4.13, the inferred PFSM model by OM and the inferred PFSM model by SA 72
4.15 Error measured while inferring PFSMs on test strings using SA and OM methods of inference 72

5.1 *Class1* PTA 79
5.2 *Class2* PTA 79
5.3 *Class1* Inferred PFSM 80
5.4 *Class2* Inferred PFSM 80
5.5 Classification using two-machine design 82
5.6 Spam:Non-Spam ratio in each Batch of the Enron-1 dataset 90
5.7 Spam:Non-Spam ratio in each Batch of the Enron-2 dataset 90
5.8 Spam:Non-Spam ratio in each Batch of the Enron-3 dataset 91
5.9 Spam:Non-Spam ratio in each Batch of the Enron-4 dataset 91
5.10 Spam:Non-Spam ratio in each Batch of the Enron-5 dataset 92
5.11 Spam:Non-Spam ratio in each Batch of the Enron-6 dataset 92
5.12 Results obtained for each Batch of the Enron-1 dataset 94
5.13 Results obtained for each Batch of the Enron-2 dataset 94
5.14 Results obtained for each Batch of the Enron-3 dataset 95
5.15 Results obtained for each Batch of the Enron-4 dataset 95
5.16 Results obtained for each Batch of the Enron-5 dataset 96
5.17 Results obtained for each Batch of the Enron-6 dataset 96

5.18 Code Length increase in bits for the inferred PFSM models on test data when tested with test data of Person-A 105

5.19 Code Length increase in bits for the inferred PFSM models on test data when tested with test data of Person-B 106

6.1 Single PFSM model constructed from tokens of *Class1* and *Class2* . . 111

6.2 Classification using single-machine PFSM model design 115

6.3 Results obtained for each Batch of the Enron-1 dataset using the single-machine PFSM model of Figure 6.1 118

6.4 Results obtained for each Batch of the Enron-2 dataset using the single-machine PFSM model of Figure 6.1 118

6.5 Results obtained for each Batch of the Enron-3 dataset using the single-machine PFSM model of Figure 6.1 119

6.6 Results obtained for each Batch of the Enron-4 dataset using the single-machine PFSM model of Figure 6.1 119

6.7 Results obtained for each Batch of the Enron-5 dataset using the single-machine PFSM model of Figure 6.1 120

6.8 Results obtained for each Batch of the Enron-6 dataset using the single-machine PFSM model of Figure 6.1 120

7.1 An Example of Hierarchical PFSM 134

7.2 Hierarchical PFSM 135

7.3 Code Length comparison between HPFSM, PFSM (PTA), inferred PFSM and one-state PFSM for random strings 2-80 141

7.4 Code Length comparison between HPFSM, PFSM (PTA), inferred PFSM and one-state PFSM for random strings 110-800 142

7.5 Code Length comparison between HPFSM, PFSM (PTA), inferred PFSM and one-state PFSM for random strings 900-5000 142

7.6 Hierarchical PFSM 143

7.7 Code Length comparison between HPFSM, PFSM (PTA), inferred PFSM and one-state PFSM models for random strings 2-230 for the PFSM of Figure 7.6 144
7.8 Code Length comparison between HPFSM, PFSM (PTA), inferred PFSM and one-state PFSM models for random strings 260-5000 for the PFSM of Figure 7.6 145
7.9 Inferred HPFSM for Person-A 147
7.10 Inferred HPFSM for Person-B 147
7.11 Code Length increase in bits in the inferred HPFSM models on test data of Person-A 149
7.12 Code Length increase in bits in the inferred HPFSM models on test data of Person-B 149

List of Tables

2.1 Transition Table for the FSM in Figure 2.2 20

2.2 Transition Table for the PFSM in Figure 2.4 24

2.3 Transition Probability Table for the PFSM in Figure 2.4 24

4.1 Code Length of PFSM from Figure 4.1 accepting D 53

4.2 55

4.3 Code Lengths in bits for the PFSMs in Figures 4.2 - 4.7 59

4.4 Code Lengths in bits for the PFSMs explaining $\{ABCD\#\}$ and $\{(ABCD\#)^{100}\}$ 60

5.1 Composition of the Enron spam datasets 84

5.2 Enron-1 Spam Features 85

5.3 Enron-1 Non-Spam Features 86

5.4 Summary of Training and Testing Dataset in each Enron Category . . 88

5.5 Average Spam - Non-Spam ratio in each Enron category 89

5.6 Results across all Enron datasets 97

5.7 Comparison of MML and MDL approaches across Enron-1 to Enron-3 datasets 98

5.8 Comparison of MML and MDL approaches across Enron-4 to Enron-6 datasets 99

5.9 Person-A Active Daily Living (ADL) 101

5.10 Person-B Active Daily Living (ADL) 102

5.11 Assignment of Characters to Locations in the ADL datasets 103

5.12 Code Lengths and Number of States in the Initial and Inferred PFSMs of the trained PFSM models in the ADL Datasets 104

6.1 MML probability for each state of the PFSM in Figure 6.1 112
6.2 Differences between the single-machine design and two-machine design 116
6.3 Comparison of Two-Part model code lengths between the Two-Machine and Single-Machine designs 117
6.4 Results across all Enron datasets using Single-Machine model 121
6.5 Enron-1 Results achieved by each classifier 126
6.6 Enron-2 Results achieved by each classifier 126
6.7 Enron-3 Results achieved by each classifier 126
6.8 Enron-4 Results achieved by each classifier 126
6.9 Enron-5 Results achieved by each classifier 127
6.10 Enron-6 Results achieved by each classifier 127

7.1 Code Length of internal PFSM S1 from Figure 7.2 137
7.2 Code Length of internal PFSM S2 from Figure 7.2 137
7.3 Code Length of internal PFSM S3 from Figure 7.2 137
7.4 Person-A Activities of Daily Living (ADL) 146
7.5 Code length in bits given by various models for the ADL of individuals 148

Learning and Inference of Probabilistic Finite State Machines using MML and Applications to Classification Problem

Vidya Saikrishna
Monash University, 2017

Supervisor: Sid Ray
Associate Supervisor: David Dowe

Abstract

Successful models learn from a particular kind of data and for a particular learning task with the performance measured through comparative studies. Probabilistic Finite State Machines (PFSMs) are models that contain regularities and patterns of text data. The various sources generating such text data include a natural language corpus, a DNA sequence and an email text corpus. The models (PFSMs) are used for analysis of the text data such as classification and prediction. This research work is focussed on learning PFSM models from two classes of text data under a supervised learning environment. The model is a hypothesis and the information-theoretic Minimum Message Length (MML) principle is used to judge the goodness of the hypothesis in relation to prediction or classification, in different situations. In short, MML has been used as a technique to select among the competing PFSM models. We propose three novel approaches for classification and prediction. We apply the approaches to two problems. The first approach is concerned with learning two-machine PFSM models for a two-class classification of text data. Two models are trained with the training data of the individual classes. The PFSM models are inferred by using MML as an objective function in the Simulated Annealing search process. The power of MML in giving a description length of model such as PFSM has been used in this research and the description length is returned in terms of two-part code length of the model. The models inferred from the training dataset are then consequently tested with the test dataset. The unknown label of

the test dataset is probabilistically estimated by measuring the amount of increase in the two-part code length of the two inferred models and the model that measures minimal increase in code length is chosen. In the second approach, the first approach is modified to learning a single-machine PFSM model for the two classes of text data. The advantages with this approach are as follows. First, the single-machine approach is more space efficient than the two-machine approach. Second, the model need not be inferred and the obvious benefit is, the model is learnt very quickly. A single PFSM model is built from the two classes of text data. There is a binomial distribution at the accepting states of the PFSM as they are reached by the text seen in both classes of text data. The MML probabilities for the binomially distributed accepting states are combined and the combined-probability is measured against a threshold. The combined-probability greater than threshold results in the test data being classified in one class and less than the threshold results in the test data being classified into other class. The third approach that we propose for classification of text data under a two-class learning environment is the idea of learning hierarchical PFSMs or HPFSMs. This is an important contribution arising out of this research on accounts of its novelty and experimentally proven good results. We discuss a method of encoding HPFSMs and compare the code length of the HPFSM model with the traditional PFSM model. For a text data, if the inherent hierarchies are somehow found or assumed, then learning the HPFSM model for that text data is cheaper than learning PFSM model for the same data. We show this comparison on at least two artificially generated HPFSM models and also on some publicly available datasets. The HPFSM models learnt are then used for classification or prediction in a similar way as discussed in the first approach. The three approaches proposed are applied on two application areas. The first application is the classification of spam and non-spam emails using the Enron spam datasets and the performance of the proposed methods are evaluated in terms of classification measures such as precision, recall and accuracy. The second application is the prediction of individuals using the Activities of Daily Living (ADL) datasets gathered from the University of California

at Irvine (UCI) machine learning repository and the performance is evaluated in terms of prediction of individuals. Existing methods such as Minimum Description Length (MDL) and Naïve Bayesian classifiers are reviewed. These methods work on constructing the Bag of Words (BOW) model for the text data as opposed to the PFSM model. The benchmark results generated from our methods are compared with these existing methods. The contributions are summarized as: 1) We propose a novel idea of HPFSMs and come up with a coding scheme that describes an HPFSM model. 2) We propose methods for classification and prediction by learning HPFSM and PFSM models. 3) We compare our results with the other methods such as Minimum Description Length (MDL) and Naïve Bayesian classifiers. The two-machine approach resulted in giving 99.75% average classification accuracy as compared to the 98.60% average classification accuracy by the MDL method on the Enron spam datasets. On the ADL datasets too, the two-machine approach resulted in accurate prediction of the individuals from the test dataset of the individual classes. The experiments done with the HPFSM models on the artificially generated datasets and on the ADL datasets show the strength of HPFSM models in terms of best compression achieved when compared to non-hierarchical models.

Learning and Inference of Probabilistic Finite State Machines using MML and Applications to Classification Problem

Declaration

I declare that this thesis is my own work and has not been submitted in any form for another degree or diploma at any university or other institute of tertiary education. Information derived from the published and unpublished work of others has been acknowledged in the text and a list of references is given.

Vidya Saikrishna
June 15, 2017

Acknowledgments

Firstly, I would like to express my sincere gratitude to my supervisors **Dr. Sid Ray** and **Associate Prof. Dr. David Dowe** for their continuous support through out my Ph.D study and related research, their patience, motivation and immense knowledge. Their guidance helped me in all time of research and writing of this thesis. I could not have imagined having better advisers and mentors for my Ph.D study.

Besides my supervisors, I would like to thank the rest of my thesis committee: Dr. Bala Srinivasan, Dr. Reza Haffari and Dr. Nandita Bhattacharjee, for their insightful comments and encouragement, but also for the hard questions which motivated me to widen my research from various perspectives. My sincere thanks goes to Dr. Sue McKemmish, for without her support in providing the faculty scholarship for the course fee, the journey would have been immensely difficult. I also owe thanks to Helen Cridland and Danette Deriane for guiding me through the process of applying for grants and scholarships at various stages in my candidature.

A special thanks to my family. Words cannot express how grateful I am to my father-in-law, mother-in-law, mother and, my brother for supporting me in all dimensions when the initial thought of Ph.D came to my mind. Their constant prayers and sacrifices sustained me thus far. I would like to express appreciation for my husband Saikrishna and my son Sai Vighnesh who spent sleepless nights and were always my support in the moments when there was no one to answer my queries. Especially my son who was very understanding and never bothered with his playtime throughout the journey of my Ph.D, I really owe a special thanks to him.

Finally, I would like to thank all my friends and specially Komal, for she allowed me to stay with her on the weekends and supported me in writing and striving towards my goal.

Vidya Saikrishna

Monash University

June 2017

Chapter 1

Introduction

This thesis examines the problem of learning models from text data and the success of the models is measured by doing a comparative analysis. The text data are seen as regular language and the inference of regular languages has important applications in the field of artificial intelligence, pattern recognition and data mining. The problem of inference of regular languages could be specifically related to applications such as modelling text, text classification and predication. The problem that is considered is explained like this. Suppose we have a dataset that represents a sequence of tokens separated by some stopping symbols or delimiter symbols. The model or the hypothesis that is the source of generation of the data has somehow become non-existent. Now, from here, various questions arise into our mind. First, is it possible to reconstruct the source from the evidence? The question could be interpreted this way also, what methods exist that try inferring the source from the evidence? Second, if it is possible to do so, then what objective function will help us reconstruct the source? Third, if there are more than one competing sources that can all generate the data, then which one of them should be considered as the most appropriate one?

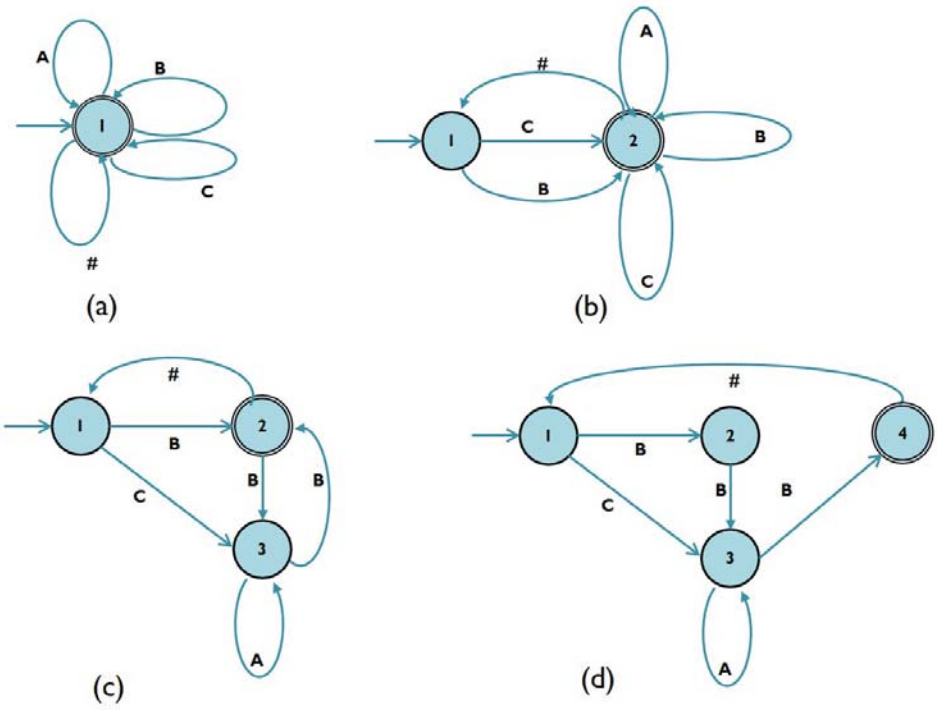
Let us discuss the problem in more detail. The data as mentioned above can be seen as a sequence of tokens separated by delimiter symbols. Now there are many sources of such kind of data. The sequence might get generated from a natural language corpus and the tokens may represent the elements of the natural language.

Here the data can be, conversation, written composition, reading, dictation, translation or lip reading. The data source can be emails with text in them. The data source can be a DNA sequence. The data sequence can refer to the movements of an individual while doing transitions from one place to another as part of his daily routine activities. The data sequence might represent language generated by a finite state machine. Similarly many examples can be thought of like this where the data can be easily transformed into a sequence of words separated by full stops. The data at this point of time look not more than a Bag of Words (BOW), where the key idea is to quantize each token as a histogram by counting the number of occurrences. The BOW model is a useful representation for data and lots of useful analysis can be done using the model like, e.g. text classification. But for a bulky information source, the BOW model may not be the best representation model for the data. The data can be alternatively represented by using a compression based technique while achieving the same benefits of the BOW model.

In order to describe the data, a suitable model needs to be selected. According to Wallace (Wallace, 2005), the model is an assertion on the observed data and the inductive inference of the model tells how probable the model is, in generating the data. The data in consideration are text data separated by delimiter symbols. For such kind of data without much complexities, a fairly simple model such as the Probabilistic Finite State Machine (PFSM) can be used. A finite state machine is a compact representation of a sequence of words and the probabilistic version of a finite state machine that describes a population of words can be inferred using a statistical inference tool such as Minimum Message Length (MML).

Inductive inference has a special role to play in model selection. The information-theoretic MML assigns a score to the model based on the structure of the model. The score can be used as a basis of comparison between the competing models. The score is calculated in the form of code length and the code lengths of the different competing models are compared. The model that gets the least score is considered as the best model. This is true because “The best explanation of the facts is the

shortest". For centuries this idea has been proposed and is generally accepted to be more or less true subject to qualifications and exceptions (Wallace, 2005). According to MML model selection criterion, the problem of inference is considered as a two-part code communication between the hypothetical transmitter and receiver. The first part code comprises the model or the hypothesis and the second part code comprises the data generated by the hypothesis. Therefore, the different models vary in their code lengths and the one with least total code length is regarded as the best one.



4

Figure 1.1: Probabilistic Finite State Machines generating L

To understand the process of inductive inference using MML, let us consider a small amount of observed data from Gaines (1976). Let the data be explained by language L where

$L = \{CAB\#CAAAB\#BBAAB\#CAAB\#BBAB\#BBB\#CB\#\}$. The words or tokens in the language L are separated by the stopping symbol #. The model that is suitable for modelling such sequence is a Finite State Machine. Now, consider

the different finite state machines in Figure 1.1 that are probable candidates for generating such a data.

Now each one of the models in Figure 1.1 can generate the sequence denoted by language L. So which one of these probable models should be considered as the best one? The answer lies in the two-part code length of the different models of Figure 1.1. The two-part code length is calculated using MML. Thus, inductive inference will help us in finding the most suitable model. The problem of model selection is crucial as any further analysis is based on the selected model.

Before explaining what the thesis contributes, I would like to briefly explain MML over here. The detailed explanation follows in Chapter 3.

MML is an information-theoretic approach used for comparing the different hypotheses that are the sources for generation of data. MML works on the Bayesian principle of calculating the posterior probability of a hypothesis given the data (Wallace and Boulton, 1968; Wallace, 1990; Wallace and Dowe, 1999b,a; Dowe, 2008a, 2011) and states that given the data, the best possible conclusion about the data can be drawn from the theory that attempts to maximize the posterior probability. Maximizing the posterior probability equivalently means maximizing the product of prior probability with the probability of data given the theory or hypothesis or likelihood. MML has emerged as a powerful tool, not only in providing a coding mechanism for Probabilistic Finite State Machines (PFSMs) but, also plays an important role in the inductive inference of discrete structures as PFSMs. An elementary information theory concept based on Huffman code converts the probability values into the form of code length, in bits. So, to compare the different hypotheses, their code lengths are compared.

It is worth mentioning that there are essential differences between MML and Bayesianism. Classical statistics typically postulates a distribution with an unknown parameter, θ, as a single model, and then attempts to infer such a (vector) θ. Classical statistics is interested largely in point estimation, but can also be interested in interval estimation and giving confidence intervals. The Bayesian approach takes a

Bayesian prior distribution, and can then proceed in a variety of different ways. One approach is simply to arrive at a posterior distribution. Another Bayesian approach is to take the posterior distribution and to obtain a point estimate by minimising an expected loss function - see, e.g., Lehmann and Casella (2006); Dowe et al. (1998). Other Bayesian point estimators exist, such as (e.g.) Minimum Message Length (MML) and various approximations (see, e.g., Wallace (2005); Dowe (2008a, 2011), and also the Bayesian maximum a posteriori (MAP) approach. There are also differences between MML and relatively recent MDL (Minimum Description Length). The MDL method is attributed to Jorma Rissanen (1978) (Rissanen, 1978) (Rissanen, 1999) and its objective is to find the best model class and the best model in it is found by the shortest code length. The MML method on the other hand is attributed to Chris Wallace and David Boulton (1968) (Wallace and Boulton, 1968) and is concerned with finding the model that is the best explanation of the data. An even more important distinction between MML and MDL is the use of subjective priors (Raman et al., 1998)(Wallace and Dowe, 1999b) (Baxter and Oliver, 1994). The MML considers and believes in incorporating the prior knowledge to the best possible extent, whereas MDL considers a uniform or universal prior thus ignoring the prior information.

The thesis work explores the application areas where MML could be used in inductive inference and where there are no accounts of any previous work done before using MML. We explain the applicability of MML to these areas with the explanation of the different problem contexts and see how MML can be used to find a solution to the raised problems.

1.1 Text classification using MML

The problem in concern is text classification using the Minimum Message Length (MML) principle. The problem can be put down by asking the following questions.

How can an approach for text classification be developed that is adaptive to the changing characteristics of input data, very much takes care of the prior information

into account, uses a kind of representation to model the input data which is efficient and can be effectively induced and, most importantly gives accurate classification results which are at par with the other established text classification techniques?

The answer is explained like this below.

Given a text corpus with already classified matter of text in different categories. The best example could be emails with classified emails in spam and non-spam category. An approach for classification of a new email with unknown class can be developed using MML. This is an area where MML has not been explored before. With MML we try to answer the above questions in the following way by developing three approaches.

1.1.1 Proposed Method-1

Emails as we know, can be viewed as collection of tokens or attributes. The tokens or attributes can be converted to a form which resembles the sequence belonging to language L as mentioned in the Introduction Section 1. Some attribute selection procedures can be employed to reduce the dimensionality of the attribute space. After having done this, a suitable model is selected that combines the two insights i.e., finding regularities or patterns with the idea of compressing. And the suitable model is the Probabilistic Finite State Machine (PFSM).

As the email corpus consists of two classes, the two PFSMs are built from the attributes belonging to the two categories. The PFSMs are inferred using the MML induction technique and simulated annealing heuristic is used that does an extensive search of the search space and infers the best model. Any model is explained by a two-part code length where one part is the model itself and the other part is the data encoded using the model. After having inferred the PFSMs, the target document is input to the two inferred PFSMs built from the two classes. The inferred PFSM that results in minimal increase in the two-part code length is more probable to have generated the tokens of the unknown target and, thus the class can be easily identified.

Now, there are accounts of email texts being classified by more or less similar approaches such as the Minimum Description Length (MDL)(Almeida and Yamakami, 2012), various versions of the Naive Bayes methods (Metsis et al., 2006) and statistical data compression models (Andrej Bratko, Gordon V. Cormack, Bogdan Filipič, Thomas R. Lynam, Blazupan, 2006). The MDL spam classification technique uses the *Maximum Likelihood* (ML) estimation for calculating the probability of the unknown parameter with no sufficient literature on how the compression was actually achieved. The prior information is also completely ignored. All the Naive Bayesian methods work on the BOW representation of the model that does not do any compression of the model. They also use a threshold value for classification where over-setting and under-setting the threshold result in either high false positives or high false negatives increasing the classification error rate.

We overcome the limitations of the above methods by using MML. The models inferred using MML are incrementally updateable and adaptive to the changing characteristics of the input data. The MML two-machine model results in competitive classification accuracy that can be compared with the other approaches of similar kind and the best part is, it achieves compression. If desired, the extensive search of the problem space can be compromised by using a greedy search strategy to infer the best model. As far as classification is concerned, without compromising the classification accuracy, the search time can be reduced by using the greedy search heuristic.

1.1.2 Proposed Method-2

The idea of using data compression in classification and other machine learning tasks in some manner or the other has led to production of many research results. This intuition arises from the principal observation that compact representation of objects are possible only after some recurring patterns or regularities are detected. This has motivated many applications of data compression algorithms to be used in machine learning and data mining problems.

The idea of the two-machine approach is not new. Most of the data compression techniques doing classification rely on building two models. What is new is building a single-machine model that is compact and can achieve comparable classification results. The proposed method-2 focuses on how building a single-machine model can be done using MML.

In the single-machine model, we build a single machine of the sequences seen in both classes of emails. And therefore is a need for an additional code length factor to be included in the code length calculation of the model along with the traditional method of calculating the code length of any PFSM model. But, inspite of this slight increase in the code length due to this additional factor, the overall code length of the single-machine model is less as compared to the added values of the code lengths of the two machines in the two-machine model. The good part of this new approach is without compromising the classification accuracy, the compression achieved is more.

The unknown class of the target document is obtained by estimating the probability of the state reached in the single-machine PFSM model. The states in this PFSM model are reached by reading both spam and non-spam emails. Therefore MML estimation for binomial distribution can be applied to obtain a probable answer. This probable answer is compared against a threshold and the target document is accordingly classified.

The two-machine classification problem can be generalized to n-machine classification problem, but building n-models for n classes would turn very expensive. By the use of the single-machine model, the n-models can be replaced by single model by the above mentioned method and the benefits are evident.

1.1.3 Proposed Method-3

The third proposed method where MML has not been applied yet is encoding hierarchical data. Now how do hierarchical data gets generated? To understand this we consider a few examples and also consider how MML can be used in encoding of the

hierarchical data. The obvious benefit by encoding the hierarchical machine is realized in terms of the two-part code length of the hierarchical model when compared with the non-hierarchical model.

As our first example, we consider a case where we are trying to encode the conversations of a multilingual person. A multilingual person speaks more than one language. We consider an example of a person speaking three different languages. The vocabularies of the three languages are different. Now, while speaking, the person does transitions from one language to another. The traditional method of encoding would encode the sentences one by one by making state transitions in the finite state machine and, at the end of each sentence, the start state is again reached. This method of encoding is completely non-hierarchical and the code length calculation has to include the vocabularies of all the languages. Now, if the conversations are carefully observed, we would understand that while speaking one language, the person tries to be in the domain of that language and his transition to other language is quite infrequent. The different languages can constitute small machines and the small machines are somehow connected externally. The external links may represent transitions from one language to another. This model representation is hierarchical and the code length calculation has to accordingly incorporate the hierarchy.

Another example why coding in hierarchy would turn beneficial could be understood this way. Let us say we have a big machine that represents our movements in a day, in the form of state transitions, from one place to another. Broadly speaking, the places that we may visit are, "university or work place", "home" and "city". We start from a local place say home. Home represents one small internal PFSM. At "home" we do state transitions locally more often by visiting different places at "home". There is one point of entry (start state) to "home" and the same point can be used as the exit point from this "home" local machine. Then we transit from "home" to another internal PFSM, say "university". We enter into the start state of the "university" local PFSM and again we perform state transitions my moving

around different places in the "university". The start state of the "university" internal PFSM is also the exit state from it, as we had in the "home" internal PFSM. If we are to specify a record of the daily movements that involves moving around various places, in the form of code length, then considering the whole picture as a non-hierarchical PFSM, would turn to be very expensive. Whereas if we do the same through hierarchical coding mechanism, we would definitely get a cheaper encoding of the daily movements.

Thus, proposed method 3 is focused on giving a hierarchical coding scheme for the hierarchical machine. The thing that requires to be done is to identify hierarchy in data and try converting it to a hierarchical representation. With sufficiently large amount of data, the coding will always be cheaper. For insufficient amount of data, we would be unnecessarily paying for the complex structure, as the hierarchical structure is always more complex than the non-hierarchical structure.

1.2 Contributions

The contributions arising out of the research work are understood in the following way:

- Modelling data using Probabilistic Finite State Machines.

  For this, applications need to be explored where data can be viewed as a sequence of tokens separated by delimiter symbols. Two such applications areas were sought. One is the *Enron-Spam* Dataset with classified spam and non-spam emails. The other one is obtained from the UCI machine learning repository, which is *Active Daily Living (ADL)* dataset that records the daily movements of a person as part of his daily routine from one place to another.

  The first dataset, that is the *Enron-Spam* dataset was used in text classification using our proposed approaches. The second dataset was found useful for building the Hierarchical PFSM model.

- Develop an induction mechanism for inference of Probabilistic Finite State Machine. Here we are proposing a new method of induction that works on greedy lines of finding an optimal solution. The optimal solution is the best PFSM and the objective function that helps finding the optimal solution is MML. We also used the simulated annealing approach to do an extensive search of the search space.

- Develop a two-machine model for text classification.

  The two models here represent the two classes. The evaluation criteria used are the common evaluating measures such as precision, recall and accuracy. Although compression is the main idea underpinning the whole thesis, other literatures surveyed that work on statistical compression techniques for text classification, do not mention the amount of compression achieved by their methods. That is the reason why we felt the need to compare with the other techniques using the above mentioned evaluation measures.

- Develop a single-machine model for text classification.

  The same evaluating measures apply here also. But here we compare with the two-machine model to show how much better compression the single-machine model achieves without affecting the classification accuracy.

- Develop hierarchical encoding mechanism for hierarchical PFSMs.

  The hierarchical model code length is compared with the non-hierarchical model code length to show the benefits of hierarchical encoding.

1.3 Thesis Outline

The thesis is organized in the following manner.

Chapter 2 focuses on highlighting the key definitions related to Regular Languages, Finite State Machines (FSMs) and Probabilistic Finite State Machines (PFSMs). The observations are assumed to be strings separated by delimiter symbols

and there are many such sources for getting such kind of data as discussed earlier in Introduction Section of Chapter 1. The sequence of strings may be treated as a regular language and any regular language can be modelled using a Finite State Machine. The Finite State Machine that exists for any regular language is either capable of accepting all the strings in the sequence or it can also be viewed as a machine generating the sequence of strings. A hypothetical FSM generating or representing a collection of tokens (words) in a finite alphabet might contain regularities that are not fully captured by the formal grammar. Therefore, the simple FSM model is extended to include some probabilistic structure in the grammar. The model is now termed as Probabilistic Finite State Machine (PFSM). A PFSM, along with generating a set of possible tokens, also defines a probability distribution on the set. The PFSM model needs to be inferred to identify the true source of data generation.

Chapter 3 begins with the formal definition and description of the Minimum Message Length (MML) principle. In this chapter we describe the formal code length calculation for binomial and multinomial distributions which are used in the code length calculation for probabilistic finite state machines. The states in the PFSM model have a binomial distribution if a two-class classification problem is concerned and will have a multinomial distribution if n-class classification problem is concerned.

Chapter 4 discusses the MML code length calculation on PFSMs. The code length calculation is done in two parts, with the first part encoding the PFSM model itself and the second part encoding the data generated by the model considered in first part. We also discuss few MML induction methods that help us in getting the minimal code length PFSM along with a discussion of our own induction method to obtain the optimal code length PFSM.

Chapter 5 discusses the two-machine model for text classification where the models are the PFSMs of the classes of data available from the training corpus. We discuss the method of classification on artificial dataset first followed by experimentation on the *Enron-Spam* datasets and the *Activities of Daily Living (ADL)*

datasets. We compare the results generated by the two-machine MML approach with the MDL approach. We discuss the possibility of applying the approach of two-class classification problem to n-class classification problem, although we defer the experimentation for future work for the n-classes. We also explain that the n-model concept can be better replaced by the single-model concept to have more compact encoding.

Chapter 6 is focused on implementing the single-machine model approach for text classification. The experimentation is followed in a similar manner as in the two-machine model classification problem. We compare the results generated by the single-machine MML approach with the different versions of the Naive Bayes classifiers.

Chapter 7 describes the hierarchical encoding of PFSMs. We get benefited from hierarchical encoding if the data have inherent hierarchy in them. The benefit is gained in terms of code length of the hierarchical model. Thus this chapter describes coding scheme to generate the code length of a hierarchical PFSM model. The method of encoding is followed by experimentation on artificial or pseudo randomly generated hierarchical PFSM model and also on the *Active Daily Living* (ADL) datasets.

Chapter 8 concludes the thesis by summarizing the contributions and discussion of the key results gained by contributions. We also discuss the possibilities of future work that can be carried out in extension to the work already done in this thesis.

Chapter 2

Finite State Machines (FSMs) and Probabilistic FSMs

2.1 Introduction

In this chapter we discuss the formal theory related to Finite State Machines (FSMs), Regular Languages and Probabilistic Finite State Machines (PFSMs).

In brief, a Finite State Machine (FSM) is a mathematical model of computation which can effectively describe grammar of a language. The grammar described by this model of computation is called a regular grammar. The model plays an important role in several applied areas of computer science, of which, text processing, compiler and hardware design are the few common ones (Sipser, 2006). Finite State Machines can effectively represent regularities and patterns. For this reason, the words or tokens which are generated out of a natural language corpus can be effectively modelled using a Finite State Machine as the corpus contains many word repetitions.

A hypothetical grammar (FSM) generating or representing a collection of strings (sentences) in a finite alphabet might contain regularities that are not fully captured by the formal grammar. We can extend the simple FSM model to include some probabilistic structure in the grammar. The model is now termed as a Probabilistic

Finite State Machine (PFSM). The grammar represented by a PFSM is called a probabilistic regular grammar. A PFSM, along with generating a set of possible strings, also defines a probability distribution on the set (Wallace, 2005, sec. 7.1.2).

2.2 Finite State Machines

Finite State Machines (FSMs), also called Finite State Automata (FSAs), provide a simple computational model with many applications. There are many ways of modelling behaviour of systems, but the use of state machines is one of the oldest and the best known (Wright, 2012). State machines are useful to model events happening in some sequence or sequence of events happening in time series. State machines allow to think about the "state" of a system at a particular point in time and characterize the behaviour of the system based on that state. Many natural and artificial systems may be modelled by defining (Wright, 2012):

- The possible states a system can occupy.
- The behaviour of the system in those states.
- How the system transitions between the states based on the input signal received.

Let us consider a simple example of a stopwatch to understand the concept of finite state machines. A stop watch has two states. One is the "Stopped" state and the other one is the "Running" state. The behaviour is modelled using the state diagram below.

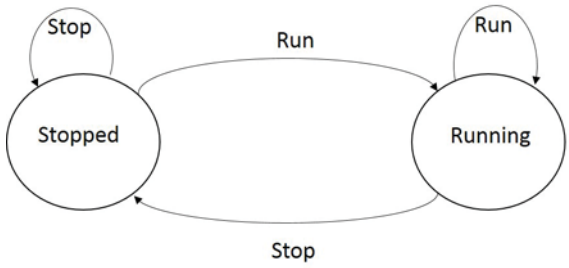


Figure 2.1: Behaviour of a stopwatch represented as an FSM

The stopwatch system starts in the "Stopped" state. In the "Stopped" state, when the input "Run" is received, the system does a transition from "Stopped" to "Running" state. Whereas, in the same state, if the input is "Stop", the system remains in the same state.

Thus, from the discussion above, several key characteristic features related to a finite state machine can be drawn. The key features are:

- The system must be describable by a finite number of states.
- The system must have a finite set of inputs that can trigger a transition between the states.
- The system must have a start state.
- The behaviour of the system at any point of time is dependent on both the current state and the current input symbol.

The key features of a finite state machine can be formally put down using a mathematical notation.

2.2.1 Mathematical Definition of a Finite State Machine

A Finite State Machine M is defined as a 5-tuple where $M = <Q, \sum, \delta, q_0, F>$. The tuples are defined as below (Lenhardt, 2009):

- Q is the set of states of M
- $\sum$ is the set of input alphabet symbols
- δ is the transition function mapping $Q \times \sum$ to Q
- q_0 is the initial state
- F is the set of final states of M

There are few other definitions associated with the finite state machines (Lenhardt, 2009). They are defined as below:

Definition 1: A string or token or word over an alphabet $\sum$ is a finite sequence of symbols from $\sum$.

Definition 2: A language over an alphabet $\sum$ is a set of strings over the alphabet $\sum$.

Definition 3: A string $\mathbf{x} = (x_1 x_2 ... x_j ... x_n)$, where x_j is a member of input alphabet set $\sum$ and $1 \leq j \leq n$, is accepted by finite state machine M, if the sequence of states reached, while reading $\mathbf{x}$, is $s_0 s_1 ... s_n$ and the following conditions are met.

1. $s_0 = q_0$

2. $\delta(s_i, x_{i+1}) = s_{i+1}$, for $i = 0, ..., n-1$

3. $s_n \in F$

Definition 4: The language accepted or recognized by M, denoted by L, is the set of all strings $\mathbf{x}$ accepted by M.

Definition 5: The language L accepted by M is called a regular language.

Definition 6: The machine M is called a **Deterministic FSM**, if for a given state and a given input symbol, the next state is known.

Definition 7: The machine M is called a **Non-Deterministic FSM**, if for a given state and a given input symbol, the next state is uncertain. In other words, there is more than one state that can be reached for the given state and the given input symbol.

2.2.2 Representation of an FSM in a state diagram

The mathematical definition of a finite state machine can be easily transformed into a state diagram as shown in the Figure 2.2. The FSM in Figure 2.2 corresponds to language L where

$L = \{CAB, CAAAB, BBAAB, CAAB, BBAB, BBB, CB\}$. The language is adopted from the reference (Gaines, 1976).

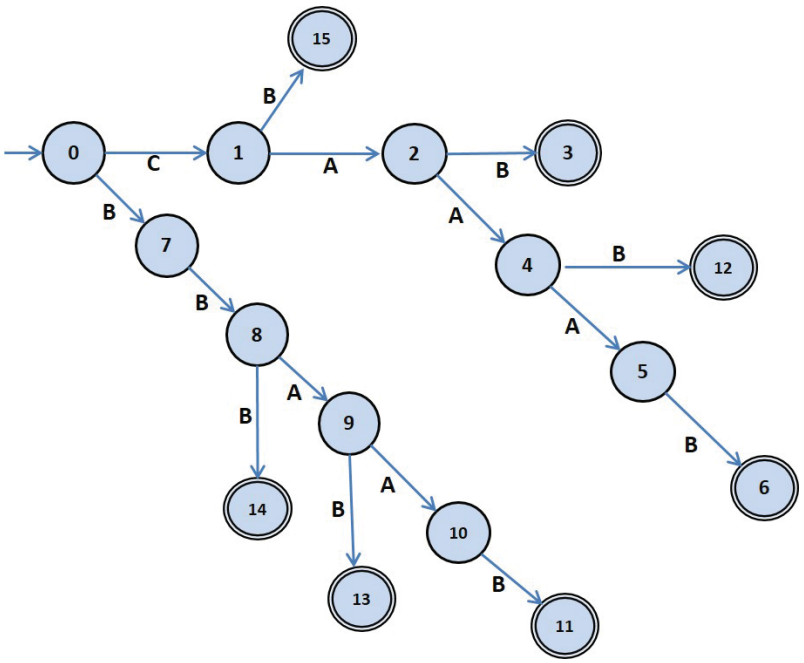


Figure 2.2: Finite State Machine accepting the above mentioned language L

All the strings in language L get accepted by the Finite State Machine in Figure 2.2. The FSM starts with state 0 and reads the string belonging to language L in a sequence by making state transitions. The final states shown in the FSM of Figure 2.2 are marked by double circles over the state numbers. After reading the complete string, the FSM again begins with state 0 to read the next string. Such FSM where the common prefixes of the different strings are marked by common states is known as the **Prefix Tree Acceptor (PTA)** of the strings of language L (Saikrishna et al., 2015).

The transitions in the FSM of Figure 2.2 can also be understood by the transition matrix in Table 2.1.

The transition matrix in Table 2.1 shows the next state transition from the current state on a particular input symbol. The blank entries in the table or the entries marked "-" are a method of specifying that there are no state transitions from the current states on those input symbols.

As we see in the FSM model of Figure 2.2, the regularities or patterns in the language can be easily handled, the model becomes very suitable for applications

Table 2.1: Transition Table for the FSM in Figure 2.2

Current State	**Next State for Symbol**		
	A	**B**	**C**
0	-	7	1
1	2	15	-
2	4	3	-
3	-	-	-
4	5	12	-
5	-	6	-
6	-	-	-
7	-	8	-
8	9	14	-
9	10	12	-
10	-	11	-
11	-	-	-
12	-	-	-
13	-	-	-
14	-	-	-
15	-	-	-

which have word repetitions or patterns. For this very reason, the FSMs can be used to model the elements of a natural language corpus or even a sequence of events.

Let us again considering the language L, where $L = \{CAB, CAAAB, BBAAB, CAAB, BBAB, BBB, CB\}$. The language L is considered as data or observations and the strings in the data can be considered as tokens or words or even sentences. Let us assume that these tokens of language L are getting generated from an FSM whose true structure is not known. The tokens belonging to language L arrive in the same order as they are mentioned in the sequence and we separate them by a delimiter symbol # to distinguish the tokens from each other. The same language L now looks like $L = \{CAB\#CAAAB\#BBAAB\#CAAB\#BBAB\#BBB\#CB\#\}$.

To model this sequence, we again start with some initial FSM and the initial FSM is the Prefix Tree Acceptor (PTA) of the language L. While reading the tokens one by one, whenever the symbol # is read, the machine again starts with the initial state of the FSM. In other words, the symbol # on the current state forces the

machine to do a transition to the initial state from the current state. Figure 2.3 shows a hypothetical Finite State Machine generating the tokens of language L.

The FSM in Figure 2.3 lacks the expressibility of denoting regularities in terms of frequency of probability of occurrence of certain structures. Therefore, the finite state machines are incapable of representing regularities that are not fully captured by formal grammar. For this reason, the FSM model is extended to include some probabilistic structure in the grammar. The extended model is now termed a **Probabilistic Finite State Machine (PFSM)**

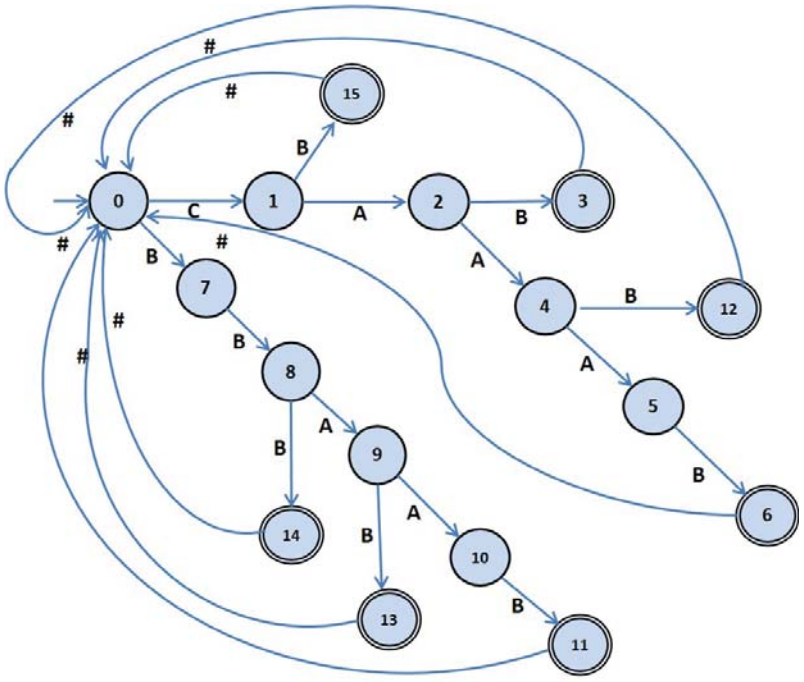


Figure 2.3: Hypothetical Finite State Machine generating L

2.3 Probabilistic Finite State Machines

Probabilistic Finite State Machines (PFSMs), also known as Probabilistic Finite State Automata (PFSAs), is an extension of a finite state machine in the form of a probabilistic structure capable of representing probabilistic transitions among states (Wallace, 2005, sec. 7.1.2)(Wallace and Georgeff, 1983). The probabilities on the arcs from any state sum to one. Any input string is accepted with a certain probability in a PFSM. The grammar represented by a PFSM is called a probabilistic

regular grammar. A PFSM defines not just a set a possible strings but also defines a probability distribution on the set. We define more precisely what PFSMs are and introduce necessary notations in connection to the use of PFSMs.

2.3.1 Mathematical Definition of a Probabilistic Finite State Machine

The PFSM is formally defined as a 6-tuple $M = <Q, \sum, \delta, \pi, q_0, F>$. The tuples are defined as below:

- Q is the set of states of M
- $\sum$ is the set of input alphabet symbols
- δ is the transition function mapping $Q \times \sum$ to Q
- π is the probabilistic function mapping $Q \times \sum$ to $[0, 1]$. This defines the probability of the next symbol in a given state. The sum of all probabilties from a given state sum to 1.
- q_0 is the initial state
- F is the set of final states of M

2.3.2 Representation of a PFSM in a state diagram

In the FSM of Figure 2.3, the transition arcs are additionally labelled with transition probabilities to transform the representation into a PFSM representation (Wallace, 2005, sec. 7.1.2).

Thus for the same language L where,
$L = \{CAB\#CAAAB\#BBAAB\#CAAB\#BBAB\#BBB\#CB\#\}$, the PFSM is shown in Figure 2.4.

When we construct an FSM from a set of sample strings, we can estimate the transition probabilities by keeping a count of the number of times each arc of the

graph is traversed (Saikrishna et al., 2015). The counts can be converted into probability estimates by dividing each count by the total count of all the arcs from that state. This turns the FSM to PFSM. The PFSM essentially tells the probability of transition from one state to another state on seeing a particular symbol from the finite alphabet set.

Table 2.2 shows the transition matrix for the PFSM in Figure 2.4 and Table 2.3 shows the transition probability table for the same Figure 2.4. From any state in Table 2.3, the sum of probabilities for all the symbols is equal to 1.

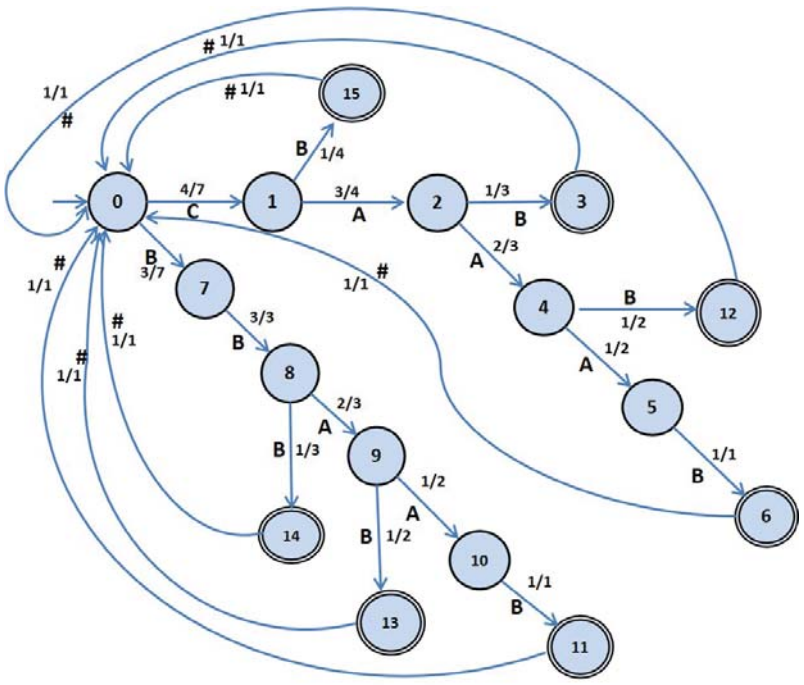


Figure 2.4: Probabilistic Finite State Machine generating L

Table 2.2: Transition Table for the PFSM in Figure 2.4

Current State	**Next State for Symbol**			
	A	**B**	**C**	#
0	-	7	1	-
1	2	15	-	-
2	4	3	-	-
3	-	-	-	0
4	5	12	-	-
5	-	6	-	-
6	-	-	-	0
7	-	8	-	-
8	9	14	-	-
9	10	12	-	-
10	-	11	-	-
11	-	-	-	0
12	-	-	-	0
13	-	-	-	0
14	-	-	-	0
15	-	-	-	0

Table 2.3: Transition Probability Table for the PFSM in Figure 2.4

Current State	**Next State for Symbol**			
	A	**B**	**C**	#
0	0	0.43	0.57	0
1	0.75	0.25	0	0
2	0.66	0.33	0	0
3	0	0	0	1
4	0.50	0.50	0	0
5	0	1	0	0
6	0	0	0	1
7	0	1	0	0
8	0.66	0.33	0	0
9	0.50	0.50	0	0
10	0	1	0	0
11	0	0	0	1
12	0	0	0	1
13	0	0	0	1
14	0	0	0	1
15	0	0	0	1

2.4 Equivalence of FSMs and PFSMs

To understand the equivalence of two finite state machines (FSMs), let us consider two finite state machines $M1$ and $M2$ shown by their state diagrams in Figure 2.5 and Figure 2.6 respectively.

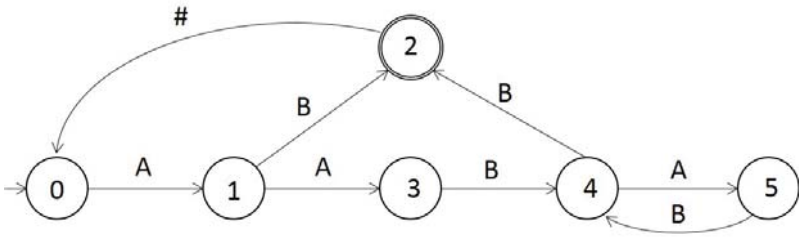


Figure 2.5: Finite State Machine M1

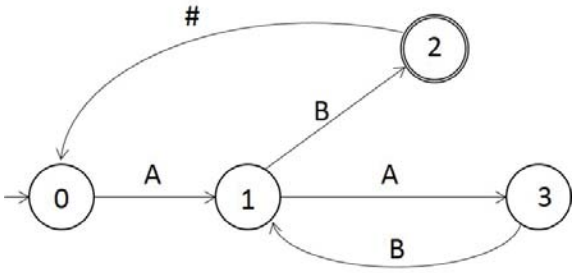


Figure 2.6: Finite State Machine M2

Let the language generated by machine $M1$ be $L1$ and the language generated by machine $M2$ be $L2$. The languages are generated from the grammar represented by the two FSMs. An FSM generates a sequence from the grammar starting from the initial state of the FSM and terminating at the final state of the FSM. The final state in the FSM is known by the double circle over the state and it reads the character # in that state. As in the "English Language Grammar", the sentences are terminated by a "Full Stop", here the symbol # has the same significance as of a "Full Stop" in "English Language Sentences". Therefore we say that the FSM generates sentences. The sentences are generated in the following manner:

The machine starts in *state 0*. It then moves to the next state by following one of the arcs leaving *state 0*. When it moves to the next state, it generates a symbol labelled by the arc that is followed. From the next state, again by following one of the arcs, the next symbol gets generated. The same steps are continued until the

arc labelled # is followed. The arc labelled # marks the end of sentence and takes the machine again to the start state. The sentences after that are generated very much in a similar manner like those with *state 0*.

Following the method of sentence generation above, machine $M1$ generates the following sentences denoted by $L1$.

$L1 = \{AB\#AABABB\#AABB\#AABABABB\#...\}$

Machine $M2$ also generates the same set of sentences when the arcs are followed in that machine. So the language generated by machine $M2$ is the same as language $L1$.

$L2 = \{AB\#AABABB\#AABB\#AABABABB\#...\}$

Two FSMs are regarded as equivalent if they generate or accept the same set of sentences.

When the machines $M1$ and $M2$ are given transition probabilities, they can no longer be considered equivalent. This is because assigning transition probabilities to the arcs of an FSM transforms an FSM model into a PFSM model and a PFSM model not only defines possible sentences but it also defines a probability distribution on that set. For instance, machine $M1$ can represent a grammar such that 70% of all generated sentences are "AB", 20% of all sentences are "AABB" but the sentences beginning with "AABA..." have an average length of 100 symbols. Machine M cannot simply represent such a population. Therefore the PFSMs cannot be considered as equivalent (Wallace, 2005, sec. 7.1.2).

2.5 Summary

This chapter discussed the basic theory related to finite state machines and probabilistic finite state machines. These models are useful in modelling sequential data where regularities or patterns can be detected. To understand a finite state machine (FSM), we consider the example of a stopwatch to explain the process of state changes in a stopwatch through an FSM model design. An FSM model is defined by identifying the set of states, set of input alphabet symbols, the initial state, set

of final states and the transition function mapping the states to the input alphabet symbols. We get familiar with a few definitions used in context of finite state machines. The finite state machine represents a grammar which is called "Regular Grammar". The "Regular Grammar" generates sentences all of which belong to "Regular Language". The sentences are sometimes used to denote words or tokens in different contexts.

We saw that the FSMs are incapable of expressing the transition probabilities and therefore the FSM model is extended to allow the inclusion of transition probabilities into the structure with additional labelling on arcs. These additional labelling on arcs denote transition probabilities. The extended model is now termed as the probabilistic finite state machine (PFSM). The PFSMs also define a probability distribution on the sentences generated along with defining a set of possible sentences. We also discussed the equivalence of two finite state machines and two probabilistic finite state machines. Two FSMs are considered equivalent if they generate or accept same language whereas this is not true for the two PFSMs.

The research work is focussed on the PFSM design model for the applications. Model construction is one aspect and the other aspect is model inference. Model inference helps in finding a better model when the source has become extinct. "Minimum Message Length (MML)" has emerged as a powerful tool in providing a method for inductive inference of structures like PFSMs by the way of encoding the PFSMs. In Chapter 4 we will discuss the MML coding method for PFSMs and the inference method to get an optimal PFSM.

Chapter 3

Minimum Message Length (MML)

3.1 Introduction

We discuss a few key concepts related to Bayes Theorem, Bayesian Inference, Huffman Coding and the Minimum Message Length (MML) principle. The MML calculation for a discrete binomial distribution and a discrete multinomial distribution are then discussed. We refer to these calculations in the code length calculation of the PFSM model later in Chapter 5.

On a very brief note here, MML works on the Bayesian principles of calculating the posterior probability of a hypothesis given the data and an elementary information theory concept based on Huffman code, converts the probability values into the form of code length in bits (Wallace, 2005).

3.2 Bayes' Theorem

In probability theory and statistics, Bayes's theorem is a result of conditional probability, stating that for two events A and B, the conditional probability of A given B is the conditional probability of B given A scaled by the relative probability of A compared to B (Rice, 2014).

Bayes's theorem is stated mathematically as the following equation:

$$Pr(A|B) = \frac{Pr(A \ \& \ B)}{Pr(B)} = \frac{Pr(B|A)Pr(A)}{Pr(B)} \tag{3.1}$$

where A and B are events and $Pr(B) \neq 0$

- $Pr(A)$ and $Pr(B)$ are the probabilities of observing events A and B without regard to each other
- $Pr(A \ \& \ B)$ is the probability of observing both events A and B
- $Pr(A|B)$, a conditional probability, is the probability of event A knowing that event B is true
- $Pr(B|A)$ is the probability of observing event B given that A is true

Figure 3.1 shows the illustration in the form of a Venn diagram for Bayes's theorem. The conditional probability of A given B ($Pr(A|B)$) in the figure is known by the size of $Pr(A \ \& \ B)$ relative to the size of $Pr(B)$.

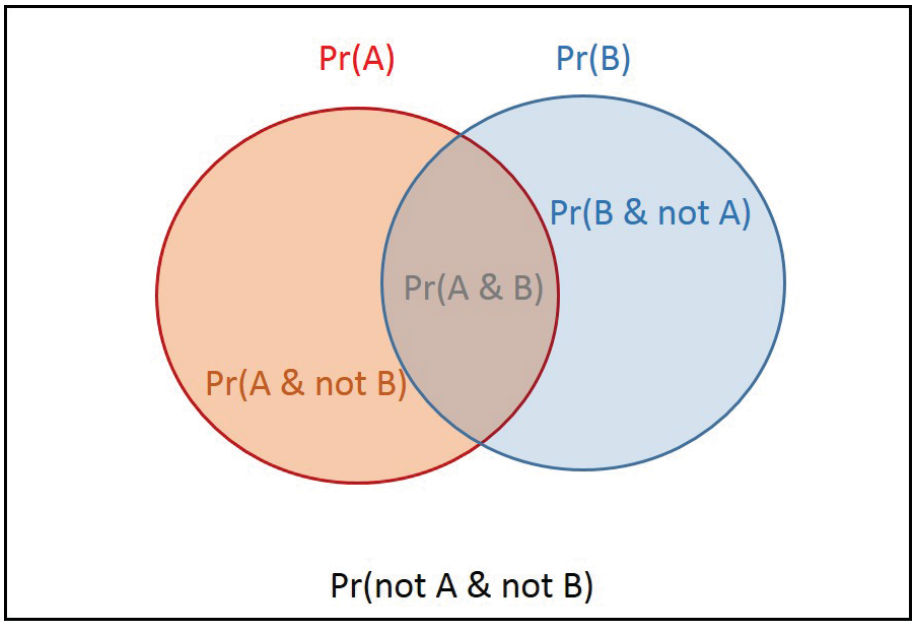


Figure 3.1: Venn Diagram illustrating Bayes's theorem

3.2.1 An example

Suppose we want to know the probability of a person having cancer at 65 years of age. Now the two events can be assumed as:

A: Person having cancer

B: Person being 65 years of age

So we are interested in finding the conditional probability of a person having cancer at 65 years of age given by $PrA|B)$. $Pr(A)$ is the probability of cancer and based on general prevalence of cancer, $Pr(A)$ can be assigned some value and, say, it is 1%. This is also known as the prior probability of cancer. If we assume that cancer and age are related, this new piece of information can be used to better assess that person's risk of having cancer. In order to apply knowledge of that person's age in conjunction with Bayes's Theorem, two additional pieces of information are needed. The needed information is as follows:

- The probability of being 65 years old. Suppose it is 0.2%.
- The probability that a person *with cancer* whose age is 65 years is 0.5%.

Knowing this along with the prior, we can calculate that the person whose age is 65 has a probability of having cancer equal to $(0.5\% \times 1\%) \div 0.2\% = 2.5\%$.

3.2.2 Bayesian Interpretation

The interpretation of Bayes's theorem depends on the interpretation of the probabilities used in the Bayes's theorem. In the Bayesian interpretation, probability is the measure of degree of belief. The Bayes's theorem relates the degree of belief in a hypothesis before and after accounting for evidence. This can be understood from a simple example of a coin flipped number of times. It is believed with an initial certainty of 50% that an unbiased coin will produce equal number of heads and tails. This is the prior belief in the hypothesis. When the coin is flipped certain number of times, the outcomes are observed and depending upon the outcomes of

coin flip, the initial belief gets updated by the Bayes's theorem. That is, the initial belief may rise, fall or remain unchanged after the results are seen.

For hypothesis H and evidence E,

- $Pr(H)$ is the initial belief in the hypothesis H, also known as prior probability of the hypothesis H.
- $Pr(E|H)$ is the likelihood of the evidence E knowing that the hypothesis is true.
- $Pr(H|E)$ is the updated belief in the hypothesis H after having accounted for the evidence E. This is also known as posterior probability of the hypothesis H.
- $Pr(E)$ is the probability of the evidence E and also a normalizing factor when different hypotheses are compared.

The Bayes's theorem is re-written according to the Bayesian interpretation in the following Equation 3.2.

$$Pr(H|E) = \frac{Pr(E|H)Pr(H)}{Pr(E)} \tag{3.2}$$

3.3 Bayesian Inference

Bayesian statistical inference allows us to find out the best hypothesis that is the source of generation of data or the evidence. This is done by the incorporation of probabilistic knowledge about the source of data independent of, or prior to, the observed data. This is where the Bayesian inference differs from the non-Bayesian inference. In non-Bayesian inference, the prior knowledge about the source of data is not considered or ignored and the different hypotheses are compared based on the likelihood of the different sources or models. The inclusion of a Bayesian prior can

allow deduction of rather more meaningful conclusions about the source (Wallace, 2005, sec. 1.13).

The inference problem is stated as follows:

There is a set of models Θ and we assume that the set is discrete. The models in the set Θ are labelled as $\{\theta_1, \theta_2, ...\}$. The evidence is also known as data D. The Bayesian approach assumes that, even before data D is known or seen, the competing models in the set Θ have initial probabilities, denoted as $Pr(\theta_i), (\theta_i \in \Theta)$. This is called the *prior probability* of the model or *prior distribution* or simply *prior*. In Bayesian approach, the data probability distribution or *likelihood* is also known for each model and this is denoted as $Pr(D|\theta_i)$. Then using the Bayes's theorem in Equation 3.2, the *posterior distribution* or *posterior probability* or simply *posterior* of the model θ_i is known by the Equation 3.3.

$$Pr(\theta_i|D) = \frac{Pr(D|\theta_i)Pr(\theta_i)}{Pr(D)} \quad \forall\theta_i \in \Theta \tag{3.3}$$

$Pr(D)$ is the marginal data probability given by

$$Pr(D) = \sum_{\theta_i} Pr(D|\theta_i)Pr(\theta_i) \tag{3.4}$$

$Pr(D)$ behaves as a normalizing constant for the posterior distribution and thus can be ignored while comparing the posterior probabilities of the different models in the discrete set Θ. If two sets of data $D1$ and $D2$ are obtained from the same source θ_i, then

$$Pr(\theta_i|D1, D2) = \frac{Pr(D1|\theta_i)Pr(D2|\theta_i)Pr(\theta_i)}{\sum_j Pr(D1|\theta_j)Pr(D2|\theta_j)Pr(\theta_j)} \tag{3.5}$$

The data sets are considered independent, that is, for a particular model θ_i, the probability of yielding $D1$ is not affected by knowing that it has also yielded $D2$.

Thus knowing the prior probability distribution over a set of discrete models, it is possible to make judgement on the most probable source based on the Bayesian method of inference. The prior on a set of possible models is given by the generalized function "$h(\theta)$" and $h(\theta)$ may represent a probability if θ is discrete.

3.4 Coding Probabilities

We start this section with some definitions that have relevance to computation of coding probabilities.

Definition 1: An *Information* is defined as something the receipt of which decreases our uncertainty about the state of the world (Wallace, 2005, sec. 2.1).

Definition 2: A *Message* is a binary sequence conveying some information (Wallace, 2005, sec. 2.1.1).

Definition 3: A *Code* is an agreement between the transmitter and the receiver as to how to represent the message in binary form. Clearly, code can be thought of as a kind of language (Wallace, 2005, sec. 2.1.1).

So, a complete message will convey several pieces of information one after another and, each piece is encoded separately.

Definition 4: When the messages are encoded piece-by-piece, each subsequence is known as a *word* and the code definition requires that the binary sequence for each word be defined (Wallace, 2005, sec. 2.1.1).

Definition 5: If codes are to be constructed for a known finite set of words, then there should be one-to-one mapping between the words and the codes. In addition, the codes should have a *prefix property*. This means, no code is a prefix of any other code and the code following this property is known as the *prefix code*.

3.4.1 Prefix Code Tree

Any binary prefix code for a set of N words can be represented by a *code tree*. Consider a set of five words $\{P\ Q\ R\ S\ T\}$. A possible prefix code for the set of five words is:

$P = 0, \quad Q = 100, \quad R = 101, \quad S = 110, \quad T = 111$

The code can be represented as a code tree in Figure 3.2.

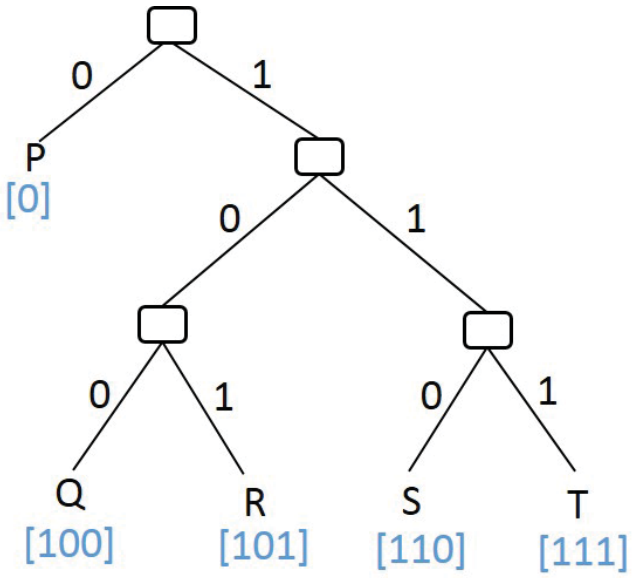


Figure 3.2: Tree for prefix code

The tree is read in the following manner to find a binary string corresponding to a word. The path from the root to the node labelled by the desired word is followed. Whenever the left branch of the tree is followed, it is noted down as "0" and the right path if followed, is noted down as "1". For example, the code for word S is generated by beginning from the root of the tree first. From the root, the right path is followed and "1" gets noted down. From the sub-tree rooted at the right of the root, another right path is followed and noted down as "1". From that node, a left branch is taken that eventually leads to the leaf S of the tree. Since the left branch is taken, it is noted down as "0". Therefore the code corresponding to word "S" is 110.

Now, the code tree in Figure 3.2 assigns the smallest string length code for word P. *String length* is the number of binary symbols used in coding of any word. So,

for word P, the string length is 1 and for the other words in the set, the string length is 3. The transmitter and the receiver first agree on the choice of a code tree first before transmitting the information. However when they agree on the coding scheme, they do not yet know the information to be sent. However, there can be a basis to believe that certain words are more probable than others in the set. For more probable words, the code string with less binary symbols can be chosen to minimize the use of communication medium, which can be expensive.

Let us assume that the transmitter and the receiver agree on the coding scheme as shown in Figure 3.2. Each word in this tree is transmitted with a certain probability and it is calculated in the following manner: First each node in the tree is assigned a level. *Level* of a node is the number of branches from root to that node. The root of the tree is at level 0 and as we go down in the tree, the level of nodes increases. Each node in the tree is then assigned a weight and *weight* of a node is calculated as 2^{-l}, where l is the level of the node. The weight of the root node is 1. The weights and roots are shown in Figure 3.3.

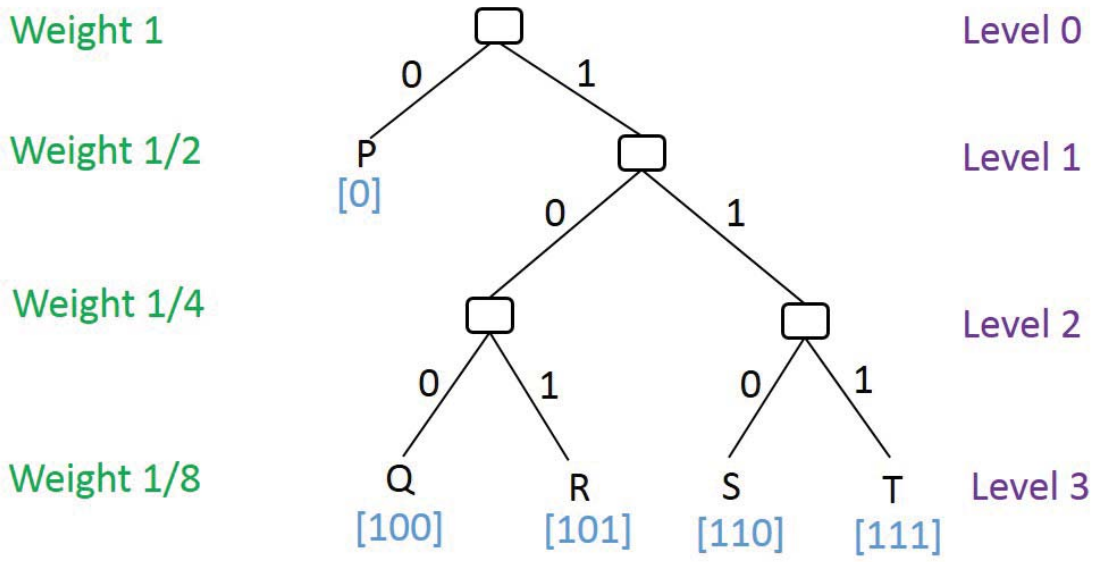


Figure 3.3: Tree for prefix code with weights and levels

The weights of the leaf nodes that represent the words in the set are [1/2, 1/8, 1/8, 1/8, 1/8] for the corresponding words $\{P,\ Q,\ R,\ S,\ T\}$ in the set. If a_i represents a particular word in the set, where $1 \leq i \leq N$ and N is the number

of words in the set, then $\sum_i 2^{-l_i} = 1$. l_i is the level of the i^{th} word in the set. The weight of any leaf node in the tree is interpreted as the probability with which the word is transmitted and the probabilities of all leaf nodes sum to 1. That is $\sum_i p_i = 1$, where p_i is the probability of the i^{th} word in the set of N elements.

Alternatively, if the probabilities with which the words are transmitted are known prior to sending the information from transmitter to receiver, the string length for each message or word can be calculated. Taking the same example of the code tree in Figure 3.3, the probability with which the word P is transmitted is $\frac{1}{2}$. The string length of the code corresponding to word P is calculated as $-\log_2 \frac{1}{2}$, which is equal to 1. The string length of the code corresponding to word Q is calculated as $-\log_2 \frac{1}{8}$, which is equal to 3. Similarly all other words in the set result in a string length of 3.

In general, the number of bits required to encode a word with transmission probability p_i, is given by $-\log_2 p_i$. Let us denote the number of bits as string length L_i for the word i in the set and the general equation to calculate the string length or code length is given by Equation 3.6.

$$L_i < (-\log_2 p_i) + 1 \quad or \quad L_i \approx -\log_2 p_i \tag{3.6}$$

An important point to discuss here is, if the logarithm of the probability is calculated with base 2, this results in the code length being reported in *bits*. Or, if the natural logarithm (ln) of the probability is calculated, the code length is reported in *nits*. The string length of the message in coded form is also sometimes referred as *code length* or *message length*. Another observation from the string length calculation is, messages with high probabilities result in less string length. Like in the example above, the word P is the most probable message and the string length for the same word is the least.

3.5 Minimum Message Length (MML) principle

The MML methodology has its roots in classical Bayesianism. For a discrete set of models Θ, where the models in the set are labelled as $\{\theta_1, \theta_2, ...\}$, the posterior probability of a selected model $\hat{\theta}$ from the set Θ, given according to Bayes's theorem, is: $Pr(\hat{\theta}|D) = \frac{h(\hat{\theta})Pr(D|\hat{\theta})}{Pr(D)}$. Here $h(\hat{\theta})$ is the prior probability of the selected model $\hat{\theta}$ and $Pr(D|\hat{\theta})$ is the likelihood of the data D generated assuming the theory $\hat{\theta}$ to be true. The selected model $\hat{\theta}$ is the theory that results in the maximum posterior probability compared to other models in the set Θ.

Choosing a theory or a model with maximum posterior probability is equivalent to choosing a theory with highest product of the *prior* and the *likelihood* function, as $Pr(D)$ is only a normalizing constant when the different theories are compared and thus can be disregarded in the calculation of posterior probability of a theory.

The theory $\hat{\theta}$ with prior probability $h(\hat{\theta})$ can give an explanation length of $-\log_2 h(\hat{\theta})$ bits according to the coding scheme discussed in Section 3.4.1 and explanation length of data D generated using $\hat{\theta}$ is given by $-\log_2 Pr(D|\hat{\theta})$. Therefore the explanation length for the posterior probability of the theory is given by Equation 3.7.

$$-\log_2 h(\hat{\theta}) \ -\log_2 Pr(D|\hat{\theta}) \tag{3.7}$$

Here $-\log_2 h(\hat{\theta})$ is the first part explanation length or code length of the theory or model $\hat{\theta}$ and $-\log_2 Pr(D|\hat{\theta})$ is the second part code length that encodes data D using the model $\hat{\theta}$. As discussed in Section 3.4.1 that highly probable events result in small code lengths, therefore maximizing the posterior probability is equivalent to minimizing $-\log_2 h(\hat{\theta}) \ -\log_2 Pr(D|\hat{\theta})$, i.e., the length of a two-part message conveying the theory and the data in light of the theory.

There are essential differences between MML and simple Bayesian induction. MML assists in the construction of prior probability distributions by establishing a

correspondence between the code used to describe a theory with the prior probability of that theory (Wallace, 2005, sec. 2.3). Classical statistics typically postulates a distribution with an unknown parameter, θ, as a single model, and then attempts to infer such a (vector) θ. Classical statistics is interested largely in point estimation, but can also be interested in interval estimation and giving confidence intervals. The Bayesian approach takes a Bayesian prior distribution, and can then proceed in a variety of different ways. One approach is simply to arrive at a posterior distribution. Another Bayesian approach is to take the posterior distribution and to obtain a point estimate by minimising an expected loss function - see, e.g., Lehmann and Casella (2006); Dowe et al. (1998). Other Bayesian point estimators exist, such as (e.g.) Minimum Message Length (MML) and various approximations (see, e.g., Wallace (2005); Dowe (2008a, 2011), and also the Bayesian maximum a posteriori (MAP) approach.

There are also differences between MML and the more recent MDL (Minimum Description Length). The MDL method is attributed to Jorma Rissanen (1978) (Rissanen, 1978, 1999) and its objective is to find the best model class and the best model in it as found by the shortest code length. The MML method on the other hand is attributed to Chris Wallace and David Boulton (1968) (Wallace and Boulton, 1968) and is concerned with finding the model that is the best explanation of the data. An even more important distinction between MML and MDL is the use of subjective priors (Raman et al., 1998; Wallace and Dowe, 1999b; Baxter and Oliver, 1994). The MML considers and believes in incorporating the prior knowledge to the best possible extent, whereas MDL considers a uniform or universal prior thus ignoring the prior information. For discussions of the generality of MML, see (Wallace and Dowe, 1999a; Wallace, 2005). For applications of MML to clustering and mixture modelling, see, e.g., (Wallace and Boulton, 1968; Wallace, 1990; Wallace and Dowe, 1994b, 2000; Visser and Dowe, 2007) and for pointers to applications of MML to a variety of other areas, see, e.g., (Wallace, 2005; Dowe, 2008a, 2011).

3.5.1 More on MML and Applications

The Minimum Message Length (MML) principle dates back to Wallace and Boulton (1968), with a variety of papers and two theses following upon this 1968 paper in the years up until 1975 Boulton and Wallace (1969, 1970); Boulton (1970); Boulton and Wallace (1973b,c,a, 1975); Wallace and Boulton (1975); Boulton (1975). This extensive body of work from 1968 to 1975 would be followed a few years later by the Minimum Description Length (MDL) principle Rissanen (1978). For further comparison between MML and the subsequent MDL principle, see, e.g., Rissanen (1998); Wallace and Freeman (1987); Rissanen (1999); Wallace and Dowe (1999a,b,c)(Wallace, 2005, sec. 10.2)(Dowe, 2011, sec. 6.7).

The original application of MML in Wallace and Boulton (1968) was to clustering and mixture modelling (with multinomial and Gaussian distributions and total assignment - all data items or things each go into one class), with subsequent work Wallace (1986, 1990); Wallace and Dowe (2000, 1994b) staying with multinomial and Gaussian distributions but extending this from total assignment to partial assignment (data things can be spread partially over different classes, although a "coding trick" Wallace (1986) proposes doing total assignment in a clever pseudo-random way). Work was then done on the von Mises circular distribution (for angular data) Wallace and Dowe (1993, 1994a) and the von Mises-Fisher spherical distribution Dowe et al. (1996), with the Poisson and von Mises circular distributions included into the "*Snob*" program[1] for mixture modelling Wallace and Dowe (1994b, 1997, 2000). Other statistical distributions are also considered in Agusta and Dowe (2002, 2003a,b). MML single latent factor analysis Wallace and Freeman (1992) has also been used to extend the modelling Edwards and Dowe (1998), as has sequential modelling Edgoose and Allison (1999); Edgoose et al. (1998); Molloy et al. (2006) and models of spatial correlation Wallace (1998); Visser and Dowe (2007), and there has also been subsequent work in the relevant areas Figueiredo and Jain (2002);

[1] see Dowe (2008a) for a discussion of the name of this program

Kasarapu and Allison (2015). Other work on MML includes decision trees (or classification trees) Wallace and Patrick (1993) and (an extension to these allowing disjunctions, or ORs, or joins, called) decision graphs Oliver and Wallace (1991); Oliver (1993); Oliver et al. (1992); Tan and Dowe (2002, 2003)(Wallace, 2005, chap. 7) and oblique decision trees Tan and Dowe (2004).

Work on MML Bayesian nets Wallace et al. (1996); Wallace (2005) and abovementioned MML decision trees has been unified in Comley and Dowe (2003, 2005), and MML Bayesian nets with hidden latent variables are considered in Visser et al. (2012). Other generative models are given in Torsello and Dowe (2008a,b), where they are used for structural representation. Some of the other many works of interest include abovementioned sequential Edgoose and Allison (1999); Edgoose et al. (1998); Molloy et al. (2006) and spatial models Wallace (1998); Visser and Dowe (2007), work on time series Fitzgibbon et al. (2004); Schmidt (2008) and econometric panel data modelling and estimation (Dowe, 2011, sec. 6.5). Some work on MML hypothesis testing includes (Dowe, 2008a, sec. 0.2.5, p539 and sec. 0.2.2, p528, col. 1 and sec. 1) (Dowe, 2008b, p433 (Abstract), p435, p445 and pp455-456) Musgrave and Dowe (2010)(Dowe, 2011, sec. 3.2, p919 and sec. 7.6, p964) Makalic and Schmidt (2011). Work on MML probabilistic finite state automata Georgeff and Wallace (1984); Edgoose and Allison (1999); Edgoose et al. (1998); Wallace (2005) is surveyed elsewhere throughout this thesis. For similar notes to much of the above, see Chmait (2017) - and for further reference and survey material on MML, see Wallace (2005); Dowe (2011). The influence of MML on quantifying intelligence and intelligence testing is outlined in (e.g.) (Dowe, 2013, sec. 4) and references therein, including Dowe and Hajek (1997a,b, 1998). For similar notes to much of the above, see Chmait (2017) - and for further reference and survey material on MML, see Wallace (2005); Dowe (2011).

3.6 Code length calculation for a Binomial distribution case

In this section, we discuss the MML calculation or MML estimate for a binomial distribution case. A binomial distribution is defined as a frequency distribution of a possible number of successful outcomes in a given number of trials in each of which there is a probability of success. Let us assume, for a binomial example, the number of trials is N, the number of successes is s and p is the unknown probability of success. The likelihood of s successes denoted as $f(s|p)$, under p probability of success, is given by:

$$f(s|p) = \binom{N}{s} p^s (1-p)^{N-s}$$

The code length L corresponding to the above likelihood function is $-\log f(s|p)$. Therefore,

$$\begin{aligned}
L &= -\log\left[\binom{N}{s} p^s (1-p)^{N-s}\right] \\
&= -\log\binom{N}{s} - \log(p^s) - \log(1-p)^{N-s} \\
\frac{\partial L}{\partial p} &= \frac{\partial}{\partial p}\left[-\log\binom{N}{s} - \log(p^s) - \log(1-p)^{N-s}\right] \\
\frac{\partial L}{\partial p} &= 0 - \frac{\partial}{\partial p} s\log p - \frac{\partial}{\partial p}(N-s)\log(1-p) \\
\frac{\partial L}{\partial p} &= \frac{-s}{p} + \frac{(N-s)}{(1-p)} \qquad (3.8) \\
\frac{\partial^2 L}{\partial p^2} &= \frac{\partial}{\partial p}\left[\frac{-s}{p} + \frac{(N-s)}{(1-p)}\right] = \frac{s}{p^2} + \frac{(N-s)}{(1-p)^2} \qquad (3.9)
\end{aligned}$$

The expected value of s, denoted as $E(s)$, is Np. Therefore the expected value of the second order derivative of L with respect to p is given by:

$$E\left[\frac{\partial^2 L}{\partial p^2}\right] = \frac{Np}{p^2} + \frac{(N - Np)}{(1-p)^2} = \frac{N}{p} + \frac{N(1-p)}{(1-p)^2} = \frac{N}{p(1-p)} \tag{3.10}$$

The expected value of the second order derivative of L with respect to p is the *Fisher Information* represented as $F(p)$. It is a function of p whose form depends only on the form of model probability function $f(s|p)$. Being the expected value of minus the second differential of the log likelihood, it indicates how sharply peaked we expect the log likelihood to be, as a function of p (Wallace, 2005, sec. 5.1). If $F(p)$ is large for some value of p, the negative log likelihood will have sharp and narrow peak and thus the estimate of p is quoted more precisely.

As there is no information regarding the prior probability $h(p)$, it is reasonable to assume a flat prior $h(p) = 1$. The two-part code length using the simple MML formula I1B (Wallace, 2005, sec. 5.1) is computed as below:

$$= -\log h(p) + (1/2)\log F(p) - \log f(s|p) + (K/2)\log k_D + (K/2) - \log \binom{N}{s}$$

Here K is the number of parameters and k_D is a lattice constant. The value of K for a binomial example is 1 and the value of lattice constant for one parameter is 1/12. The above equation is simplified and the MML estimate for p denoted as $\hat{p}_{MML}$ is given by Equation 3.11.

$$\hat{p}_{MML} = \frac{s + \frac{1}{2}}{(N+1)} \tag{3.11}$$

The *maximum likelihood* (*ML*) estimate of p can also be found by equating Equation 3.9 to zero.

$$\frac{\partial L}{\partial p} = \frac{-s}{p} + \frac{(N-s)}{(1-p)} = 0$$
$$\rightarrow \frac{(N-s)}{(1-p)} = \frac{s}{p}$$
$$\rightarrow p = \frac{s}{N}$$

The ML estimate for p denoted as $\hat{p}_{ML}$ is given by Equation 3.12.

$$\hat{p}_{ML} = \frac{s}{N} \tag{3.12}$$

3.7 Code length calculation for a Multinomial distribution case

The code length calculation for a binomial distribution case generalizes directly to a multinomial distribution case with m possible outcomes for each of the N trials. Let us denote the number of outcomes of each type as $s_1, s_2, ..., s_m$ and their respective probabilities as $p_1, p_2, ..., p_m$. The likelihood of $s_1, s_2, ..., s_m$ outcomes under p probability is given by:

$$f(s_1, s_2, ..., s_m|p) = \frac{N!}{s_1!s_2!...s_m!} p_1^{s_1} \times p_2^{s_2} \times ... \times p_m^{s_m}$$

The code length L being negative log of the likelihood function, is given by:

$$L = -\log f(s_1, s_2, ..., s_m|p) = -\log \left[\frac{N!}{s_1!s_2!...s_m!} p_1^{s_1} \times p_2^{s_2} \times ... \times p_m^{s_m} \right]$$
$$= -\log \binom{N}{s_1, s_2, ..., s_m} - \sum_{i=1}^{m} s_i \log p_i$$

The *Fisher Information* represented as $F(p_1, p_2, ..., p_m)$ is the expected value of the second order derivative of L with respect to p and is given by:

$$F(p_1, p_2, ..., p_m) = \frac{N^{m-1}}{\Pi_{i=1}^{m} p_i}$$

The MML estimate for p_i, where p_i is the probability of a particular outcome i and $1 \leq i \leq m$, assuming a uniform prior is given by:

$$\hat{p}_{iMML} = \frac{s_i + (1/2)}{N + (m/2)} \tag{3.13}$$

The ML estimate of the probability of a particular outcome i is given by:

$$\hat{p}_{iML} = \frac{s_i}{N} \tag{3.14}$$

3.8 Summary

In this chapter, we discussed the formal key concepts related to Bayes's Theorem, Bayesian Inference, Coding probabilities using Huffman coding and most important, the Minimum Message Length (MML) principle.

The importance of Bayes's theorem lies in the fact that, it makes use of prior knowledge in the computation of posterior probability of a hypothesis. The belief in a particular theory is updated based on the likelihood of the hypothesis and the prior information. Bayesian inference helps in selecting a model when there are more sources generating data and the best source has to be inferred. According to Bayesian inference, the model with the highest posterior probability is regarded as the best source generating the data. The way of converting the probabilities associated with any event into codes is through the use of prefix codes, the Huffman

Coding. The information theory concept based on Huffman code states that an event occurring with probability Pr can be coded in -$\log_2 Pr$ bits. Highly probable events result in small code lengths.

The Minimum Message Length (MML) principle, being inherently Bayesian, combines the above two concepts as one. That is, it works on the Bayesian principles of calculating the posterior probability of a hypothesis and then converts the posterior probability in the form of code length by the use of Huffman codes. Therefore the code length of the posterior probability of a hypothesis is the sum of the code lengths of the likelihood function and the prior probability of the hypothesis. The two-part code length is the code length of the hypothesis given by $-\log_2 h(\theta)$ and the code length of the data generated in light of the hypothesis given by $-\log_2 Pr(D|\theta)$. The MML inference mechanism selects that model, from the set of possible models, that gives the least two-part code length.

The understanding of the Minimum Message Length (MML) principle is applied to the case of discrete probability distributions seen in binomial and multinomial distributions. The multinomial distribution case is a generalization of binomial distribution case. So generally speaking, in a multinomial distribution case, there are a certain number of possible outcomes on a sequence of N trials. If the number of outcomes is 2, it becomes a binomial distribution case. In both the cases, the negative log of the likelihood function is first represented in the form of code length. The *Fisher Information* is then computed by the expected value of the second order derivative of the code length of the likelihood function. The information is then used in the two-part code length calculation using the simple MML formula I1B (Wallace, 2005, sec. 5.1). The MML estimate for the probability of a particular outcome is then known. We also mentioned the *Maximum Likelihood* (ML) estimate for the probability of a particular outcome in both the distribution cases.

Chapter 4

MML Encoding and Inference of PFSMs

4.1 Introduction

Finite State Machines (FSMs) can be effective representations for a data of sequential nature or data observed in time series. Some sources generating such data are natural language corpus, email corpus and DNA sequences. What is commonly observed in these data sources is presence of some patterns or regularities, for which FSMs are a suitable form of representation. The data can be viewed as a set representing a collection of words or tokens separated by some delimiter symbols. However powerful FSMs might seem, they are incapable of expressing regularities beyond those fully captured by the formal grammar (FSM) (Wallace, 2005, sec. 7.1). That is, they lack the ability to express probabilistic transitions among the states. For this reason, as discussed in Chapter 2, the FSM structure is extended to allow inclusion of probabilistic structure and the extended model is called a probabilistic finite state machine (PFSM). A PFSM defines not just a set of possible words but also defines a probability distribution on the set.

MML has emerged as a powerful tool, not only in providing an encoding mechanism to find the code length of a PFSM but, it also plays an important role in the

inductive inference of discrete structures such as PFSMs. This chapter discusses how an inductive hypothesis (PFSM) can be modelled from a finite set of sentences drawn from a finite alphabet set. An inductive hypothesis is said to be an abstraction over a set of sentences, but the inference of hypothesis tells how probable the model is in generating those sentences. So, we also discuss the method of inference using MML in this chapter.

4.2 Modelling an Inductive Hypothesis

Consider a set representing a collection of sentences. A sentence in turn, is a collection of data items drawn from a finite alphabet set. To make the distinction between sentences and data items clear, we again consider the same example considered in Section 2.2.2, where a set with sentences $\{CAB, CAAAB, BBAAB, CAAB, BBAB, BBB, CB\}$, is taken. In the set, the sequences separated by ",", are the sentences and each sentence is a combination of symbols considered from the alphabet set $\{A, B, C\}$.

Now, an abstraction over the set of these data items can be modelled using an inductive hypothesis. The abstraction, in general, consists of two parts (Wallace and Georgeff, 1983; Georgeff and Wallace, 1984; Raman et al., 1998; Collins and Oliver, 1997)

- A statement of the hypothesis H itself.
- A specification of data D given this hypothesis H.

A good hypothesis in general will minimize the second component of abstraction (Raman et al., 1998). One of the ways in which the hypothesis can be stated is as list of probabilities of occurrence of each data item in the set. Codes can then be assigned to these data items, by way of assigning the shortest code to the most probable data item and longest code to the least probable data item. If the symbols representing the data items are encoded using an alphabet with n unique symbols,

then the information theory guidelines suggest that an optimal code length for the symbol representing the i^{th} data item in the set is calculated as $-\log_n p_i$, where p_i is the probability of occurrence of data item i from the alphabet set (Shannon, 1948). By the method of Huffman coding (Huffman, 1952; Gallager, 1968), this is achieved in practice with the construction of prefix codes over the binary alphabet. Thus, if data item i occurs with probability p_i, then the prefix code corresponding to the data item i is $-\log_2 p_i$, according to Huffman coding.

Another method of stating the hypothesis is the abstraction specified as a Probabilistic Finite State Machine (PFSM). The PFSM as discussed in Section 2.3 is a deterministic state machine modified to have a stochastic transition function. That is, the movement from one state to another is governed by probabilities. In the next section, we discuss the MML method of encoding a hypothesis specified as a PFSM. This was developed by (Wallace and Georgeff, 1983), who looked at the general problem of inferring a structure for PFSM from a given set of sentences with zero or more embedded delimiters.

4.3 Modelling a PFSM

Abstraction in the form of a PFSM is modelled using Minimum Message Length (MML). As a recap here, MML principle has its roots in the classical Bayesian theory of calculating the posterior probability of a hypothesis H. Computing the posterior probability equivalently means computing the product of the prior probability of the hypothesis and the probability of the data D generated in light of the hypothesis (Oliver and Hand, 1994; Cheeseman, 1990). This equivalently means to compute the sum of the following code lengths:

- code length of the hypothesis H.
- code length of the data D encoded using this hypothesis.

The problem of modelling an inductive hypothesis from a given set of observations becomes one of choosing between the competing models. Georgeff and Wallace

(1984) proposes the MML principle to help make the decision. The hypothesis that minimizes the sum of the above code lengths is regarded as the best one. Quantitatively, the sum can be put down as the following formula that calculates the two-part code length of a hypothesis in bits.

$$\begin{aligned} CodeLength(H|D) &= CodeLength(H) + CodeLength(D|H) \\ &= -\log_2 Pr(H) - \log_2 Pr(D|H) \end{aligned} \tag{4.1}$$

4.3.1 Assertion code for hypothesis H

If the hypothesis is stated in the form of a PFSM, then the number of bits derived to encode it can be calculated. Let us consider the PFSM of Figure 2.3 in Chapter 2, which is the Prefix Tree Acceptor (PTA) of the Gaines (1976) data. Let us now call the collection as the observed data D. So, $D = \{CAB\#CAAAB\#BBAAB\#CAAB\#BBAB\#BBB\#CB\#\}$. The definitions used in the context of finding the code length of a PFSM are enumerated below.

1. S is the number of states in the FSM.
2. Σ is the input alphabet set.
3. V is the cardinality of the input alphabet set.
4. n_{ik} is the number of transitions from state i on symbol k, where $k \in \sum$.
5. M is the total number of arcs from all the states.
6. a_i is the number of arcs leaving the current state i

Note that $\sum_{i=1}^{S} a_i = M$.

The distinction between arcs and transitions is as follows. An arc represents a single transition from one state to another state on seeing a particular symbol from the finite alphabet set Σ. Whereas, transition represents the frequency of occurrence

of an arc on a particular symbol. The calculation of code length for the hypothesis H assumes any arc in the PFSM to have transited at least once for a uniform prior assumption of the hypothesis. The PFSM of Figure 2.3 is shown again in figure below and we replace the probability values labelling the arcs of Figure 2.3 with the frequency of occurrence of symbols on the arcs. The figure is redrawn below with the changed labels on the arcs.

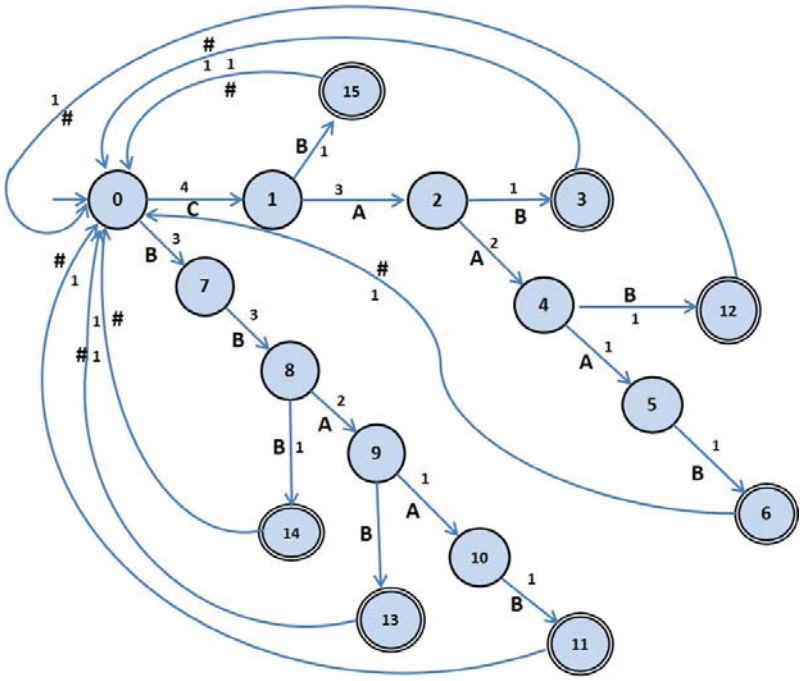


Figure 4.1: PFSM (PTA) of Gaines data D

The hypothesis H is encoded considering the number of states in the PFSM, number of arcs leaving each state of the PFSM, the labels on the arcs and the destinations. (Wallace, 2005, sec. 7.1.3) describes a better coding scheme and the coding scheme works as follows.

- The code begins by first stating the number of states in the PFSM. As the number of states is specified by S, it takes $\log_2 S$ number of bits to state the number of states in the PFSM. For the PFSM in Figure 4.1, the number of states is 16 and the cost to state this is $\log_2 16$ bits.

- Next the arcs leaving each state are stated. The number of possibilities for the number of arcs from each state is V, by considering a uniform distribution

of the symbols from each state. Therefore from each state, the arcs leaving each state are encoded in $\log_2 V$ bits. The number of arcs leaving each state is quantified as a_i, where i is a state and $0 \leq i \leq (S-1)$. The number of different possibilities for a_i is between 1 and V.

- The set of symbols leaving any state i is encoded next. The different number of ways of selecting a_i symbols from a set of V symbols is calculated as $\binom{V}{a_i}$. Therefore the number of bits required to state this set of symbols from state i is, $\log_2 \binom{V}{a_i}$.

- The PFSM specification includes the destination state reached when a symbol is input on a current state i. The different a_i symbols on the current state i makes the PFSM transit on a_i states. As the destination belongs to one of the states from 0 to $S-1$, the number of bits required to specify the destination states from current state i is $a_i \log_2 S$. But, for the arcs labelled #, the destination state is implied. Therefore the number of bits required in this case is $(a_i - 1)\log_2 S$.

Using the above coding scheme any PFSM can be stated but, the code above contains some inefficiencies. The numbering of the states other than *State* 0 can be done in any arbitrary manner stating $(S-1)!$ equal length code lengths for the same PFSM. Therefore, the redundancy is removed by obtaining a correction in the above code, by subtracting $(S-1)!$ from the calculated code length of the PFSM. The code is still redundant as it permits descriptions of PFSMs some of whose states cannot be reached from the starting state. This amount of redundancy, however, is small and can be thus ignored (Wallace, 2005, sec. 7.1.3).

For the PFSM of Figure 4.1, the asserted costs for the number of arcs, labels and destinations for each state are calculated and shown in Table 4.1.

Table 4.1: Code Length of PFSM from Figure 4.1 accepting D

State	a_i	**Cost**	**Label(s)**	**Cost**	**Dest.(s)**	**Cost**
0	2	$\log_2 V$	(C, B)	$\log_2 \binom{V}{2}$	(1,7)	$2\log_2 S$
1	2	$\log_2 V$	(A, B)	$\log_2 \binom{V}{2}$	(2,15)	$2\log_2 S$
2	2	$\log_2 V$	(B, A)	$\log_2 \binom{V}{2}$	(3,4)	$2\log_2 S$
3	1	$\log_2 V$	$(\#)$	$\log_2 \binom{V}{1}$	(0)	0
4	2	$\log_2 V$	(A, B)	$\log_2 \binom{V}{2}$	(5,12)	$2\log_2 S$
5	1	$\log_2 V$	(B)	$\log_2 \binom{V}{1}$	(6)	$\log_2 S$
6	1	$\log_2 V$	$(\#)$	$\log_2 \binom{V}{1}$	(0)	0
7	1	$\log_2 V$	(B)	$\log_2 \binom{V}{1}$	(8)	$\log_2 S$
8	2	$\log_2 V$	(A, B)	$\log_2 \binom{V}{2}$	(9,14)	$2\log_2 S$
9	2	$\log_2 V$	(A, B)	$\log_2 \binom{V}{2}$	(10,13)	$2\log_2 S$
10	1	$\log_2 V$	(B)	$\log_2 \binom{V}{1}$	(11)	$\log_2 S$
11	1	$\log_2 V$	$(\#)$	$\log_2 \binom{V}{1}$	(0)	0
12	1	$\log_2 V$	$(\#)$	$\log_2 \binom{V}{1}$	(0)	0
13	1	$\log_2 V$	$(\#)$	$\log_2 \binom{V}{1}$	(0)	0
14	1	$\log_2 V$	$(\#)$	$\log_2 \binom{V}{1}$	(0)	0
15	1	$\log_2 V$	$(\#)$	$\log_2 \binom{V}{1}$	(0)	0

The cost columns in the table are added and the cost of encoding the hypothesis H without applying correction is shown by Equation 4.2.

$$\begin{aligned} &\sum_{i=1}^{S} \log_2 \binom{V}{a_i} + \sum_{i=1}^{S} a_i \log_2 S + \sum_{i=1}^{S} \log_2 V + \log_2 S \\ &= \sum_{i=1}^{S} \log_2 \binom{V}{a_i} + M \log_2 S + S \log_2 V + \log_2 S \end{aligned} \tag{4.2}$$

After applying the correction to the above code by subtracting $(S-1)!$ from the equation above, the code length for the hypothesis H for the PFSM in Figure 2.1 is given by Equation 4.3.

$$CodeLength(H) = \sum_{i=1}^{S} \log_2 \binom{V}{a_i} + M \log_2 S + S \log_2 V + \log_2 S - \log_2(S-1)! \tag{4.3}$$

The formula in Equation 4.3 is used further in calculating the first part code length of any PFSM. The second part code length, that encodes the data D, assuming the hypothesis to be true is discussed in the section following this.

4.3.2 Assertion code for data D generated by hypothesis H

This part of the code length asserts the data D generated by the hypothesis H. From each state of the PFSM, there are multiple transitions on the arcs leaving the state, therefore the distribution from each state appears to be multinomial. If a uniform prior distribution from each state is assumed, then the MML estimate for probability of a particular outcome can be known by the use of formula 3.13 discussed in Chapter 3. But, this assumption is valid only when the sample size is known before transmitting the data. This is not usually the case, as after asserting the discrete structure of the PFSM, the assertion starts encoding the data as it comes. A possible solution to this is, if we desire to use the same MML probability calculation for a particular transition on a symbol from a state, the expected number of transitions from each state can be assumed. But in this case the computation of *Fisher Information* expressed as a function of the structure of the PFSM and transition probability distribution from all its states, would be computationally expensive. A much better solution to encode data is by the use of an adaptive code.

The adaptive code differs from the other code discussed in Chapter 3 in Section 3.7, in the assumption made about the receiver's prior knowledge. In this type of coding, the data are encoded in segments, each giving the next symbol to be seen in making a transition from the current state. The adaptive coding method is detailed in the section following next.

4.3.2.1 Adaptive Coding of data D with a uniform prior

To understand the method of adaptive encoding we consider a small example of a discrete Binomial distribution where a sequence of *heads* and *tails* are recorded in a series of coin tosses. Let us denote the outcome *head* as h and outcome *tail* as t. The sequence is as follows:

$$h\ h\ t\ h\ t\ h\ h\ t\ t\ t$$

The events in the sequence are independent of each other. The sequence starts with the outcome h. Before seeing this first h, if the two outcomes are considered equally probable or in other words both the outcomes are assumed to be present in equal numbers before the start of the actual data, then the probability of seeing the first symbol as h is $\frac{1}{2}$. The probability of seeing the second symbol as h is $\frac{2}{3}$, probability of seeing the third symbol as t is $\frac{1}{4}$ and so on. The probability of occurrence of each subsequent symbol, as it appears, is mentioned in Table 4.2. As the sequence appears, the probabilities are multiplied incrementally and when the whole data is seen, the final probability of seeing five number of *heads* and five number of *tails* is given by the last column of the table.

Table 4.2

		1		2		3		4		5		6		7		8		9		10	product
Data		h		h		t		h		t		h		h		t		t		t	
number of h	1	-	2	-	3	-	3	-	4	-	4	-	5	-	6	-	6	-	6	-	
number of t	1	-	1	-	1	-	2	-	2	-	3	-	3	-	3	-	4	-	5	-	
total number of h & t	2	-	3	-	4	-	5	-	6	-	7	-	8	-	9	-	10	-	11	-	
running estimate of probability		1/2		2/3		1/4		3/5		2/6		4/7		5/8		3/9		4/10		5/11	((no. of h)! * (no. of t)!)/ (no. of h + no. of t + 1)!

The idea of adaptive coding considered for a Binomial distribution case can be extended for the case of a discrete Multinomial distribution. For a Multinomial distribution case, let us say there are m possible outcomes for each of the N trials. The number of outcomes of each type are denoted as $s_1, s_2, ..., s_m$. The respective probabilities are denoted as $p_1, p_2, ..., p_m$. By incrementally multiplying the probability of the outcomes as we progress, the final probability expression is given by

Equation 4.4. In the expression, a uniform prior is assumed over all the possible outcomes before the outcomes are seen and the prior is $\frac{1}{m}$.

$$p_1 \times p_2 \times ... \times p_m = \frac{(s_1!s_2!...s_m!)(m-1)!}{(N+m-1)!} \tag{4.4}$$

Now we apply the method of adaptive coding to the states of PFSM where the states observe a multinomial distribution. There are a_i arcs transiting from a state i or we can say the number of possible outcomes from state i are a_i. Each outcome is labelled by a symbol k and $k \in \sum$. The code length needed to encode all the transitions from the current state i is given by $-\log_2 \frac{(a_i-1)!\Pi_k(n_{ik}!)}{(t_i+a_i-1)!}$, where n_{ik} is the number of times the state i has already seen symbol k transiting from it, t_i is the total number of times the state has already been left (Wallace, 2005, sec. 7.1.6). The code length of the individual states are summed over S and that gives the complete two-part code length for data D generated by the PFSM. The code length of the data D generated by the hypothesis H is given by the Equation 4.5.

If it is desired to treat the probabilities as an essential feature of the inferred model, a small correction from (Wallace, 2005, sec. 5.2.13) can be added to the above expression, as detailed in (Wallace, 2005, sec. 7.1.6), given by approximately $(\frac{1}{2}(\pi(a_i-1)) - 0.4)$ for each state. For the state having only one exit arc, there is no need to calculate the transitional probability.

$$CodeLength(D|H) = \sum_{i=1}^{S} \log_2 \frac{(t_i+a_i-1)!}{(a_i-1)!\Pi_k(n_{ik}!)} \tag{4.5}$$

The total two-part code length for the PFSM is written as CodeLength(H) + CodeLength($D|H$), which is approximately equal to the following expression in Equation 4.6.

Total Two-Part CodeLength =

$$\sum_{i=1}^{S}\left\{\log_2\frac{(t_i+a_i-1)!}{(a_i-1)!\Pi_k(n_{ik}!)}+\log_2\binom{V}{a_i}\right\}+M\log_2 S+S\log_2 V+\log_2 S-\log_2(S-1)! \tag{4.6}$$

4.3.3 A Few Examples

As the first example, we compute the two-part code length on the PFSMs of Gaines (Gaines, 1976) data. The sequence is represented as $D = \{CAB\#CAAAB\#BBAAB\#CAAB\#BBAB\#BBB\#CB\#\}$.

Figures 4.2 - 4.7 show the PFSMs explaining the Gains data.

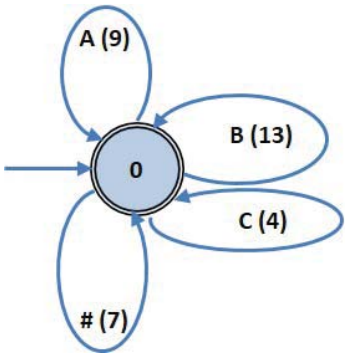


Figure 4.2: 1-state PFSM explaining Gaines data D

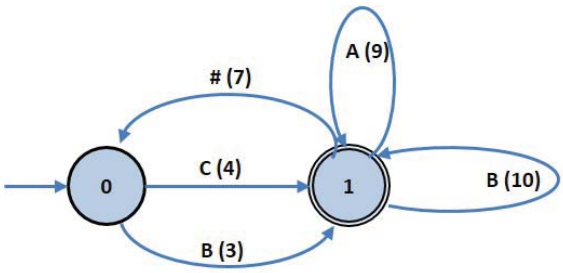


Figure 4.3: 2-state PFSM explaining Gaines data D

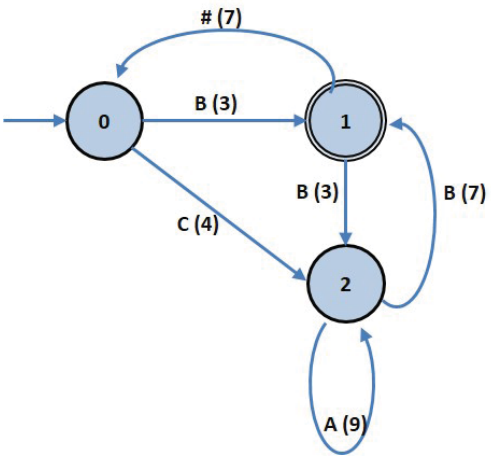


Figure 4.4: 3-state PFSM explaining Gaines data D

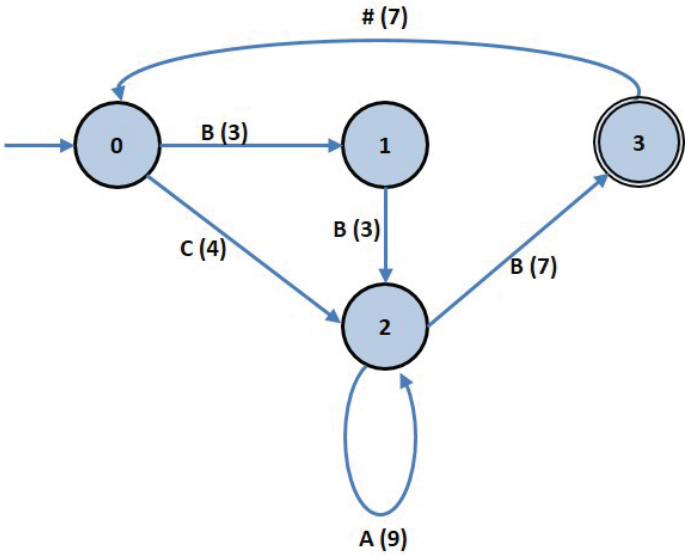


Figure 4.5: 4-state PFSM explaining Gaines data D

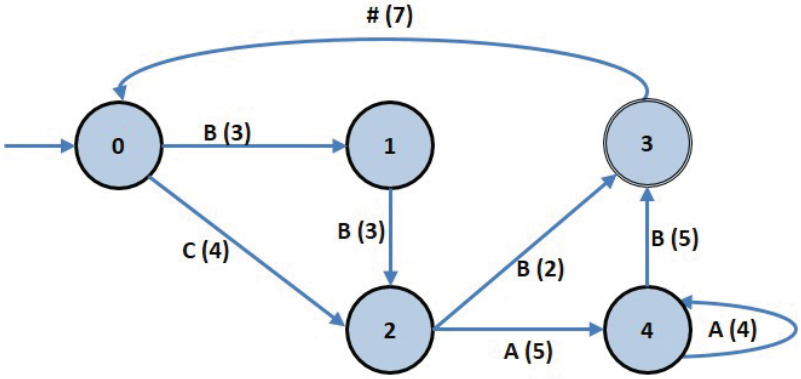


Figure 4.6: 5-state PFSM explaining Gaines data D

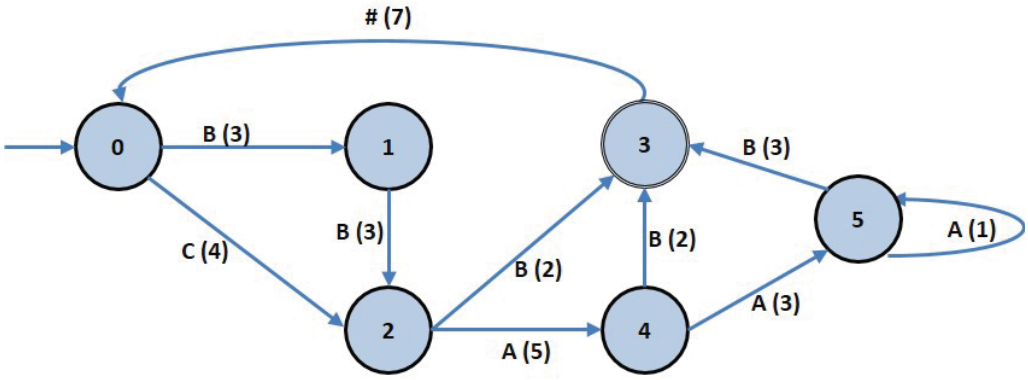


Figure 4.7: 6-state PFSM explaining Gaines data D

Table 4.3 shows the two-part code lengths of the PFSMs explaining Gaines data.

Table 4.3: Code Lengths in bits for the PFSMs in Figures 4.2 - 4.7

States	**Two-Part code lengths in bits on Gaines Data using Eq.(4.6)**
1	77.4146
2	76.1269
3	70.8393
4	66.6234
5	78.6874
6	90.3504

Of the various PFSMs explaining Gaines data D, the PFSM with four number of states gives the least two-part code length. With the increase in the number of states, the likelihood of the data generated using the PFSMs increases as the code length becomes shorter and shorter. But a further increase in number of states beyond four states starts reducing the likelihood. Thus for the Gaines data, the best PFSM is the PFSM with four states in Figure 4.5, that describes the data in the best possible manner.

As another example we consider the sequence of events represented as $\{ABCD\#\}$. The number of different PFSMs explaining the sequence of events are shown in Figure 4.8. The PFSMs explaining the sequence start with one state to four states in the figure. The one-state PFSM gives the most compact representation and also the best representation in terms of the two-part code length resulted by the model. The 5-state theory is the worst theory that can be formed at this stage as it results in the largest two-part code length. The code lengths can be seen in column-2 of Table 4.4. The one-state theory that does not assign any specific order to any of the events is the most likely one and the initial guess that a compact theory is indeed the best seems convincing.

Now, if we suppose that the sequence of events repeats 100 times, then the initial guess about the most compact theory being declared as the best, is no more valid. The third column of Table 4.4 can be referred to see the amount of code length

decrease with the increase in the number of states. In this case, the 5-state theory is most likely than the other theories that are trying to explain the same sequence of events. So, Table 4.4 is actually doing a comparison of two-part code lengths in bits for the different PFSMs for the events $\{ABCD\#\}$ observed once and the same sequence of events observed 100 times, represented as $\{(ABCD\#)^{100}\}$.

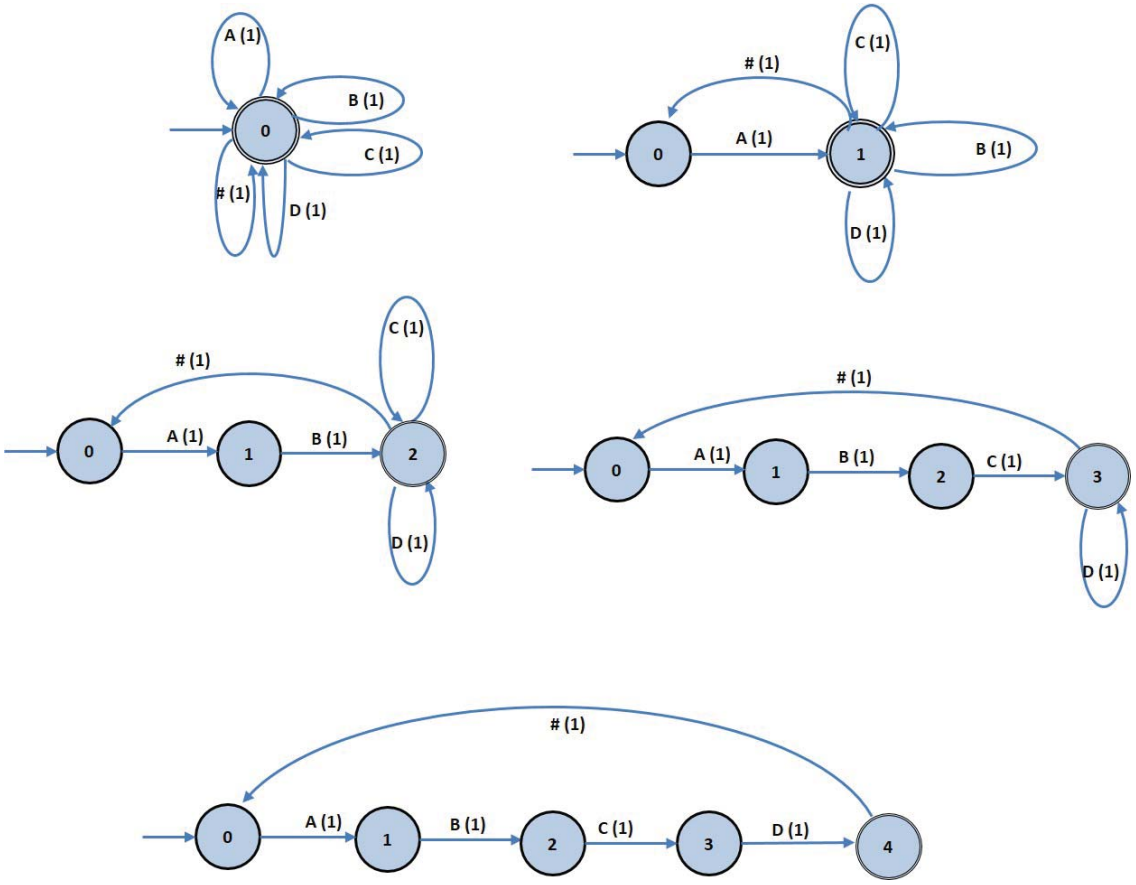


Figure 4.8: PFSMs explaining the sequence $\{ABCD\#\}$

Table 4.4: Code Lengths in bits for the PFSMs explaining $\{ABCD\#\}$ and $\{(ABCD\#)^{100}\}$

States	**Code Lengths in bits for $\{ABCD\#\}$**	**Code Lengths in bits for $\{(ABCD\#)^{100}\}$**
1	**24.1130**	1190.720
2	33.7568	843.610
3	34.9332	528.456
4	36.5754	260.125
5	37.5659	**65.856**

4.4 Inference of PFSM using MML

The number of PFSMs that can exist for a given data sequence is computationally intractable (Raman et al., 1997). For S number of states in a PFSM, the number of PFSMs that all can account for the same data sequence, is exponential in S. According to Gold (1978), Angluin (1978) and Feldman (1972), the problem of searching the minimal PFSM from a given data sequence is known to be NP-complete. Therefore in searching the best PFSM to account for a given data sequence along with making the problem of searching tractable, a trade-off between the guarantee of optimality of the solution is done with tractability considerations in mind, and this is achieved using heuristics.

To infer a stochastic PFSM with transition probabilities that best reflects a given data sequence, Raman (1997), Raman and Patrick (1997a), Clelland (1995), Clelland and Newlands (1994), Raman and Patrick (1997b), Collins and Oliver (1997) and Hingston (2001), propose various approaches. The method on inferencing works along the lines of building a minimal or maximal canonical PFSM. The minimal PFSM to start with is the one state PFSM where the states are split in each iteration until no further split is possible. The maximal PFSM to begin with is the Prefix Tree Acceptor (PTA) of the data sequence. The states in the PTA are progressively merged until no further merge is possible.

In this section, we discuss two methods of inferencing a PFSM, represented as a PTA of the data sequence. The methods work along the lines of ordered merging and random merging of states in the PFSM. The method of ordered merging is completely novel and proposed by us. This method uses a greedy search heuristic to find a near optimal PFSM and most suitable for applications where optimality can be compromised for quickness of the solution. The second method proposed by Raman and Patrick (1997b) uses a simulated annealing heuristic to find an optimal solution. This method can guarantee an accurate solution by appropriate setting of the *temperature* and *rate of cooling*. We have re-implemented this method in our own way by doing minimal modifications in the original algorithm in the way

that suits our needs. By optimal solution we mean the minimum two-part code length PFSM and the objective function to get the optimal solution is the Minimum Message Length (MML) principle.

4.4.1 Inference of PFSM by Ordered Merging (OM)

For inference using Ordered Merging (OM), the induction process begins by considering the input in the form of a PTA of the data sequence. The node pairs are merged in stages satisfying two major constraints:

- First, the merging always remains deterministic. That is, when two states are merged, the transitions on any input symbol on the merged states should be unambiguous. Any symbol from the merged states should not lead transitions to more than one state. If this is not satisfied, the second condition is never tried.

- Second, the two-part code length of the new PFSM after merge is lesser than the two-part code length of the PFSM before merge. Since the method works along the lines of searching using greedy search heuristic, the merge is only tried when the two-part code length of the PFSM after merge is lesser than the two-part code length of the PFSM before merge.

The stages of the node pair merges are given in detail in the subsections below:

4.4.1.1 First Stage Merge

In the first stage of the merge process, the final states of the initial PFSM are merged. The final states are those states in the PFSM that have transitions on input symbol #. This transition on input symbol # leads back to the initial state of the PFSM.

To see the effect of this merging, we reconsider the PFSM in Figure 4.1, which is the PTA of the Gaines data. The two-part code length calculated for this PTA using Equation 4.6 is **181.205** bits. The final states with input symbol # transitions

from them, are merged and the resulting PFSM after merging final states is shown in Figure 4.9. The two-part code length of the new PFSM is **136.365** bits.

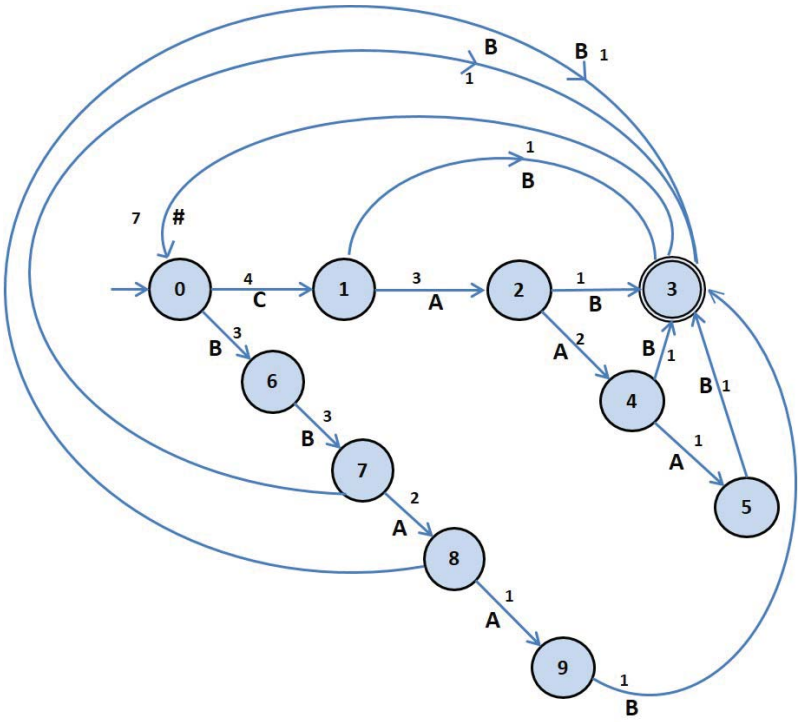


Figure 4.9: PFSM with final states merged from Figure 4.1

An important observation to be noted here is, it is not necessary that all the final states get merged as one by applying the First Stage merge Process. So there might be chances of still getting more than one final states after the First Stage Merge is over. This is attributed to the constraints put on merging.

4.4.1.2 Second Stage Merge

In the second stage of the merge process, the states directly connected to the final states are merged with each other. No merging with the final states is done here. The process of merging through the list of states directly connected with the final state continues until no further merging turns beneficial. The effect of applying the Second Stage Merge is seen in Figure 4.10. The two-part code length of this PFSM is **66.6234 bits**.

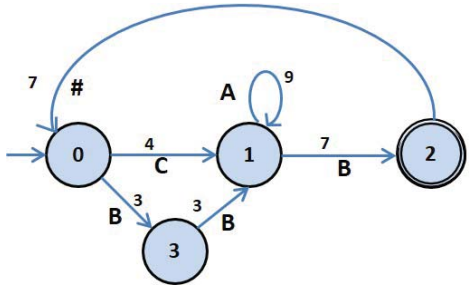


Figure 4.10: PFSM with Second Stage states merged from Figure 4.9

4.4.1.3 Third Stage Merge

In the third stage of the merge process, pair of states are merged where one state is the final state and the other state is the state connected to it. We loop through the list until we reach a PFSM whose code length is the minimum. This PFSM can now be termed as the MML PFSM. This kind of stage-wise merging is more systematic as opposed to the random merge pairs in the Beam Search Algorithm (Raman and Patrick, 1997b). Merging states in Third Stage of the inference process generates the same PFSM as in Figure 4.10.

4.4.1.4 Ordered Merging (OM) algorithm

We formally write the method of inferencing through Ordered Merging in the form of an algorithm. The merges achieved in different stages are mentioned as three separate algorithms. Algorithm-2 is very much similar to Algorithm-1. The only difference between the two algorithms is, the merging is applied on different lists in both the cases. Algorithm-1 considers the list of final states for merging the states, whereas Algorithm-2 considers the list of states directly connected to final states for merging the states. The two-part code length calculated using MML is used as an objective function to obtain the minimum code length PFSM.

Algorithm 1 Ordered Merging First Stage

Input: PTA
Output: Reduced PFSM after First Stage Merge

```
1: ListOfFinalStates ← List of Final States
2: NoOfFinalStates ← Number of Final States
3: old_MML ← CodeLengthOfCurrentPFSM
4: for i ← 1 to NoOfFinalStates − 1 do
5:     for j ← i + 1 to NoOfFinalStates do
6:         if (CanBeMerged(ListOfFinalStates[i], ListOfFinalStates[j])) then
7:             new_MML ← Merge(ListOfFinalStates[i], ListOfFinalStates[j])
8:             if (new_MML ≤ old_MML) then
9:                 old_MML = new_MML
10:                i ← i − 1
11:                break
12:            else
13:                UndoMerge()
14:            end if
15:        end if
16:    end for
17: end for
```

Algorithm 2 Ordered Merging Second Stage

Input: PFSM resulting from First Stage Merge
Output: Reduced PFSM after Second Stage Merge

```
1: ListOfStates ← List of states directly connected to Final States
2: NoOfStates ← Number of states directly connected to Final States
3: old_MML ← CodeLengthOfCurrentPFSM
4: for i ← 1 to NoOfStates − 1 do
5:     for j ← i + 1 to NoOfStates do
6:         if (CanBeMerged(ListOfStates[i], ListOfStates[j])) then
7:             new_MML ← Merge(ListOfStates[i], ListOfStates[j])
8:             if (new_MML ≤ old_MML) then
9:                 old_MML = new_MML
10:                i ← i − 1
11:                break
12:            else
13:                UndoMerge()
14:            end if
15:        end if
16:    end for
17: end for
```

Algorithm 3 Ordered Merging Third Stage

Input: PFSM resulting from Second Stage Merge
Output: Minimum Code Length PFSM or the Inferred PFSM

```
1: old_MML ← CodeLengthOfCurrentPFSM
2: new_MML ← 0
3: ListOfFinalStates ← List of Final States
4: NoOfIterations ← Initial Number Of Iterations
5: i ← 0
6: while (new_MML < old_MML OR i ≤ NoOfIterations) do
7:     state1 ← Random State from ListOfFinalStates
8:     state2 ← Random State from States connected to state1
9:     if (CanBeMerged(state1, state2) then
10:        new_MML ← Merge(state1, state2)
11:        if (new_MML ≤ old_MML) then
12:            old_MML = new_MML
13:            new_MML ← 0
14:        else
15:            UndoMerge()
16:            i ← i + 1
17:        end if
18:    else
19:        i ← i + 1
20:    end if
21: end while
```

4.4.2 Inference of PFSM using Simulated Annealing (SA)

We are basing the inference of a PFSM using Simulated Annealing (SA) by following a path that considers random merging of nodes instead of ordered merging. Raman and Patrick (1997b) use the method of inference using SA to get the minimal code length PFSM and we briefly explain the method in this section.

To understand the method of inferencing using SA, we first try to explain what Annealing is. Annealing is a physical process in which metals are heated to very high temperature and then they are gradually cooled. The high temperature causes the electrons of the metal to emit photons and in this process of doing so, the metal gradually descends from high energy state to low energy state. The emitted photons may bump into another electron, causing it to move to high energy state, but the probability of happening this decreases with cooling. The probability p is quantified as $p = e^{-\frac{\Delta E}{kT}}$, where ΔE is the positive change in energy level of electron, T is the

temperature and k is Boltzmann's constant. The rate of cooling is called annealing schedule and it plays a very important role in the formation of the final product. Rapid cooling will prohibit the electrons to descend to lower energy state and as a result there will be formation of regions of stable high energy. Too much slow cooling will be a waste of time. Therefore there has to be an optimum choice of the annealing schedule and that is determined empirically.

4.4.2.1 Simulated Annealing (SA)

In simulated annealing procedure, the Boltzmann's constant is irrelevant as it is specific to the physical process. Therefore the probability formula now becomes $p' = e^{-\frac{\Delta E}{T}}$. As ΔE refers to the change in the energy state of an electron in Annealing procedure, in Simulated Annealing it refers to the change in objective function. Thus in this case ΔE is the change in the two-part code length of the current PFSM. The temperature in Annealing is set in *kelvin* and in SA, the temperature is set to some value which is number of bits. The value is again determined empirically as in Annealing.

The modified probability is now given by $p' = e^{\frac{-(new_{CodeLength} - old_{CodeLength})}{T}}$, where $new_{CodeLength}$ is the code length of the changed PFSM after applying node merge and $old_{CodeLength}$ is the code length of the PFSM before applying merge. The change in the objective function is guaranteed to be a positive quantity as the probabilistic acceptance is only tried when the $new_{CodeLength}$ is larger than the $old_{CodeLength}$. For a negative change in the objective function, where the $new_{CodeLength}$ is always smaller than the $old_{CodeLength}$, the merge is always considered as it results in better solution.

The Simulated Annealing method of inferencing is formalized as an algorithm in Section 4.4.2.2.

4.4.2.2 Simulated Annealing (SA) algorithm

Algorithm 4 Simulated Annealing

$old_{MML} \leftarrow Code\ Length\ of\ Current\ PFSM$
$Temperature \leftarrow Initial\ Temperature$
$CurrentState \leftarrow Intitial\ State\ of\ current\ PFSM$
while $Temperature \neq 0$ **do**
    $RandomState1 \leftarrow Random\ state\ of\ current\ PFSM$
    $RandomState2 \leftarrow Random\ state\ of\ current\ PFSM$
    while $RandomState1 = RandomState2$ **do**
        $RandomState1 \leftarrow Random\ state\ of\ current\ PFSM$
        $RandomState2 \leftarrow Random\ state\ of\ current\ PFSM$
    end while
    if $CanBeMerged(RandomState1, RandomState2)$ **then**
        $new_{MML} \leftarrow Merge(RandomState1, RandomState2)$
        if $new_{MML} \leq old_{MML}$ **then**
            $old_{MML} = new_{MML}$
        else
            $p = e^{\frac{-(new_{MML} - old_{MML})}{Temperature}}$
            $RandomNumber = Random(0, 1)$
            if $RandomNumber \leq p$ **then**
                $old_{MML} = new_{MML}$
            end if
        end if
    end if
    $Temperature \leftarrow Temperature - 1$
end while

4.5 Experiments and Discussion

In this section we run the two algorithms for PFSM inference on a few randomly generated strings that were generated using two initial PFSMs.

The first initial PFSM is the Gaines machine in Figure 4.5. The PFSM though labels all the 6 arcs in the figure with transition frequencies but we assume them to be not present there and we generate random number of strings by setting up initial probabilities of transition between the transition arcs. The alphabet size is 4 including the delimiter symbol. We assume a uniform prior for a multinomial distribution case. To cover the possibility of including all the arcs being visited

atleast once in the process of random string generation, we start with a minimum of 50 number of strings and go upto a maximum of 1000 number of strings.

Figure 4.11 shows a comparison of the two-part code length computation using the **true PFSM model** of Figure 4.5, the inferred PFSM model induced by **Ordered Merging** induction method and the inferred PFSM model induced by using **Simulated Annealing** search method.

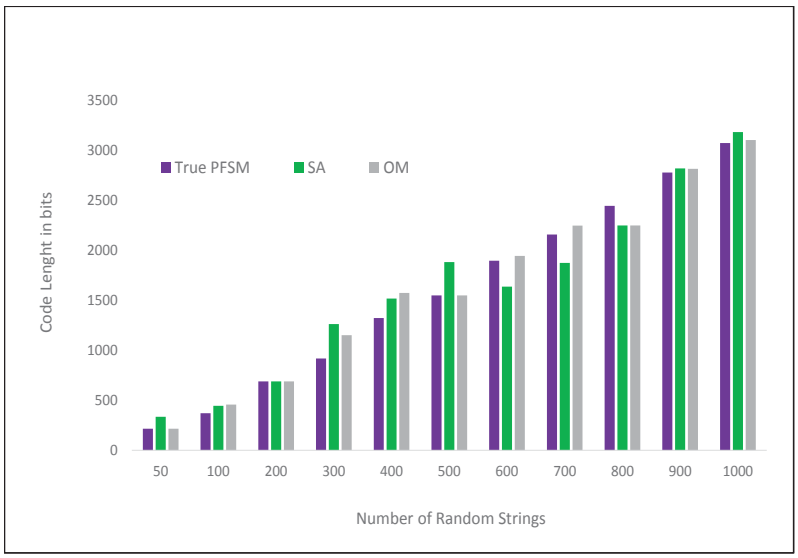


Figure 4.11: Two-part Code Length comparison between the true PFSM model of Figure 4.5, the inferred PFSM model by OM and the inferred PFSM model by SA

Experimental results indicate that the Simulated Annealing approach is able to correctly infer and converges to the true model approximately 2 times of the total 11 trials with different number of random strings. The Ordered Merging approach on the other hand shows a better performance by converging to the true model approximately 4 times of the total 11 trials. We started the Simulated Annealing search by setting the initial temperature to 20000 bits. We experimented with different temperatures and the most reasonable to consider was 20000 bits. A value higher than 20000 bits considered for the initial temperature, only took longer time to search through the search space to result an optimal solution but, did not give a better result than what we generated from this 20000 bits value.

We also plot a curve for the *Error* obtained in the process of inferring by the two inference methods against the different test strings. The *Error* is calculated as $(1 - (\frac{codelength(inferred_{PFSM})}{codelength(true_{PFSM})}))$. The plot is shown in Figure 4.12.

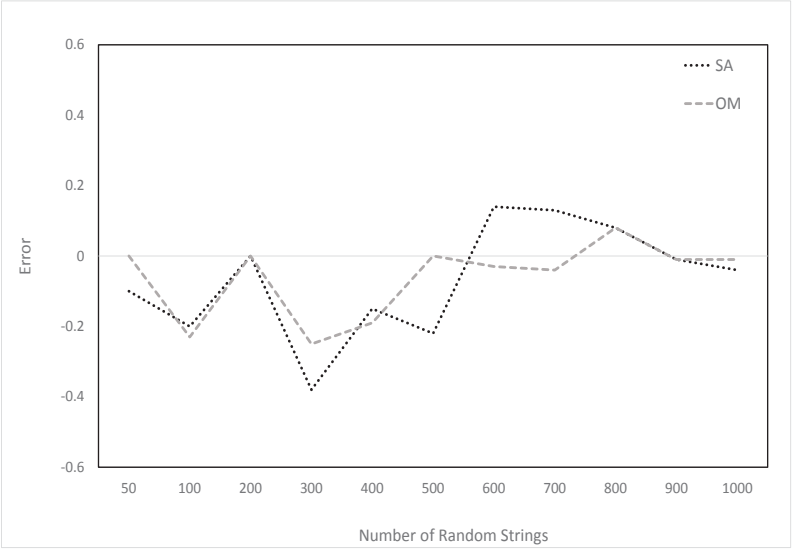


Figure 4.12: Error measured while inferring PFSMs on test strings using SA and OM methods of inference

The *Error* plot in the figure indicates that the differences between the two-part code lengths of the true PFSMs and the inferred PFSMs resulted by Ordered Merging approach are less as compared to the ones inferred by Simulated Annealing approach. This indicates a possibility of less error seen in the method of inference using the OM method. But the method purely works on greedy lines of selecting a suitable path from the current situation. The SA approach, although resulted in more error than the OM method, does a more extensive search of the search space. By the process of random merging applied on the current PFSM, if the machine gets into a worse solution (which might happen as high temperature increases the probability of acceptance of worse solutions), it becomes necessary to keep track of a few best machines found so far, so that there is a way to backtrack. The negative *Error* in the figure indicates some sort of redundancy in the randomly generated structure.

The second initial PFSM that we consider is the machine in Figure 4.13.

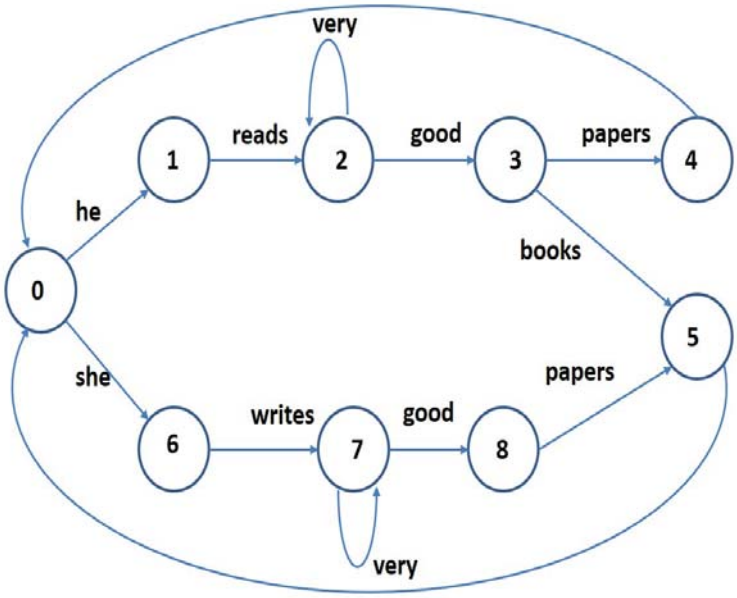


Figure 4.13: Sample PFSM

This sample PFSM represents a small scale representation of the dataset, of which we try to build a PFSM model from the corpus of Enron spam dataset. We speak about the details of dataset in Chapter 5. The only difference is that the PFSM in Figure 4.13 represents the model constructed at sentence level and, for the Enron spam dataset, we build the model working at word level. For this sample PFSM again, considering a uniform probability distribution from each state on the different symbols, random number of strings are generated ranging from number 50 to number 1000. By assigning unique characters to each word labelling the arcs in the PFSM, the total alphabet size computed is 9 including the delimiter symbol.

The code length comparisons between the true models and the inferred models using the two inference methods for the different number of random strings are shown in Figure 4.14 and the error plot is shown in Figure 4.15.

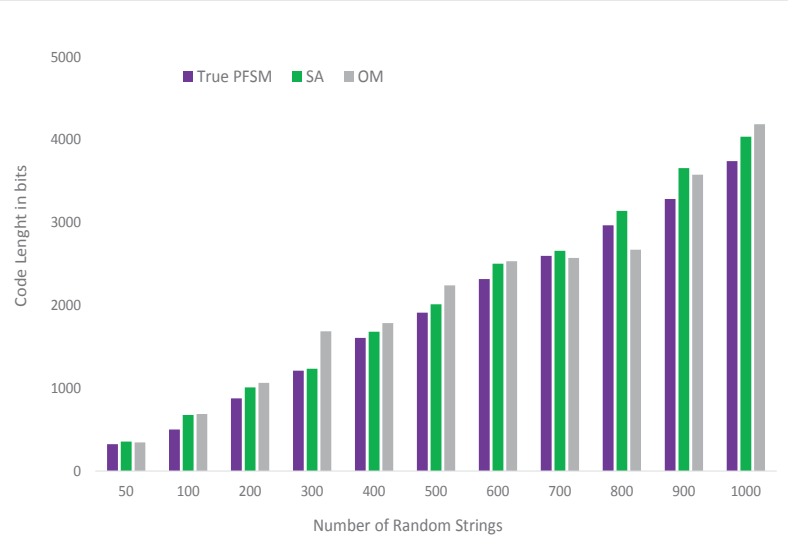


Figure 4.14: Two-part Code Length comparison between the true PFSM model of Figure 4.13, the inferred PFSM model by OM and the inferred PFSM model by SA

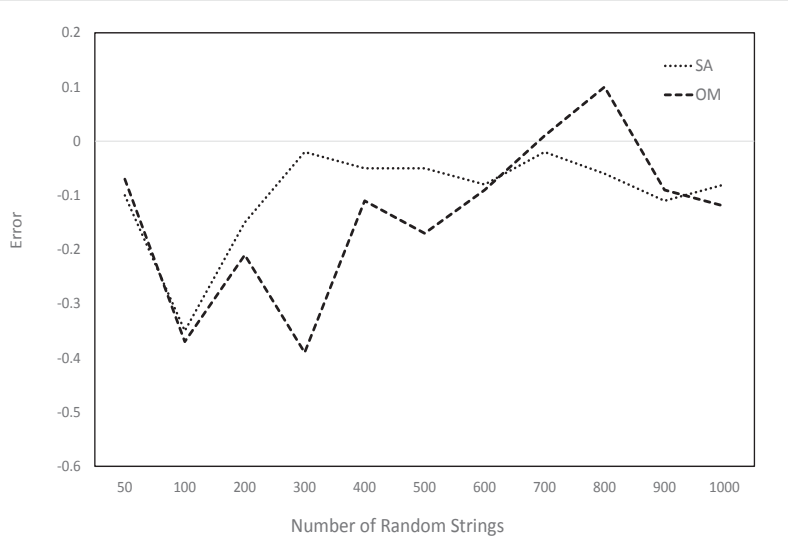


Figure 4.15: Error measured while inferring PFSMs on test strings using SA and OM methods of inference

The results for the PFSM of Figure 4.13 indicate that the inferred PFSMs were always resulting in more two-part code lengths than their corresponding true PFSMs. For this second example the performance of the SA inference approach was better than the OM inference approach as the error encountered in the process on inferring

using the SA method was in the range of -0.35 to -0.02. Whereas for the OM method, the error was in the range of -0.39 to 0.1. This second example had more number of states with more large input alphabet size than the Gaines machine of Figure 4.5. For even more big examples like the one that we will be considering for learning PFSMs mentioned in Chapter 5, the SA method can be a reliable choice. The method does an extensive search of the search space at an added cost of additional space which is required to keep the track of a few best PFSMs.

4.6 Summary

On the concluding notes we summarize the whole chapter by leading a discussion that first describes a method to encode an inductive hypothesis modelled as a PFSM. The modelling of a PFSM includes modelling the structure of PFSM followed by modelling data using the structure. This two-part modelling is correlated with the Bayesian theorem. So relating with the Bayesian theorem, the code-length of the posterior probability of a hypothesis modelled as a PFSM is calculated by summing up the code length of the prior probability of the hypothesis and the code length of the transition probabilities.

The first part of code length i.e. the code length of the PFSM structure itself is modelled by making use of some known information or prior information about the PFSM. This prior information includes the number of states of PFSM, the cardinality of the input alphabet set and the number of arcs from each state of the PFSM. The second part of the code length, i.e. the code length of the data using the structure of PFSM, is calculated by encoding the transition probabilities from each state of the PFSM model. The code length calculation method makes use of the transition frequencies instead of transition probabilities and that is known from the data that is observed. The incremental code for a multinomial distribution case with a uniform prior is proportional to (approximately normalized) $p_1^0, p_2^0, ..., p_m^0$ and the adaptive coding starts with pre-counts of 1 in each class.

Then in the further discussions, we speak about the MML inference methods to find a minimal PFSM. We discuss two methods here. One is the way through Ordered Merging (OM) of node pairs and another way is through the random merging of node pairs using the Simulated Annealing (SA) heuristic. Both the methods, in general, start the process of induction by considering the initial PFSM in the form of a Prefix Tree Acceptor (PTA) of the data sequence. The OM method follows a sequence of node pair merges in different stages and the approach used is the greedy approach. By the use of greedy approach, the node pair merge that does not result in lesser two-part code length than the original two-part code length, is not considered for merging. The SA approach on the other hand might consider a worse solution in favour of future better solutions. The SA approach simulates the physical annealing process where a metal is heated to a very high temperature and then gradually cooled. The probability of accepting a bad solution is higher in the initial stages of merging when the temperature is high. But as the temperature gradually decreases, this tendency of accepting bad solutions also decreases. In the SA approach, there is a necessity to use an extra space at the cost of doing more extensive search, to keep track of the best solutions found so far.

The two methods of PFSM inference are then applied on a few randomly generated strings using two initial PFSM structures. The first structure is the Gaines machine in Figure 4.5 and the second structure is the machine in Figure 4.13. Random strings are generated staring from number 50 to number 1000 by setting up initial probabilities of transition from each state of the PFSM. The two-part code length of the true PFSM that generated the strings is compared against the two-part code length of the inferred PFSMs and the error is noted down. For the first structure the OM method performs better as the overall error noted down in the process of inferring is less as compared to the SA method. But, for the second structure which is bigger than the first structure in terms of number of states and cardinality of the input alphabet set, the SA method results in better performance than the OM method. In general, both the methods used for the second structure did not

converge to the true PFSM model in any of the cases considered for the trainings sets. This may be attributed to two facts. The first one is that the initial PTA PFSMs built using the training random strings are very large and certain deterministic conditions imposed while merging the node pairs never allows certain merges to happen. The second fact is that the random structure is redundant.

In the chapters ahead, the method of inference that we will be using is the SA method as the method atleast guarantees more possibilities of node pair merges than the OM method. The setting of the initial temperature and the rate of cooling can be determined empirically depending upon the problem context.

Chapter 5

Learning Two-Machine PFSM models

5.1 Introduction

This chapter is an application to learning and inference of PFSM models using MML on a text based corpus and the objective is to observe the amount of correctness achieved while trying to classify some text from the same corpus whose label is unknown. PFSM models can be easily constructed from training examples that contain repetitive text and the patterns are captured well with a further possibility of compression. We are considering a case of supervised learning where the texts belonging to different categories are known in advance. The text is viewed as a sequence of words or tokens separated by delimiter symbols.

The two-machine model learning approach learns the PFSM models from the classified text categories. The learnt models are then inferred to reflect the best and most compact representations. These trained models are then used to test the set of text tokens whose label is unknown. The testing is done by measuring the amount of increase in the two-part code lengths when the set of test text tokens is made input to both the trained models. The trained model that measures minimal

increase in the two-part code length is more probable to have generated the set of test tokens and this way the set of test tokens are categorized.

This chapter focusses on how the models are learnt and inferred from the training sets. We apply the method of learning and inference on the real world datasets namely the Enron spam datasets (Enr) and the Activities of Daily Living (ADL) datasets (Ordonez et al., 2013). The Enron spam datasets contain the text-based classified spam and non-spam datasets and the models can be learnt directly from them without the need of any preprocessing to be done on the text files. The voluminous text corpus is reduced to a sequence of tokens by the application of a feature selection process. The ADL datasets on the other hand first require a preprocessing to be done on the raw data before learning can be applied on them. We discuss the details of the datasets in the chapter in further sections.

5.2 Design of the two-machine PFSM models

The word two-machine is emphasized to get an idea of learning two models from two categories. The examples that we have considered are the ones that strictly possess two categories of data. For the sake of convenience let us denote the two categories as, $class1$ and $class2$ respectively. The idea of two-machine approach is not new. Most of the data compression techniques doing classification rely on building two models. One model is constructed from a sequence of positive examples and the other one from a sequence of negative examples.

To understand, we consider the following tokens in $class1$ and $class2$.

$$class1 = \{(aab\#)^8(abc\#)^5(abb\#)^2(baa\#)^6\},$$
$$class2 = \{(aab\#)^1(abc\#)^5(abb\#)^{10}(bab\#)^5\}$$

The numbers in the sets denote the frequencies of occurrences of tokens in the sets. The interpretation is done like this. $Class1$ and $Class2$ commonly represent a few tokens that are seen in both classes. The amount of usage for those common tokens are different in each class. In addition to that $Class1$ contains tokens that

are not seen in *Class*2 and similarly, *Class*2 contains tokens that are not seen in *Class*1. This example is very much analogous to the corpus of spam and non-spam emails where the tokens represent words used in spam and non-spam emails. Certain words are used in high frequency in spam emails than the non-spam emails. And this is true the other way round also. There are certain words that are seen only in spam emails and they don't have a usage in non-spam emails. So, after breaking the emails into a sequence of tokens, the PFSM models are constructed. The two models corresponding to two different categories are constructed. The initial models as discussed earlier in Chapter 4 in Section 4.4.2 are the Prefix Tree Acceptors (PTAs) of the tokens. The initial models in the form of PTAs are shown in Figure 5.1 and Figure 5.2.

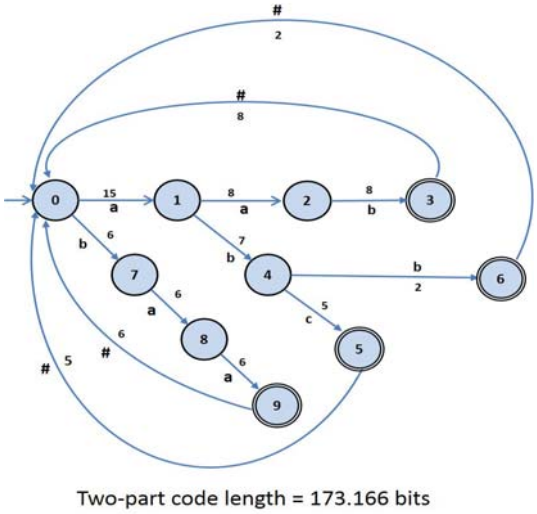


Figure 5.1: *Class1* PTA

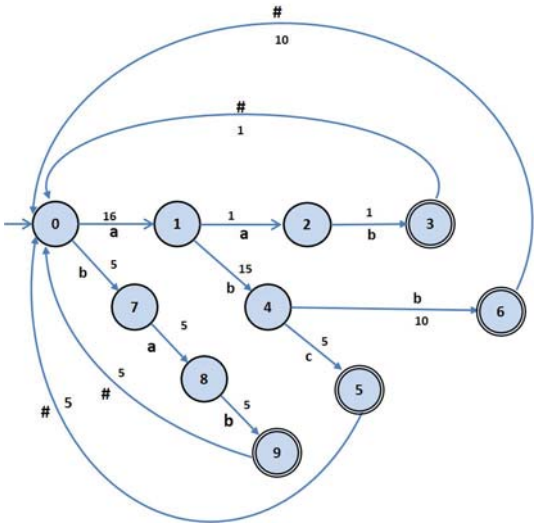


Figure 5.2: *Class2* PTA

Figure 5.2: *Class2* PTA

The PFSMs are inferred using the SA (Simulated Annealing) approach and are shown in Figure 5.3 and Figure 5.4.

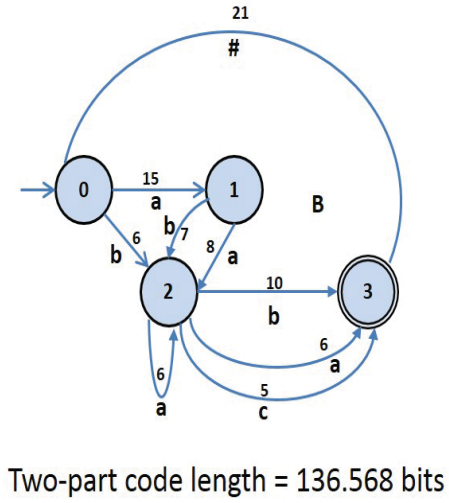


Figure 5.3: *Class1* Inferred PFSM

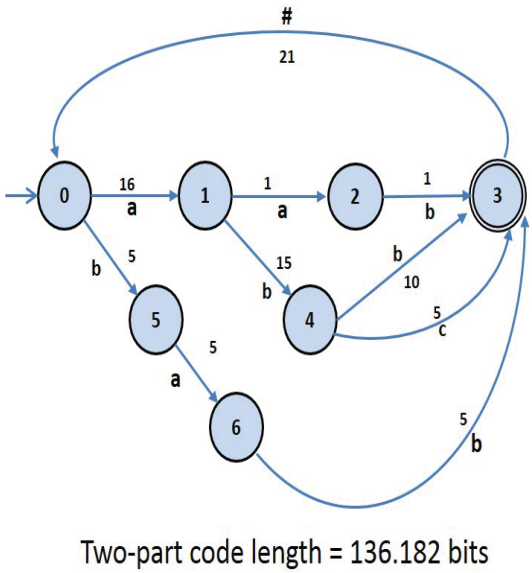


Figure 5.4: *Class2* Inferred PFSM

5.3 Classification using the two-machine PFSM models

The classification of the target sequence using the two-machine PFSM model is done in the following manner and pictorially shown in Figure 5.5.

- The tokens are first learnt from the training dataset that contains the two categories.

- Appropriate feature selection method can be applied for voluminous training datasets.

- The PFSM models are then built from the tokens learnt from the two categories.

- Next, the models are inferred using the Simulated Annealing search method that makes use of MML as an objective function in generating the most compact and the most probable form.

- The target sequence of tokens from the test dataset are input to the inferred PFSM models generated above.

- The model that measures minimal increase in the two-part code length becomes the classification class for the target sequence of tokens.

Let the target sequence whose class is unknown be denoted by the following set.

$Target = \{aab\#abc\#\}$

The target sequence is input to both the inferred PFSM models. The increase in two-part code length in $class1$ model is 4.4825 bits whereas, the increase in two-part code length in $class2$ model on the same target sequence is 7.1722 bits. Therefore, for the target sequence, the $class2$ model is more probable to have generated the sequence and thus becomes the classification class for the target sequence. The probability can be quantified as $\frac{2^{7.1722}}{2^{7.1722}+2^{4.4825}} = 0.865$.

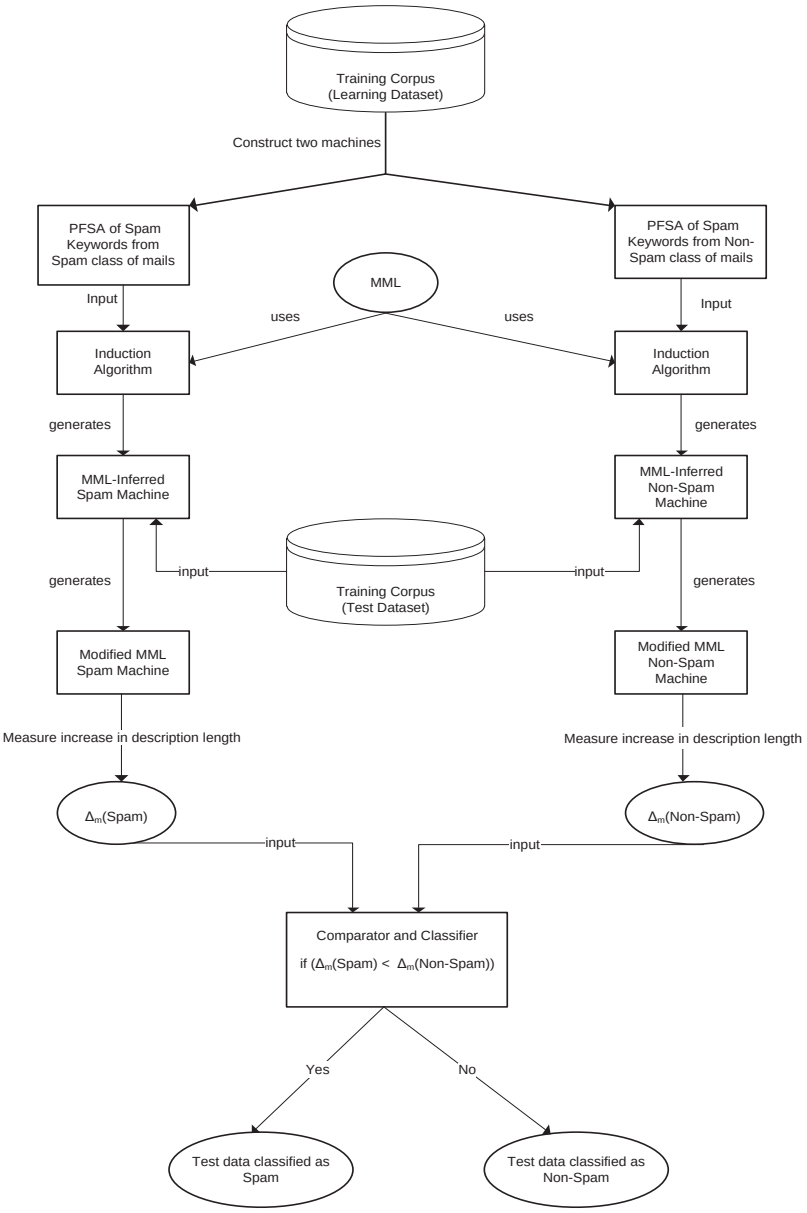


Figure 5.5: Classification using two-machine design

5.4 Experiments

The experiments are conducted in two different experimental set-ups. The description of the data sets used in the experiments, the evaluation metrics and the results obtained are discussed in distinguished sections. We also compare the results with the state of art classifiers.

5.4.1 Experimental set-up A

We have used the Enron spam datasets [1] in our experimental evaluation. The dataset is a collection of personal files of approximately 150 Enron employees made public. The experiment used the mailboxes of six employees namely, **farmer-d**, **kaminski-v**, **kitchen-l**, **williams-w3**, **beck-s** and **lokay-m**. The spam messages were collected from sources namely, **SpamAssasign corpus (SA)**, **Honey pot project (HP)**, **Bruce Guenter and Georgios Paliouras (BG, GP)**. The six non-spam message collections of the six Enron employees were paired with the spam collections from the above mentioned spam sources generating six Enron spam datasets as Enron-1, Enron-2, ..., Enron-6. The first three datasets, Enron-1 to Enron-3, used a non-spam - spam ratio of approximately 3:1 and the other three datasets used a non-spam - spam ratio of approximately 1:3. Each dataset consists of approximately 5000 - 6000 messages.

The six datasets are available in a preprocessed form. The following preprocessing steps were applied.

- The owner of the mailbox was removed from the messages by checking if the owner appeared in the "To:, Cc:, or Bcc:" fields of the messages.

- All the HTML tags and the headers of the messages were removed keeping only subjects and bodies of messages.

- The spam message containing non-Latin characters were removed because the non-spam messages were all written in Latin characters.

Table 5.1 summarizes the characteristics of the entire Enron spam dataset with details including the number of spam and non-spam messages in each dataset and the periods they were collected in.

[1]The Enron Spam datasets are available from http://www.iit.demokritos.gr/skel/i-config and http://www.aueb.gr/users/ion/publications

Table 5.1: Composition of the Enron spam datasets

Dataset	non-spam + spam	non-spam:spam	non-spam period, spam period
Enron-1	farmer-d + GP	3762:1500	[12/99, 1/02], [12/03, 9/05]
Enron-2	kaminski-v + SH	4361:1496	[12/99, 5/01], [5/01, 7/05]
Enron-3	kitchen-l + BG	4012:1500	[2/01, 2/02], [8/04, 7/05]
Enron-4	williams-w3 + GP	1500:4500	[4/01, 2/02], [12/03, 9/05]
Enron-5	beck-s + SH	1500:3675	[1/00, 5/01], [5/01, 7/05]
Enron-6	lokay-m + BG	1500:4500	[6/00, 3/02], [8/04, 7/05]

5.4.1.1 Model Training and Feature Selection

Each of the Enron spam dataset contains distinguished spam and non-spam messages as discussed above. For training the spam and the non-spam PFSM models, 50% of the messages in each category (spam and non-spam) were considered. The following feature selection procedure was applied and the models were built on the selected features. For convenience, let us denote the set of spam messages as SM and the set of non-spam messages as NSM.

1. The messages in the SM set were first tokenized by the application of the *Bag of Words* approach. This was done by the removal of stop words and delimiter symbols. As mentioned above 50% of the total spam messages were tokenized and it resulted in an enormous collection of features set.

2. To reduce the dimension of the feature space, the features were arranged in decreasing order of the frequency of their usage in 50% of the spam messages considered for training. In other words, a histogram representing the frequency of occurrence of each feature was formed.

3. From the total features collected, 3000 high frequency features were selected.

4. Each feature looked like a token with a certain frequency of occurrence.

5. The spam PFSM model was then built on these 3000 high frequency features.

The non-spam model was built by following the same procedure as listed above. A small subset of the selected features with their frequency of use in spam messages of the Enron-1 spam dataset can be seen in Table 5.2.

Table 5.2: Enron-1 Spam Features

Feature No.	Feature	Fequency	Feature No.	Feature	Fequency
1	http	201	29	stop	58
2	$	148	30	html	56
3	www	139	31	meds	55
4	click	127	32	send	55
5	free	121	33	site	55
6	email	114	34	biz	53
7	information	101	35	easy	53
8	money	99	36	future	53
9	time	99	37	info	53
10	online	88	38	limited	53
11	offer	80	39	dollars	52
12	mail	74	40	list	52
13	special	73	41	low	52
14	message	72	42	net	52
15	order	70	.	.	.
16	price	69	.	.	.
17	today	68	.	.	.
18	prices	67	2990	advertising	3
19	link	65	2991	aesthete	3
20	paliourg	65	2992	affiliates	3
21	back	64	2993	affirmative	3
22	receive	62	2994	affront	3
23	address	61	2995	aficionado	3
24	buy	61	2996	agate	3
25	visit	60	2997	agency	3
26	prescription	59	2998	agenda	3
27	world	59	2999	aggravate	3
28	office	58	3000	agnomen	3

Table 5.3 represents a small subset of the collection of non-spam features from the Enron-1 dataset.

Table 5.3: Enron-1 Non-Spam Features

Feature No.	Feature	Frequency	Feature No.	Feature	Frequency
1	enron	738	42	price	161
2	cc	691	43	ticket	161
3	forwarded	570	44	back	158
4	ect	534	45	production	158
5	pm	527	46	effective	156
6	hou	514	47	america	155
7	gas	506	48	daily	151
8	daren	495	49	set	151
9	hpl	490	50	fyi	146
10	corp	480	51	august	145
11	attached	410	52	june	145
12	deal	391	53	actuals	141
13	meter	383	54	clynes	141
14	questions	348	55	contact	137
15	farmer	303	56	gary	137
16	month	285	57	week	132
17	day	263	58	today	131
18	volume	258	59	number	126
19	call	253	60	plant	125
20	sitara	250	61	teco	123
21	mmbtu	247	62	bob	121
22	nom	245	63	days	121
23	file	229	64	mail	121
24	information	222	65	work	120
25	robert	204	66	tap	118
26	time	199	67	sale	117
27	contract	198	.	.	.
28	volumes	196	.	.	.
29	xls	195	.	.	.
30	july	192	2889	transmitted	4
31	change	190	2990	ttimeonline	4
32	ami	184	2991	tue	4
33	deals	180	2992	turnaround	4
34	chokshi	171	2993	typically	4
35	pat	171	2994	understood	4
36	flow	169	2995	unseen	4
37	make	169	2996	utilize	4
38	nomination	168	2997	vances	4
39	texas	167	2998	vanessa	4
40	energy	162	2999	vastar	4
41	north	161	3000	venture	4

5.4.1.2 Evaluation Procedure

In the evaluation procedure, spam recall (SR), non-spam recall (NSR), spam precision (SP) and non-spam precision (NSP) were used as the measures of performance evaluation. The number TP denotes true positives which is equal to the number of messages correctly classified as spam, the number TN denotes true negatives and is equal to the number of messages correctly classified as non-spam, the number FP denotes false positives and is equal to the number of non-spam messages misclassified as spam and the number FN denotes false negatives and is equal to the number of spam messages misclassified as non-spam. SR is calculated as $\frac{TP}{TP+FN}$, NSR is calculated as $\frac{TN}{TN+FP}$, SP is calculated as $\frac{TP}{TP+FP}$ and NSP is calculated as $\frac{TN}{TN+FN}$ (Metsis et al., 2006).

Weighted accuracy rate $WAcc$ and weighted error rate $WErr$ have also been used as measures for cost sensitive evaluation because precision and recall do not take into account the cost of misclassification done. The penalty for classifying a non-spam message as spam is more severe than letting a spam message pass the filter. Therefore Androutspoulos et al. (Androutsopoulos et al., 2000) introduced these cost sensitive measurements and they are defined as $WAcc = \frac{\lambda.TN+TP}{\lambda.N_s+N_{ns}}$ and $WErr = \frac{\lambda.FP+FN}{\lambda.N_s+N_{ns}}$ where N_s and N_{ns} are the number of spam and non-spam messages respectively.

Three different values of λ :1,b 9 and 999 were introduced by Androutspoulos et al. (Androutsopoulos et al., 2000). A value of λ equal to 1 denotes a scenario where classifying a non-spam message as spam and classifying a spam message as non-spam are equally penalized. Values of λ equal to 9 or 999 denote a scenario where classifying a non-spam message as spam is 9 or 999 times more severe. In our experiments the value of λ as 1, has been considered.

5.4.1.3 Testing and Results

The spam and non-spam PFSM models were trained on 50% of the classified message in each of the Enron spam dataset. Consequently, the remaining 50% were

considered for testing purpose. Table 5.4 provides a summary of the number of spam and non-spam messages considered for training and testing processes. We also mention the two-part code lengths of the spam and non-spam PFSM models in the same table.

Table 5.4: Summary of Training and Testing Dataset in each Enron Category

Dataset	**No. of Trained Messages**		**No. of Tested Messages**		**Two-Part Code Length in bits**	
	Spam	Non-Spam	Spam	Non-Spam	Spam Model	Non-Spam Model
Enron-1	750	1836	750	1836	758418	869953
Enron-2	748	2180	748	2181	692412	751346
Enron-3	750	2006	750	2006	721248	802158
Enron-4	2250	750	2250	750	834624	714673
Enron-5	1837	750	1838	750	897214	744318
Enron-6	2250	750	2250	750	884268	739217

The messages were tested in batches. The batches were formed by using the following sequence of steps.

1. In each Enron category, the testing messages belonging to spam and non-spam classes, were first grouped as one set. This grouping was done in the following way.

    - First the non-spam messages were organized according to their sequence of arrivals.
    - Random number of slots equal to the number spam messages in the Enron category, were created.
    - The spam messages were inserted according to their arrival sequence into these random slots.

2. After the grouping was done, the messages were divided into batches containing 100 messages in each batch. Each Enron category resulted in approximately 25-30 batches that were used for testing. A few last batches in a the Enron categories had less than 100 messages.

3. Each batch had a varying spam:non-spam ratio and was tested on the trained spam and non-spam PFSM models of its own Enron category.

4. We noted down the recall, precision and weighted accuracy % for each batch. We did not note down error as the λ value considered was 1 and the weighted error rate in that case is 1 - (weighted accuracy rate).

Figures 5.6 - 5.11 show the spam:non-spam ratio in each batch of the Enron-1 to Enron-6 datasets. Each batch is a combination of varying spam and non-spam messages and in total they contain 100 messages that are to be tested. Table 5.5 shows the spam:non-spam ratio averaged over each batch of the corresponding Enron category. Enron-1 to Enron-3 datasets have approximately three times more non-spam messages than the spam messages in each batch and Enron-4 to Enron-6 datasets have approximately three times more spam messages than the non-spam messages in each batch.

Table 5.5: Average Spam - Non-Spam ratio in each Enron category

Category	**Avg. Spam:Non-Spam ratio**
Enron-1	0.44
Enron-2	0.35
Enron-3	0.38
Enron-4	3.02
Enron-5	2.52
Enron-6	3.05

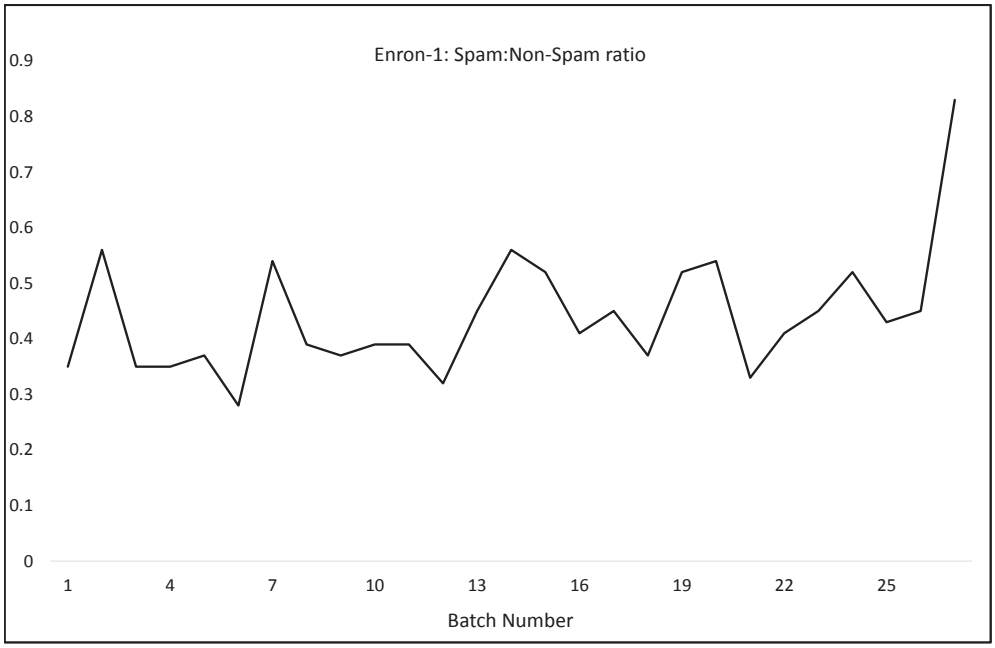


Figure 5.6: Spam:Non-Spam ratio in each Batch of the Enron-1 dataset

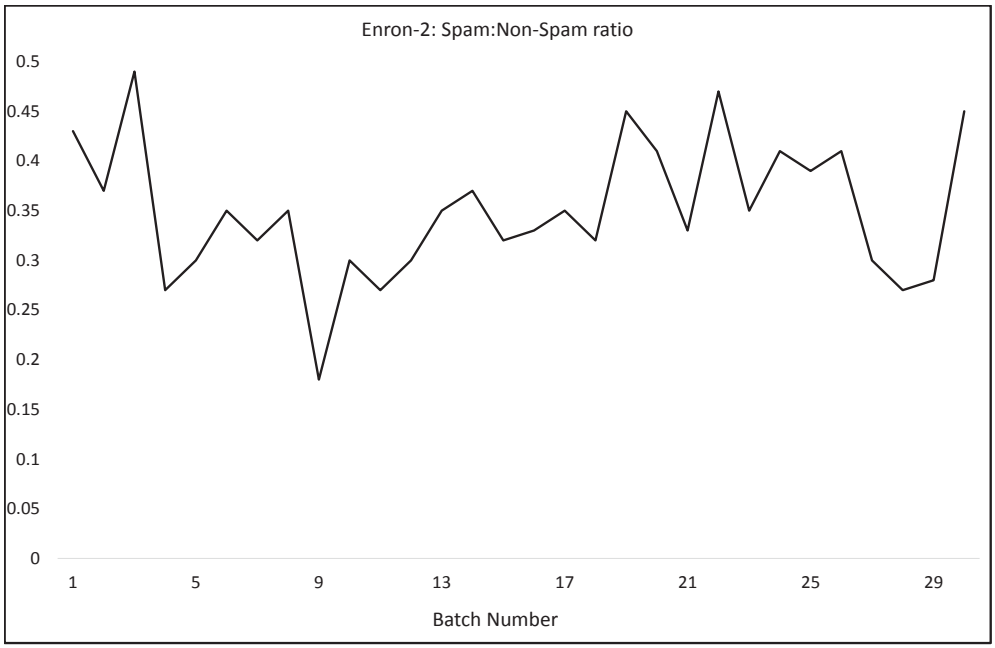


Figure 5.7: Spam:Non-Spam ratio in each Batch of the Enron-2 dataset

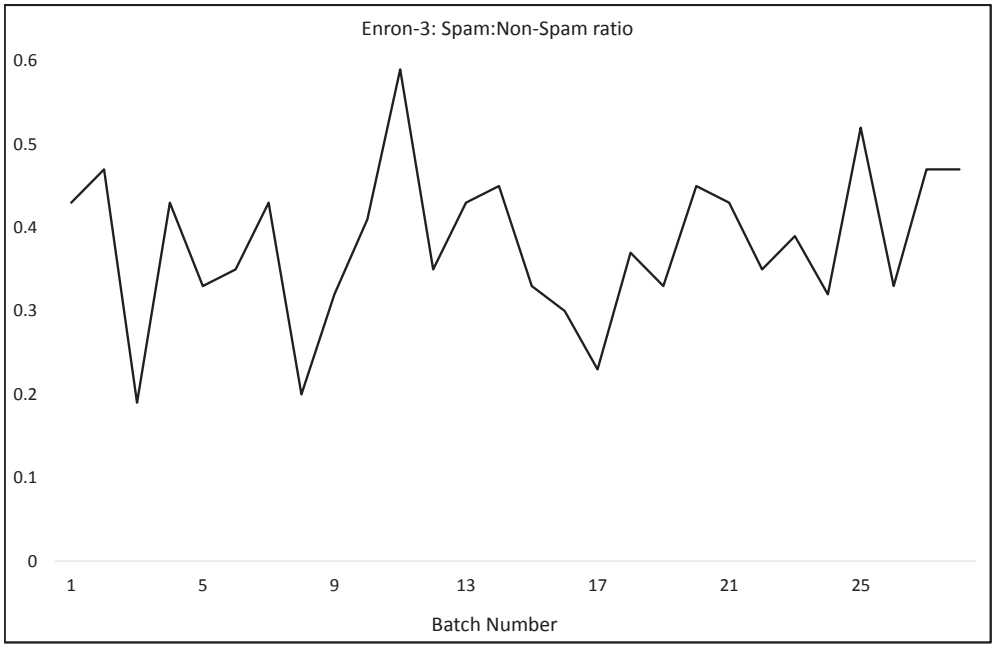


Figure 5.8: Spam:Non-Spam ratio in each Batch of the Enron-3 dataset

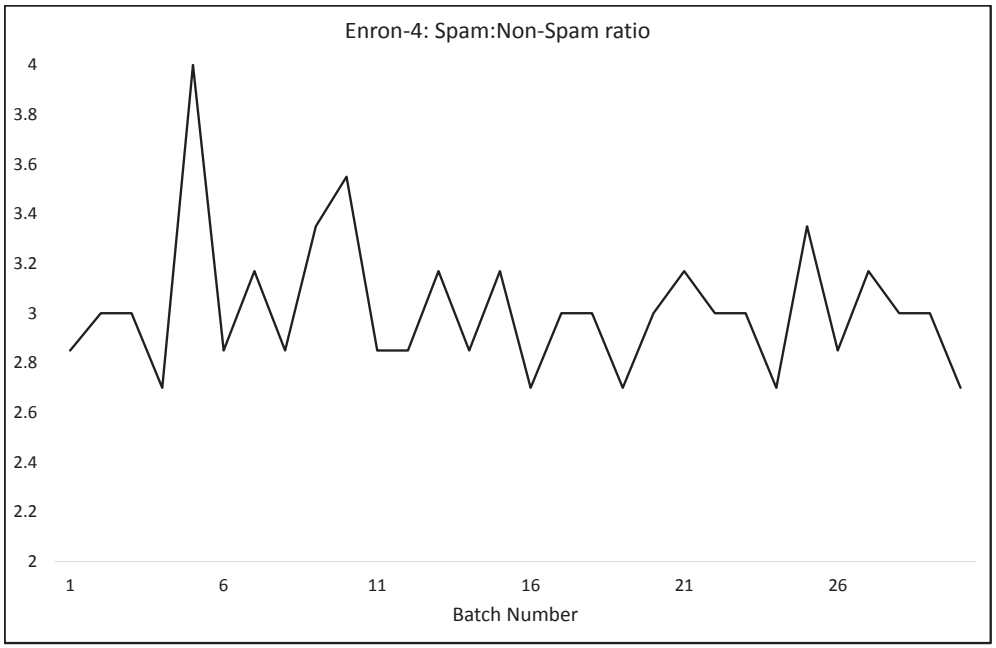


Figure 5.9: Spam:Non-Spam ratio in each Batch of the Enron-4 dataset

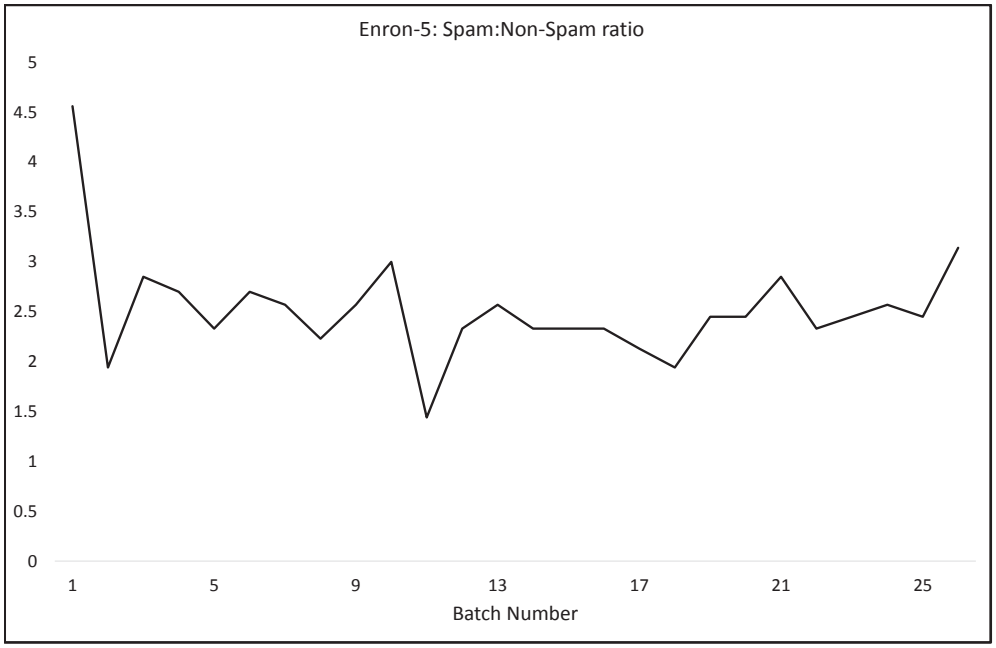


Figure 5.10: Spam:Non-Spam ratio in each Batch of the Enron-5 dataset

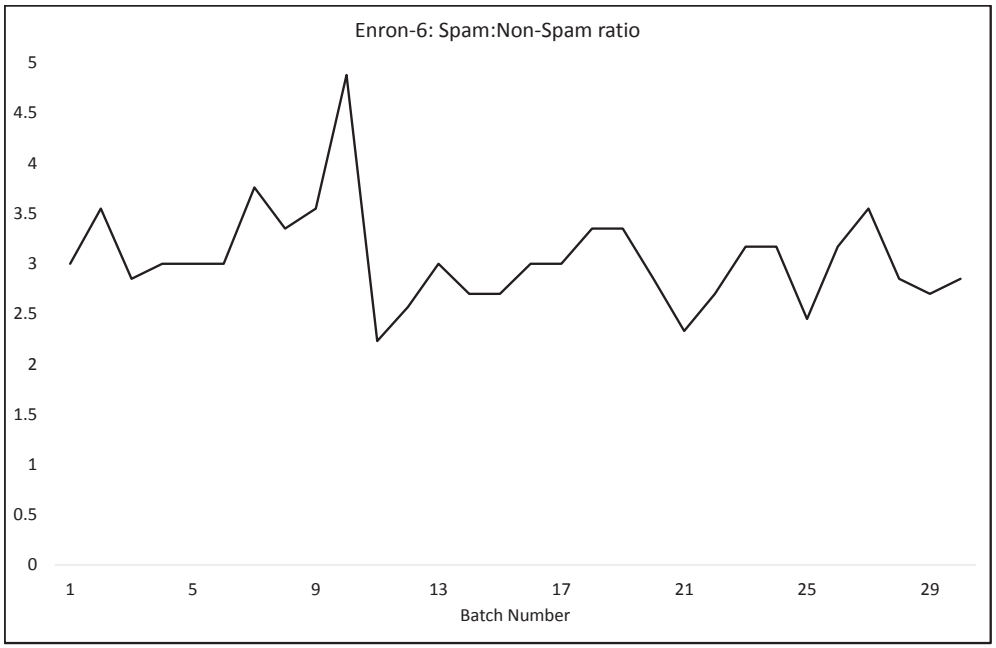


Figure 5.11: Spam:Non-Spam ratio in each Batch of the Enron-6 dataset

Figures 5.12 - 5.17 show the calculations for spam recall, non-spam recall, spam precision, non-spam precision and weighted accuracy in each batch of the tested messages of the Enron datasets. The figures don't show the results for each batch, instead the values have been averaged over each group of five batches and shown as % in the figures. The values are again summarized and shown in Table 5.6 for each of the Enron datasets. In each performance category the best results are bold faced.

Enron-2 dataset gives the best results for spam recall, non-spam recall, non-spam precision and also weighted accuracy among the other Enron datasets. Whereas Enron-4, Enron-5 and Enron-6 achieve the highest score for spam precision. The results achieved outperform the state of art classifiers. We compare our results computed on the two-machine model with a classifier (Almeida and Yamakami, 2012) which uses the subsequently related Minimum Description Length (MDL) principle (Rissanen, 1978). We discuss the MDL approach for classification and do a comparison of results with the MML approach in the next section. We have considered the same scenario for comparison in the MDL approach too and work on the same 3000 high frequency attributes that were considered for our approach using MML.

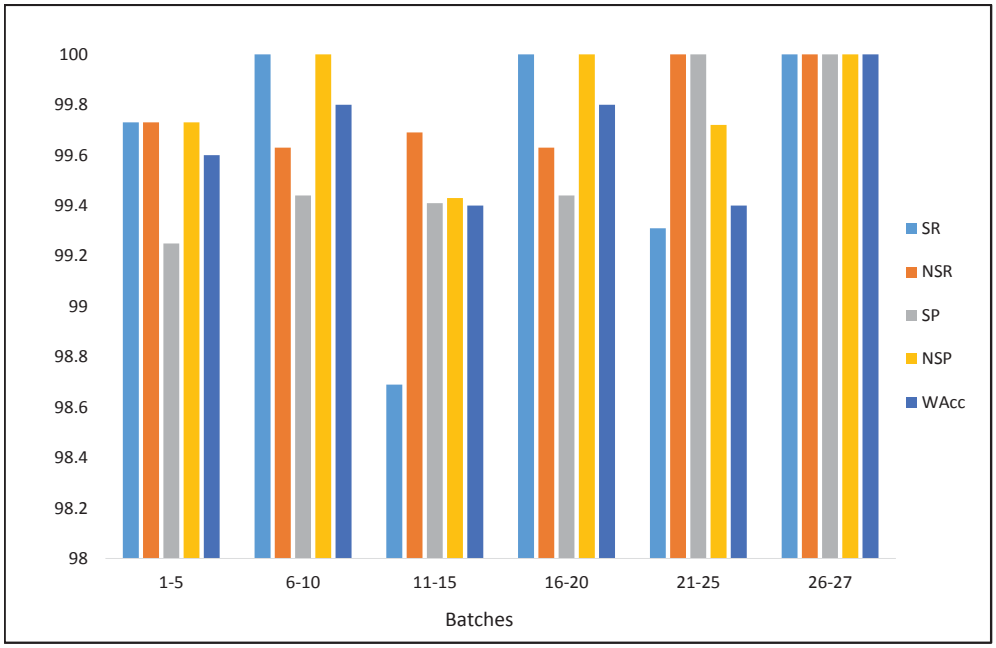


Figure 5.12: Results obtained for each Batch of the Enron-1 dataset

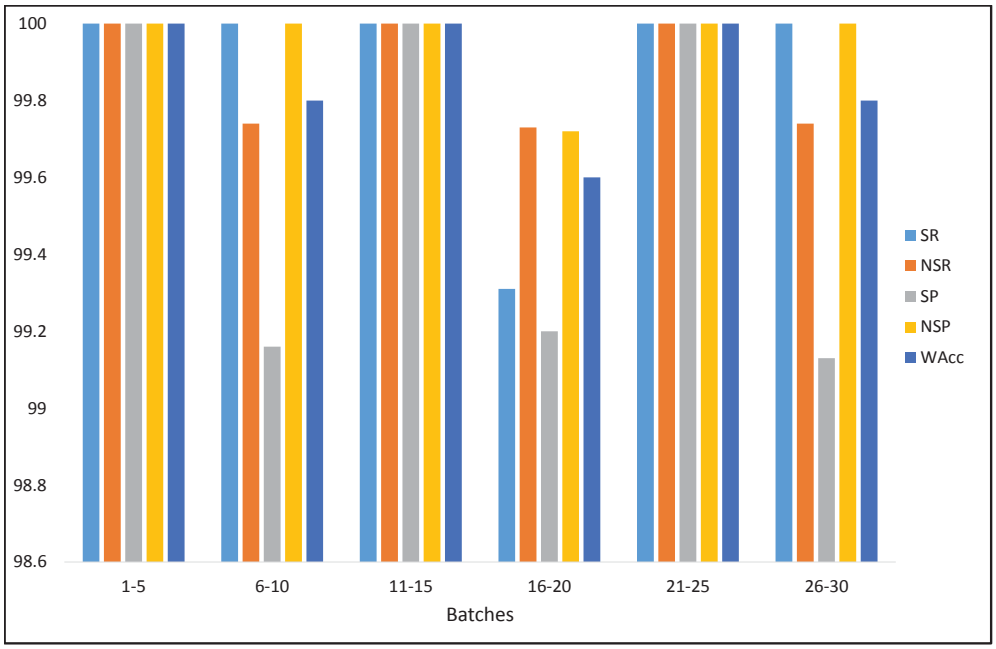


Figure 5.13: Results obtained for each Batch of the Enron-2 dataset

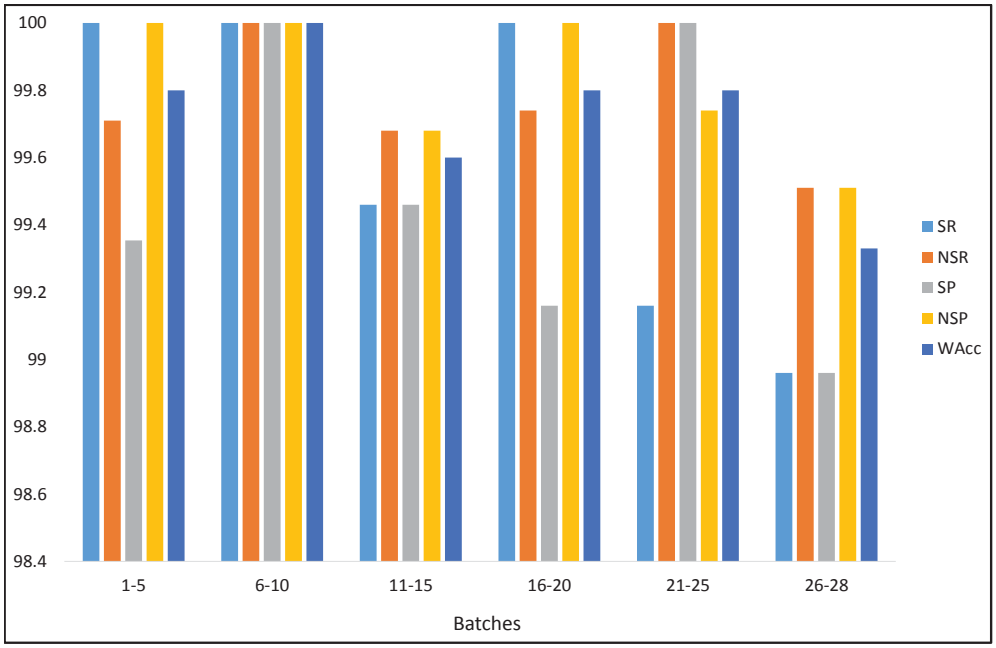


Figure 5.14: Results obtained for each Batch of the Enron-3 dataset

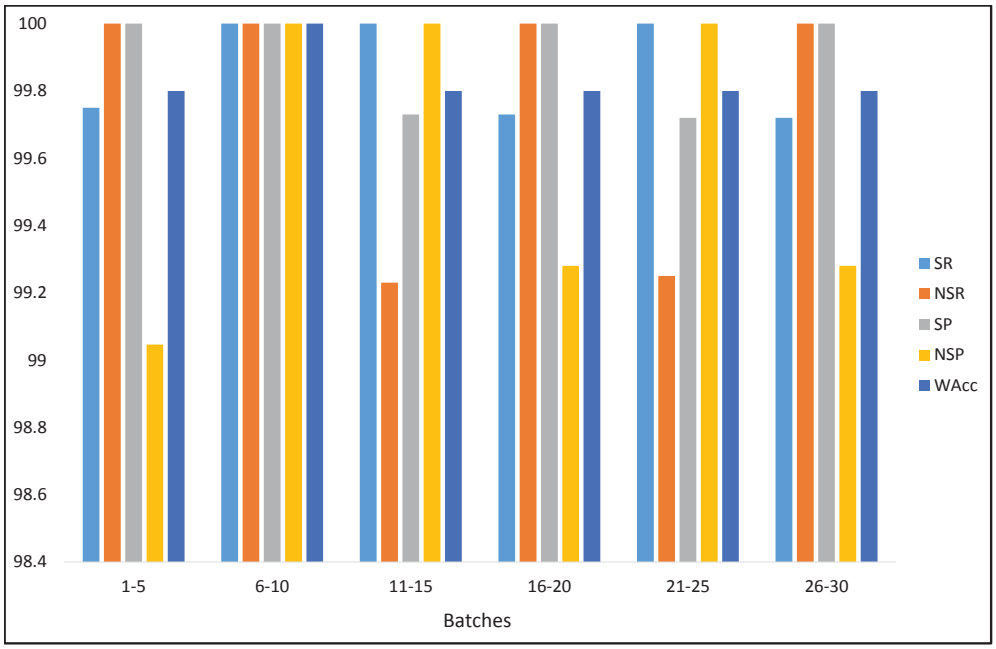


Figure 5.15: Results obtained for each Batch of the Enron-4 dataset

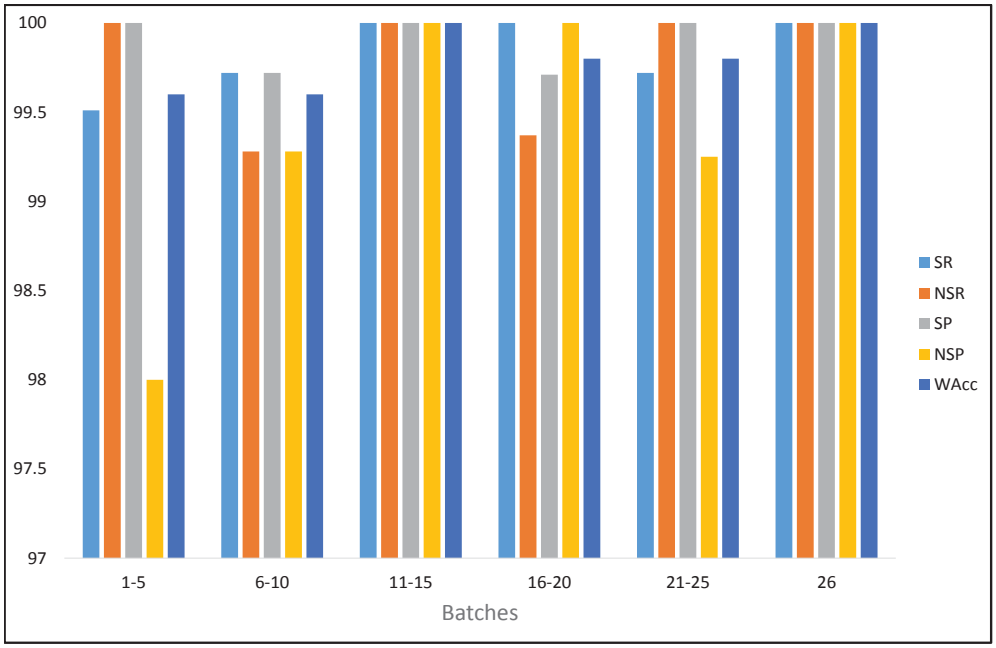


Figure 5.16: Results obtained for each Batch of the Enron-5 dataset

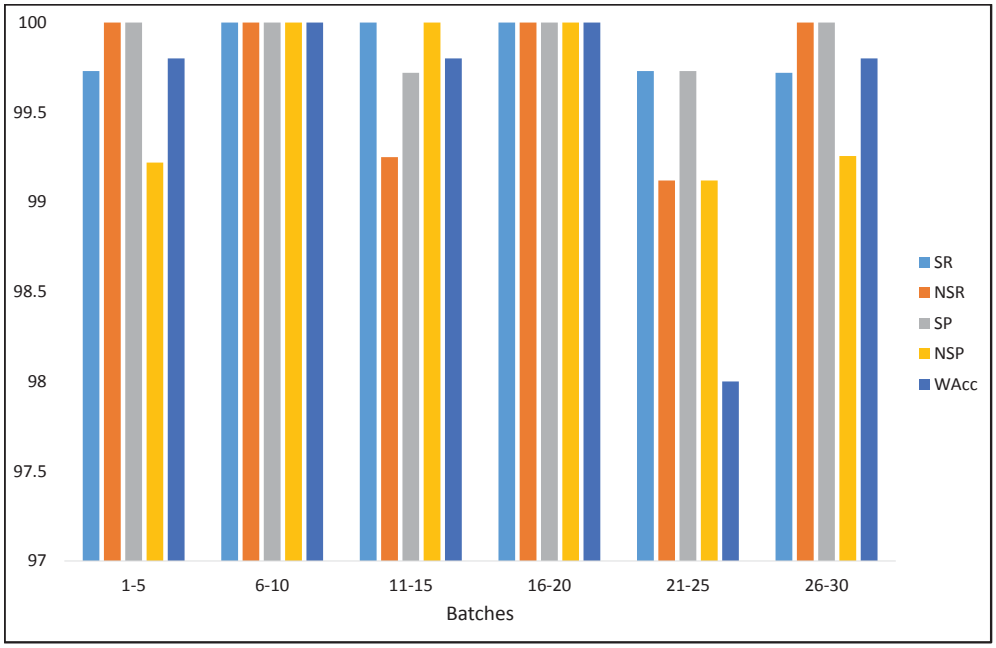


Figure 5.17: Results obtained for each Batch of the Enron-6 dataset

Table 5.6: Results across all Enron datasets

Category	SR(%)	NSR(%)	SP(%)	NSP(%)	WAcc(%)
Enron-1	99.62	99.78	99.59	99.81	99.72
Enron-2	**99.88**	**99.86**	99.58	**99.95**	**99.86**
Enron-3	99.59	99.77	99.48	99.82	99.72
Enron-4	99.86	99.74	**99.90**	99.60	99.83
Enron-5	99.82	99.77	**99.90**	99.42	99.80
Enron-6	99.86	99.72	**99.90**	99.59	99.56

5.4.1.4 Minimum Description Length (MDL) classifier

A compression-based spam classification approach was proposed by Almeida et al. (Almeida and Yamakami, 2012), that uses the Minimum Description Length (MDL) principle (Rissanen, 1978) for spam classification. The MDL principle is a formalization of Occam's Razor in which the best theory can be represented in a most compact form. The model selection is based on the degree of compression achieved and accordingly the model gets selected.

The principle states that if we have a probability distribution of the elements on a finite and countable set T, then there exists a prefix code for an element $x \in T$ such that $L(x) = -\log_2 Pr(x)$, where $L(x)$ is the code length in bits associated with the probability of $x \in T$. From this, it is implied that large probabilities result in small code lengths and vice versa.

Assuming the classes to be $c \in \{\text{spam, non-spam}\}$, the two models are built by extracting the terms from the already classified messages. For a new message m, which is a sequence of k tokens and whose class label is unknown, the probability of each token t_i belonging to a particular class c given by the maximum likelihood estimation is:

$$P_{t_i} = \frac{n_c(t_i) + \frac{1}{|\phi|}}{n_c + 1}$$

where n_c denotes the sum of all the terms that appear in messages belonging to class to c and $|\phi|$ is assumed to be 2^{32} (Almeida and Yamakami, 2012). The total code length for all the terms in the token set t of a new message m belonging a particular class c is calculated as:

$$L_m(c) = \sum_{i=1}^{|k|} - \log_2 \left(\frac{n_c(t_i) + \frac{1}{|\phi|}}{n_c + 1} \right)$$

The class that minimally increases the code-length becomes the classification class for the new message m. Almeida et al. (Almeida and Yamakami, 2012) suggests that if $L_m(spam) > L_m(non\text{-}spam)$, then the new message m is classified as spam, otherwise non-spam. This classification criterion should be the other way round i.e., if $L_m(non\text{-}spam) > L_m(spam)$, then the message m is classified as spam otherwise non-spam. Also the introduction of the factor $\frac{1}{|\phi|}$ in the equation above is unclear and seems that it has been introduced to have non-zero value to the probability for tokens not seen in a particular class of mail. The estimate given by maximum likelihood is $P_{t_i} = \frac{n_c(t_i)}{n_c}$.

5.4.1.5 Comparison of Two-Machine Model approach using MML with the MDL approach

In this section we compare the results generated by the two-machine model approach using MML and the MDL approach proposed by Almeida et al. (Almeida and Yamakami, 2012). Table 5.7 and Table 5.8 show the results computed by the MDL classifier and the results achieved by the MML approach are also mentioned for comparison.

Table 5.7: Comparison of MML and MDL approaches across Enron-1 to Enron-3 datasets

Dataset	**Enron-1**		**Enron-2**		**Enron-3**	
Measures	**MML**	**MDL**	**MML**	**MDL**	**MML**	**MDL**
SR(%)	99.62	94.46	99.88	93.33	99.59	90
NSR(%)	99.78	99.01	99.86	99.77	99.77	**100**
SP(%)	99.59	94.62	99.58	99.28	99.48	**100**
NSP(%)	99.81	98.75	99.95	97.1	99.82	96.4
WAcc(%)	99.72	97.56	99.86	97.61	99.72	97.28

The analysis is done the following way. For each Enron category we infer the misclassification resulted while trying to classify spam messages and while trying to

Table 5.8: Comparison of MML and MDL approaches across Enron-4 to Enron-6 datasets

Dataset	**Enron-4**		**Enron-5**		**Enron-6**	
Measures	**MML**	**MDL**	**MML**	**MDL**	**MML**	**MDL**
SR(%)	99.86	97.00	99.82	99.73	99.86	98.67
NSR(%)	99.74	**100**	99.77	96.00	99.72	92.00
SP(%)	99.90	**100**	99.90	98.39	99.90	97.48
NSP(%)	99.60	94.02	99.42	99.31	99.59	97.56
WAcc(%)	99.83	97.83	99.80	98.65	99.56	97.50

classify the non-spam messages. The overall classification accuracy is obvious from the tables.

An observation into the tables indicates that the MDL classifier produces 100% accurate results in the Enron-3 and Enron-4 categories as far as classifying the non-spam messages are concerned. There is no misclassification with the MDL classifier. Whereas the MML approach achieves 99.72% accuracy, on an average, in classifying the non-spam messages in the same datasets. The average accuracy achieved in Enron-1 dataset by the MML two-machine model approach while classifying the spam messages is 99.71%, whereas in the same category the MDL method results in 96.61%. In classifying the non-spam messages, in the Enron-1 category, the MML method gives 99.69% accuracy and the MDL method gives 96.60% accuracy. In overall accuracy (WAcc%), the MML method performs better than the MDL method.

In Enron-2 category, the MML method results in 99.91% ac uracy when spam messages are classified. The MDL method on the other hand results in 95.22% accuracy. While classifying the non-spam messages, the MML method results in 99.72% accuracy and the MDL method results in 99.43% accuracy. In Enron-3 dataset, the spam classification accuracy achieved by the MML approach is 99.70% and the MDL approach classifies spam messages by resulting in 93.2% accuracy. In Enron-4 dataset again, the classification accuracy of the spam messages resulted by the MML method is 99.74% and the MDL method classifies them at 95.52% rate.

In Enron-5 and Enron-6 datasets also, the MML classifier and the MDL classifier behave similarly. The average spam classification is 99.67% by the MML method and the average spam classification is 98.80% by the MDL method. The average non-spam classification by the MML method is 99.78% and that achieved by the MDL method is 96.76%.

One common benefit with both the methods is that they are not sensitive to thresholds. The setting of the threshold results either in high spam recall or high non-spam recall. One has to compromise at the expense of other. It is not possible to get both the values high. Whereas with the compression based models like the MML and MDL, it is possible to get both values high.

5.4.2 Experimental set-up B

In the experimental set-up B, we have used the Activities of Daily Living (ADL) datasets to construct two-machine PFSM models. The datasets were gathered from the University of California at Irvine (UCI) machine learning repository (Ordonez et al., 2013). In the sections following we give a description of datasets and the experiments done using the datasets.

5.4.2.1 Description of Datasets

This dataset comprises information regarding the Activities of Daily Living (ADL) performed by the two users on daily basis in their own home settings. This dataset is composed by two instances of data, each one corresponding to a different user and summing up to 35 days of fully labelled data. The dataset gives a description of the individuals daily movements in different rooms. Each instance of the dataset is described by three text files, namely: description, sensors events (features), activities of the daily living (labels). Sensor events were recorded using a wireless sensor network and data were labelled manually. For simplicity, the individuals are called as “Person-A” and “Person-B” respectively. The individual settings are described as below:

For "Person-A", the description file leads us to the following information. The number of labelled days for "Person-A" is 14. A 4 room house is considered for the movements. There are in total 10 activities that the person does and the activities include Leaving, Toileting, Showering, Sleeping, Breakfast, Lunch, Dinner, Snack, Spare-Time/TV and Grooming. There are 12 sensors that are activated in different parts of the 4 room house. There are 5 broad categories for the sensors that include Passive Infrared (PIR), Magnetic, Flush, Pressure and Electric. The second file that is the Activities of Daily file gives a description of labelled activities in the form of a table that lists the start time, end time and the name of the activity. The third file that we have used in our experiments gives description in tabular form for the start time and end time of the activity, the location of the activity captured using sensors and the place where it happened. The entries look like the one listed in Table 5.9.

Table 5.9: Person-A Active Daily Living (ADL)

Start time	**End time**	**Location**	**Type**	**Place**
28/11/2011 2:27	28/11/2011 10:18	Bed	Pressure	Bedroom
28/11/2011 10:21	28/11/2011 10:21	Cabinet	Magnetic	Bathroom
28/11/2011 10:21	28/11/2011 10:23	Basin	PIR	Bathroom
28/11/2011 10:23	28/11/2011 10:23	Toilet	Flush	Bathroom
28/11/2011 10:23	28/11/2011 10:32	Shower	PIR	Bathroom
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
11/12/2011 15:41	11/12/2011 15:43	Basin	PIR	Bathroom
11/12/2011 15:43	12/12/2011 0:22	Sear	Pressure	Living
12/12/2011 0:31	12/12/2011 7:22	Bed	Pressure	Bedroom

For "Person-B", the number of labelled days is 21. The movements are captured in a 5 room house. The activities include Leaving, Toileting, Showering, Sleeping, Breakfast, Lunch, Dinner, Snack, Spare-Time/TV and Grooming. Here again, 12 sensors are activated that capture movements. The broad categories of the sensors are the same as used in "Person-A's" home setting. The details of the activities and the place where it happened are mentioned in Table 5.10.

Table 5.10: Person-B Active Daily Living (ADL)

Start time	**End time**	**Location**	**Type**	**Place**
11/11/2012 21:14	12/11/2012 0:21	Seat	Pressure	Living
12/11/2012 0:22	12/11/2012 0:22	DoorL	PIR	Living
12/11/2012 0:23	12/11/2012 0:23	Door	PIR	Kitchen
12/11/2012 0:24	12/11/2012 0:24	Door	PIR	Kitchen
12/11/2012 0:24	12/11/2012 0:24	DoorL	PIR	Living
.	.	.	.	.
.	.	.	.	.
.	.	.	.	
2/12/2012 21:18	2/12/2012 21:18	DoorL	PIR	Living
2/12/2012 21:18	2/12/2012 21:18	DoorL	PIR	Living
2/12/2012 21:19	3/12/2012 1:03	Seat	Pressure	Living

5.4.2.2 Conversion of the ADL datasets for PFSM modelling

The datasets given in the tabular form needs to be converted into a sequence of tokens separated by delimiter symbols. This representation in the form of sequence of tokens makes the dataset ready to be modelled by a PFSM and some useful analysis can be done using the PFSM models. The activity instances in the two datasets corresponding to two users are modelled as two PFSMs.

To do so, we make use of the "Place" column and the "Location" column in Tables 5.9 and 5.10. We assign unique character to each distinguished location in the tables and mark end of sentence (in other words, place a delimiter symbol to the end of token) whenever there is a change in place. Table 5.11 shows the symbols used to identify the locations in the two datasets.

The sequence of tokens for "Person-A" generated from Table 5.9 looks like:

$\{A\#BCDE\#FGHFG\#C\#L\#CD\#...\}$

For "Person-B", the sequence is generated from Table 5.10 and looks like:

$\{AB\#CD\#BAB\#DD\#E\#...\}$

Table 5.11: Assignment of Characters to Locations in the ADL datasets

S. No.	Person-A		Person-B	
	Location	Character	Location	Character
1	Bed	A	Seat	A
2	Cabinet	B	DoorL	B
3	Basin	C	Door	C
4	Toilet	D	DoorB	D
5	Shower	E	Basin	E
6	Fridge	F	Toilet	F
7	Cupboard	G	Bed	G
8	Toaster	H	Fridge	H
9	Cooktop	I	Microwave	I
10	Microwave	J	Shower	J
11	Maindoor	K	Cupboard	K
12	Seat	L	Maindoor	L

5.4.2.3 Building PFSM models

Two PFSM models are built from the sequences generated in the above manner and each one corresponds to each single user. We train the models on nearly half of the labelled datasets as we did in the Enron datasets. The remaining half is used for testing purpose.

For "Person-A" we have 14 days of labelled data. We use the sequence generated from the first 7 days to train the PFSM model for "Person-A. Similarly, we have 21 days of labelled data for Person-B. We use 11 days of training data for "Person-B" and the remaining 10 days are used in testing. The initial PFSMs are the PTAs of the sequences and the PFSMs are inferred using the Simulated Annealing (SA) approach to get reduced to compressed representations. Table 5.12 shows the two-part code length of the initial PFSMs and the inferred PFSMs in bits for the two individuals. We also show the initial number of states in the PTAs. When inferred using SA, the final number of states in the induced PFSMs are also shown.

Table 5.12: Code Lengths and Number of States in the Initial and Inferred PFSMs of the trained PFSM models in the ADL Datasets

Individual	Code Length (bits)		Number of States	
	Initial PFSM (PTA)	Inferred PFSM	Initial PFSM (PTA)	Inferred PFSM
Person-A	1691.02	1143.39	65	32
Person-B	4098.8	3374.84	99	47

5.4.2.4 Prediction of Individuals

A more useful analysis of the PFSM models learnt from the training datasets is done by experimentation. As mentioned above, the ADL datasets consist of 35 days of labelled data. This includes 14 days of labelled for "Person-A" and 21 days of labelled data for "Person-B". We use 50% of the total available datasets for learning the models, the remaining 50% is used for testing purpose. The testing datasets are divided into sequence of transitions corresponding to each day and hence we get 7 days in the test dataset for "Person-A" and 10 days in the test dataset for "Person-B". The sequence of transitions corresponding to each test day in test dataset is input to both the models and the amount of increase in message length is noted down in bits. The model that minimally increases the two part code length is more probable to have generated that sequence.

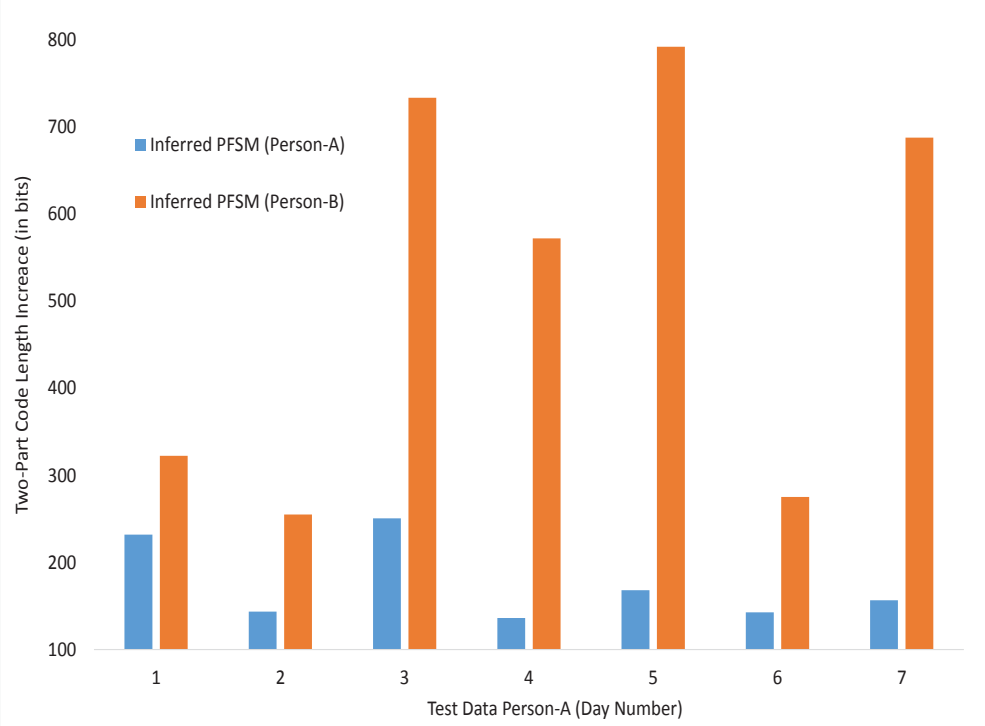


Figure 5.18: Code Length increase in bits for the inferred PFSM models on test data when tested with test data of Person-A

The increase in code length is quantified in Figure 5.18 and Figure 5.19 on the test dataset for each observed day for each of the trained PFSM models. Inferred PFSM (Person-A) model shows a lesser increase in code length for its own test dataset for every test day, whereas the inferred PFSM (Person-B) shows a greater amount of increase in code length for "Person-A's" test data for each test day. This results to arrive at the conclusion that inferred PFSM (Person-A) model correctly predicts "Person-A's" movements for each test day as belonging to inferred PFSM (Person-A) model than what the other model would do for test day of "Person-A". Similarly inferred PFSM (Person-B) model correctly predicts the model from the movements, if the movements belong to "Person-B", by showing a lesser increase in two part code length.

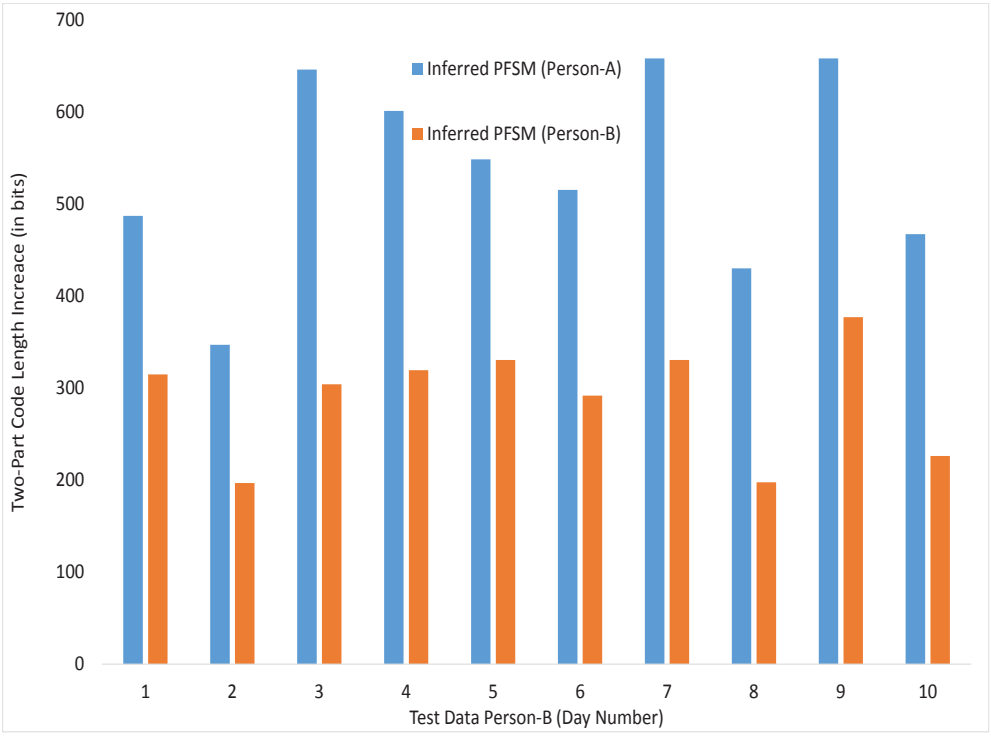


Figure 5.19: Code Length increase in bits for the inferred PFSM models on test data when tested with test data of Person-B

On a concluding note it can be said that individual prediction, given sequence of movements, can be done accurately by the concept of designing the two-machine PFSM models. We did not observe any misclassification when the testing was done with test datasets of the individuals on the inferred PFSM models of both the individuals.

5.5 Summary

We summarize the chapter in this section by highlighting the key concepts that were discussed and the experiments that were performed. The key to this chapter was to learn building the PFSM models from the datasets and consequently using the models that can do classification accurately and also, where the results can be compared with the state of art classifiers.

The word two-machine, in the whole context, is to emphasize the fact that we have two classes of datasets and we are doing learning under a supervised learning environment. We started by considering pseudo examples in the two classes. One class being referred as $Class1$ and the other one referred as $Class2$. The classes were considered as a sequence of tokens separated by delimiter symbols. With this representation of the classes, building the PFSM models was fairly easy. Once the models were constructed from the classes, the models were inferred by the Simulated Annealing (SA) induction process. The details of the SA method can be found in Chapter 4. Now for doing classification of the test dataset whose class label is unknown, the test dataset was input to both the inferred PFSM models that were built from the training dataset of the two classes. The model that resulted in minimal increase in the two-part code length with the test dataset was considered to be more probable of generating the test dataset and this is how the unknown label of the test dataset was found.

To test the theory of doing classification using the two-machine model design, we considered experimentation on the real datasets. Two different experiment situations were considered. In one experimental set-up, we used the Enron spam datasets which internally contains 6 classified spam and non-spam datasets. In the other experimental set-up, we considered the Activities of Daily Living (ADL) datasets that contain the record of the daily activities performed by two individuals in their home settings. In both the experimental set-ups the PFSM models were built on 50% of the total datasets available and testing (classification) was done on the remaining half of the datasets.

In Experimental set-up A, spam and non-spam PFSM models were built by considering 3000 high frequency attributes in both the classes from the training dataset. The test data were evaluated in terms of precision, recall and accuracy. We also compared these measures with the well known Minimum Description Length (MDL) classifier. In almost all the cases, the results generated by the PFSM-MML models were far better than that computed with the MDL classifier. The average

misclassification % seen when trying to classify spam emails across all the Enron datasets with MML approach was 0.28% and that seen with the MDL approach was 3.63%. The average misclassification % seen when trying to classify non-spam emails across all the Enron datasets with MML approach was 0.25% and that seen with the MDL approach was 1.95%. The average overall classification percentages across all the Enron spam datasets by the MML approach was 99.75% and that with the MDL approach was 98.60%.

In Experimental set-up B, the ADL datasets were considered. The sequence of movements into various places in the home were captured as strings by assigning unique characters to different locations in the home. The PFSM models corresponding to two different individuals were learnt and then, based on the models learnt, the prediction of the individuals with a sequence with unknown label were tried. The model learnt for "Person-A" correctly predicted the sequence belonging to "Person-A" when tried on "Person-A's" test dataset by showing a minimal increase in the two-part code length. The same situation was true when "Person-B's" test dataset was predicted with the PFSM model belonging to "Person-B".

The idea of building two models for two classes is not new, but with MML this is our first successful experimentation. The main advantage with the two-machine approach is that it is not sensitive to thresholds and therefore it is possible to get both the precision and recall values high at the same time. In the chapter ahead, we will discuss the single-machine model learning for two classes and classification using the single-machine PFSM model. The advantage of this method over the two-machine method is that the model building is cheaper than the two-machine approach. But in this method there is no other option other than using the threshold value to do the classification and therefore we compromise the classification accuracy. The details will follow in Chapter 6.

Chapter 6

Learning Single-Machine PFSM model

6.1 Introduction

This chapter discusses the method of learning a single-machine PFSM model from the given categories or classes of data. The single-machine model, as the name suggests, is a single model that is learnt from the training dataset of the two classes and has the advantage of being cheaper in terms of bits required to represent the model as compared to the two PFSM models. This chapter discusses the model construction and classification by using the single-machine PFSM model approach.

The single-machine PFSM model is learnt by considering the attributes belonging to the two classes. If required, the feature selection procedure can be applied to reduce the dimensionality of the feature space. As already explained in Chapter 5, while building models on the Enron datasets, this procedure necessarily has to be present. The two-part MML code length of the single PFSM model learnt from the two classes of attributes is computed the same way as computed in the two-machine PFSM model except for the fact that there is an additional information in the states of the single-machine PFSM learnt and this is elaborated later in Section 6.2. This additional information is, each state in the single model is reached by the attributes

seen in both classes and so, there is a binomial distribution case that applies to these states. MML calculation for the binomially distributed states adds to the traditional two-part code length calculation of a PFSM.

To classify a test dataset which is a sequence of tokens and whose class label has to be known, it is input to the single model. The unknown class of the new sequence of tokens is inferred using the probability estimated by MML. The threshold of 0.5 is considered and the probabilities resulted from the single machine model by the input of the test data sequence is compared with this threshold. Threshold value of 0.5 refers to a case where a token is seen equally in the two classes. Probability values above the threshold result in the test dataset being classified into one category, and less than the threshold result in the test dataset being classified into the other category. A value equal to threshold would mean that the test dataset is equally probable of belonging to both classes and for a black and white classification, it can be put into the category where misclassification results in less penalty if we are doing a cost sensitive evaluation.

The experiments were performed on the Enron spam datasets as detailed out in Chapter 5 in Section 5.4.1. We have used the same measures for evaluation as we used in the two-machine model. Since this is a case of threshold sensitive classification, we compared our approach with the other threshold sensitive techniques and obtained good classification results. The description of the datasets and the evaluation measures have already been discussed in Chapter 5 and here we simply focus on the results and comparisons. Parts of this chapter have been presented in *International Conference on Eco-Friendly Computing and Communication Systems, 2016* (Saikrishna et al., 2016).

6.2 Design of the single-machine PFSM model

To describe the construction of the single-machine PFSM model, we again consider the same example as considered in the two-machine model design in Chapter 5.

The following tokens are considered in $Class1$ and $Class2$.

$$Class1 = \{(aab\#)^8(abc\#)^5(abb\#)^2(baa\#)^6\},$$
$$Class2 = \{(aab\#)^1(abc\#)^5(abb\#)^{10}(bab\#)^5\}$$

The single-machine PFSM model is constructed by considering the tokens belonging to the two classes. The model looks like the one in Figure 6.1.

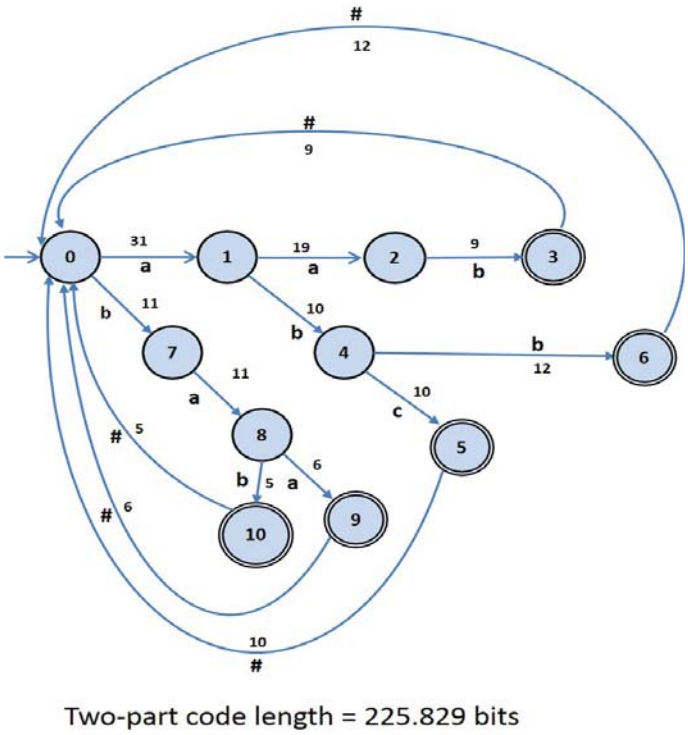


Figure 6.1: Single PFSM model constructed from tokens of *Class1* and *Class2*

Each state in the PFSM of Figure 6.1 is reached by the input alphabets that are seen in both the classes. For example, from PFSM of Figure 6.1, state 3 is reached 8 times by the token *aab* when reading the token from $Class1$ and the same state 3 is reached 1 time when the token is read from $Class2$. Therefore, a binomial distribution case is observed at each state of the PFSM. The MML probabilities for the binomially distributed states are calculated by the computation discussed in Chapter 3 in Section 3.6. The probabilities are encoded and added to the two-part code length of the PFSM structure. The MML probability for State q_i ($1 \leq i \leq S$, where S is number of states) is given by Equation 6.1. The details of the derivation can be found in Chapter 3 in Section 3.6. The two-part code length of the PFSM model of Figure 6.1 is 225.829 bits. The single-machine model is more cheaper than

the two-machine models learnt for $Class1$ tokens and $Class2$ tokens. The number of bits required by the two-machine model is 272.75 bits when we add the two-part code lengths of the inferred PFSM of Figure 5.3 and the inferred PFSM of Figure 5.4.

$$Pr(q_i) = \frac{N_{Class1} + \frac{1}{2}}{N_{Class1} + N_{Class2} + 1} \tag{6.1}$$

In the expression above, the variable N_{Class1} is used to represent the number of times state q_i is reached by reading the input alphabet from $Class1$ category. And similarly, N_{Class2} denotes the number of times state q_i is reached from the symbol in $Class2$ category. The MML probability above assumes a uniform prior (**?**).

Table 6.1 shows the MML probabilities for each such state of PFSM. For further clarity, we also show the number of times the state is reached by the input alphabet in the respective classes.

Table 6.1: MML probability for each state of the PFSM in Figure 6.1

State	**Input Alphabet**	**NClass1**	**NClass2**	**MML probability**
1	a	15	16	0.48
2	a	8	1	0.85
3	b	8	1	0.85
4	b	7	15	0.33
5	c	5	5	0.5
6	b	2	10	0.19
7	b	6	5	0.54
8	a	6	5	0.54
9	a	6	0	0.93
10	b	0	5	0.08

6.3 Classification using the single-machine PFSM model

In the single-machine PFSM model, the state merges cannot be applied as the accepting states are expressive in telling the class probability for a particular token. Every token takes the PFSM to finally reach an accepting state starting from the start state and moving through a sequence of state transitions. In other words, the inference of the PFSM cannot be done. The individual MML probabilities are combined by the Bayesian method of combining multiple evidences and the final probability can be known for the test tokens by that method.

To test a new data with k tokens, the individual probabilities denoted by $Pr(T_i)$, where $1 \leq i \leq k$ and T_i is a token, are combined. Thus if the *CombinedProbability* is greater than 0.5, the test dataset can be categorized as $Class1$, otherwise the test dataset is categorized as $Class2$.

The *CombinedProbability* for k tokens is calculated by the following equation:

$$\frac{(Class1_{instances}/M)Pr(T_1)Pr(T_2)...Pr(T_k)}{(Class1_{instances}/M)Pr(T_1)Pr(T_2)...Pr(T_k)+(Class2_{instances}/M)(1-Pr(T_1))(1-Pr(T_2))...(1-Pr(T_k))} \tag{6.2}$$

In the above formula the occurrence of tokens are considered as independent events. The formula has been derived by using the Bayesian principles of combining multiple evidences that assumes conditional independence (Anderson, 2007). $Class1_{instances}$ denotes the number of instances in $Class1$ category, $Class2_{instances}$ denotes the number of instances in $Class2$ category and M denotes the total number of instances seen in both classes. Therefore $M = Class1_{instances} + Class2_{instances}$. If the number of instances in both classes are assumed equal, then the equation is written in the form as shown below:

$$\frac{Pr(T_1)Pr(T_2)...Pr(T_k)}{Pr(T_1)Pr(T_2)...Pr(T_k) + (1 - Pr(T_1))(1 - Pr(T_2))...(1 - Pr(T_k))} \tag{6.3}$$

To understand how the single-machine PFSM model works, we consider the same target sequence as considered in the two-machine model design. Let the target sequence whose class label is unknown be denoted by the following set.

$Target = \{aab\#abc\#\}$

The target or the test sequence is input to the single-machine PFSM model. The two tokens result in different accepting state numbers, state 3 and state 5. The corresponding MML probabilities for these states from Table 6.1 are 0.85 and 0.5. By the use of Equation 6.3, the *CombinedProbability* for the tokens resulted by the PFSM model of Figure 6.1 is 0.85. The *CombinedProbability* is measured against threshold of 0.5 and since this value is greater than 0.5, the test dataset with sequence of tokens $\{aab\#abc\#\}$ is classified under category $Class1$.

The method of building and classification using the single-machine PFSM model is shown pictorially in Figure 6.2

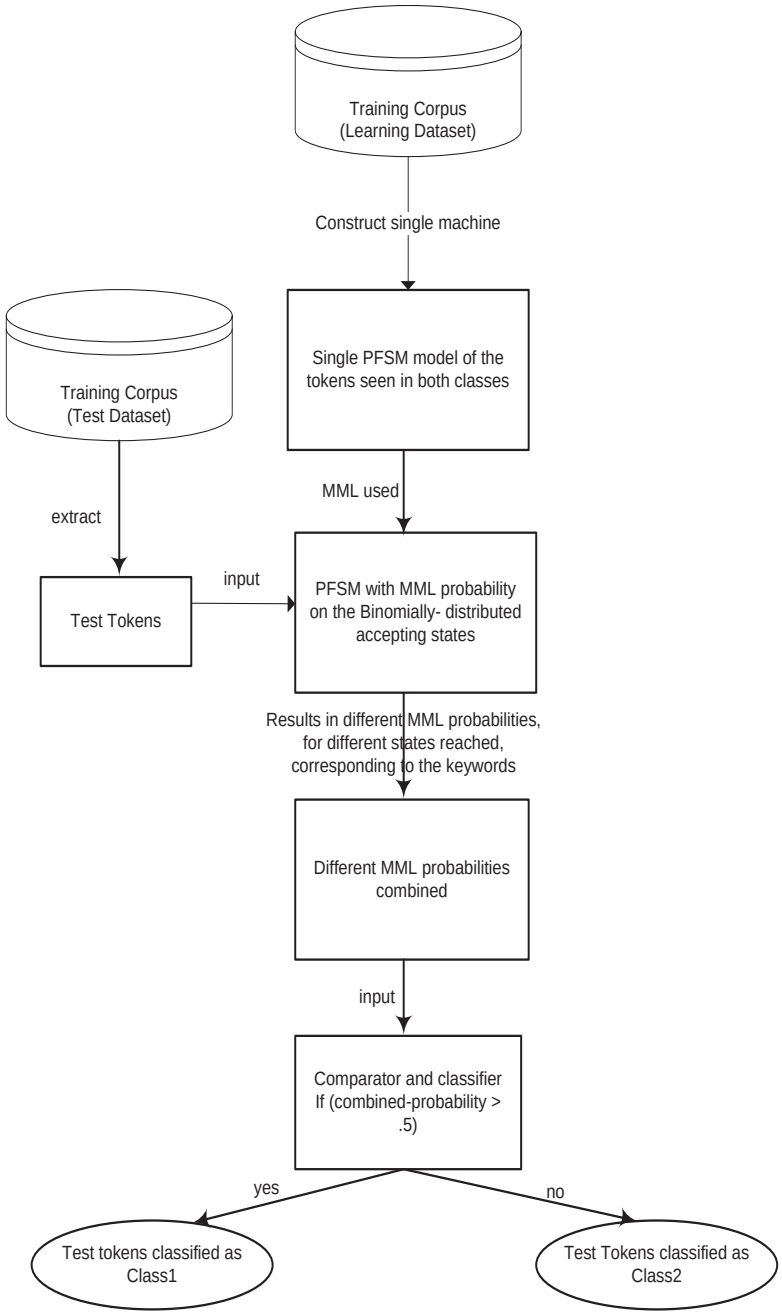


Figure 6.2: Classification using single-machine PFSM model design

In case, the test dataset contains a token or a group of tokens that has not been learnt by the single-machine PFSM model, the token or the group of tokens are added to the model with a transition count of 1 in each class. And, when the MML probability is calculated for the token or the group of tokens, the probability value returned is 0.5. The test data in this case can be put in any class depending on where the inclusion would result in minimum penalty.

6.4 Differences between the Single-Machine PFSM model and the Two-Machine PFSM models

The differences between the single-machine design and two-machine designs are summarized in Table 6.2.

Table 6.2: Differences between the single-machine design and two-machine design

S. No	Differentiating Features	Two-Machine Models	Single-Machine Model
1.	Popularity	The two-machine model design is more popular than the single-machine model design. The idea of building multiple models from the classified categories of the training corpus has been used in the past with success. The models are learnt by constructing PFSM models from the categories of data. The novelty is introduced by the use of MML approach, both in model learning and classification.	The single-machine model on the other hand is less popular than the two-machine model. Here also the model is leant by building PFSM model from categories of data. MML has more role to play in classification than learning the models itself. The model two-part code length is computed in the same way using MML as in the two-machine model design.
2.	Cost of Building Models	The cost of building two-machine models is more than what is required for the single-machine PFSM model. The two PFSMs are inferred and the total cost of constructing models is the sum of the two-part code lengths of the individual inferred PFSMs.	The single-machine model requires less number of bits for model construction. The two-part code length of the model is computed the same way except for one difference. There is additional information in the states of PFSM that also needs to be encoded. The additional information is the MML probability of the Binomially distributed states.
3.	Time required to Infer Models	The two PFSMs are inferred using the Simulated Annealing method by applying state merges. The inferred PFSMs then represent the optimal code length models. The time required to get the minimal PFSMs depends on the initial temperature set and the initial temperature is determined empirically.	The PFSM need not be inferred in the single-machine design as the state merges cannot be applied. This is the reason that models are learnt very fast and are more or less equally efficient in doing classification as the two-machine models.
4.	Threshold Sensitiveness	There is no requirement of using any threshold values to do the classification. It is possible to get both precision and recall rate high at the same time.	The thresholds are required in single-machine design for classification. Incorrect setting of threshold either results in high false positive rate or high false negative rate.
5.	Suitability for multiple classes classification	The idea of constructing multiple models comes from the notion of multiple classes. In practice, it is not always feasible to construct multiple models for multiple classes as each of the class needs to be inferred.	The single-machine model looks for feasible when multiple classes are present. This machine will definitely be cheaper than constructing multiple models for various classes. The accepting states will have a Multinomial distribution and MML probabilities for the same can be known. Only disadvantage with this construction is careful setting of the threshold value.
6.	Classification Accuracy	Theoretically the two-machine model is guaranteed to give good classification results. This is also proven experimentally in the sections further that discusses results.	The classification accuracy attained in the single-machine design is not that pleasing as compared to the two-machine model design, but it still gives satisfactory results that can be compared with other similar approaches.

6.5 Experiments

In this section we compare the results obtained with the single machine design with the various forms of Naive Bayes classification methods. The results are computed on the Enron spam datasets, for which the results computed using the Naive Bayes classification methods are already calculated in the previous works in the references (Almeida and Yamakami, 2012)(Androutsopoulos et al., 2000). We briefly review the methods and show a comparison of results by these methods with our single-machine design.

Before comparing, we first show a comparison of the single-machine model code length with the two-machine model code length. The two-machine model discussed in Chapter 5 learns two PFSM models from the two classes of data. The individual class PFSM model code lengths are added and compared with the code length of the single-machine PFSM model. The model is trained on the same training dataset of the Enron spam datasets. Table 6.3 shows the comparison and we observe that in each category of the Enron spam dataset, the single-machine design always results in a cheaper model than the two-machine model. On an average, the single-machine models for all the Enron categories are 25630 bits cheaper than the corresponding two-machine models.

Figures 6.3 to 6.8 show the results in terms of precision, recall and weighted accuracy for the tested batches of the Enron spam datasets.

Table 6.3: Comparison of Two-Part model code lengths between the Two-Machine and Single-Machine designs

Dataset	**Training Dataset (No. of Messages)**		**Two-Machine Design**			**Single-Machine Design**
	Spam	**Non-Spam**	**Spam Model Code Length**	**Non-Spam Model Codel length**	**Total Code Length**	**Single-Machine Code Length**
Enron-1	750	1836	758418	869953	1628371	1615627
Enron-2	748	2180	692412	751346	1443758	1417643
Enron-3	750	2006	721248	802158	1523406	1484288
Enron-4	2250	750	834624	714673	1549297	1516277
Enron-5	2837	750	897214	744318	1641532	1623562
Enron-6	2250	750	884268	739217	1623485	1598652

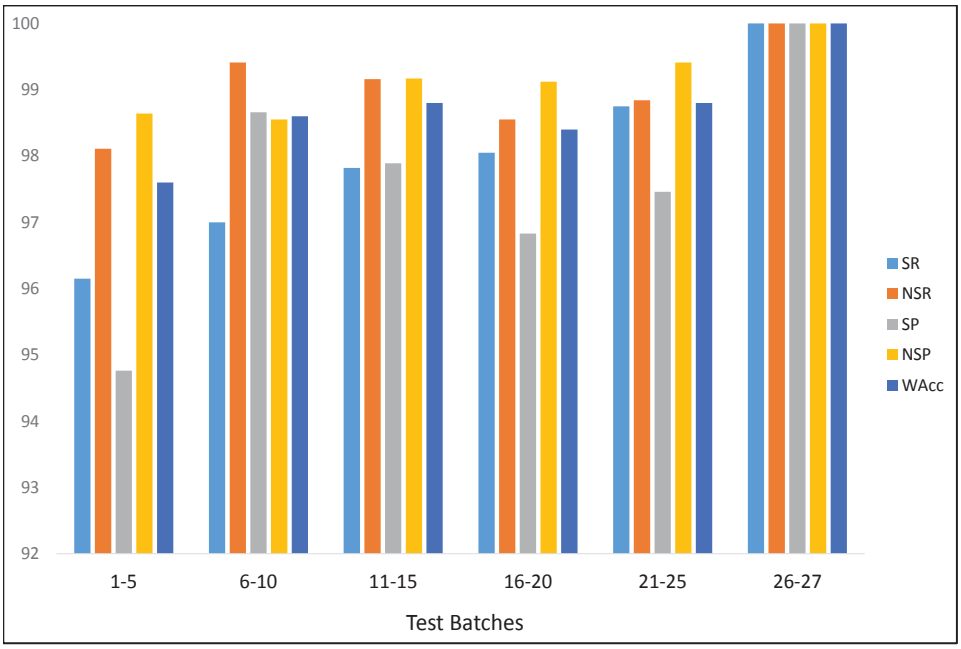


Figure 6.3: Results obtained for each Batch of the Enron-1 dataset using the single-machine PFSM model of Figure 6.1

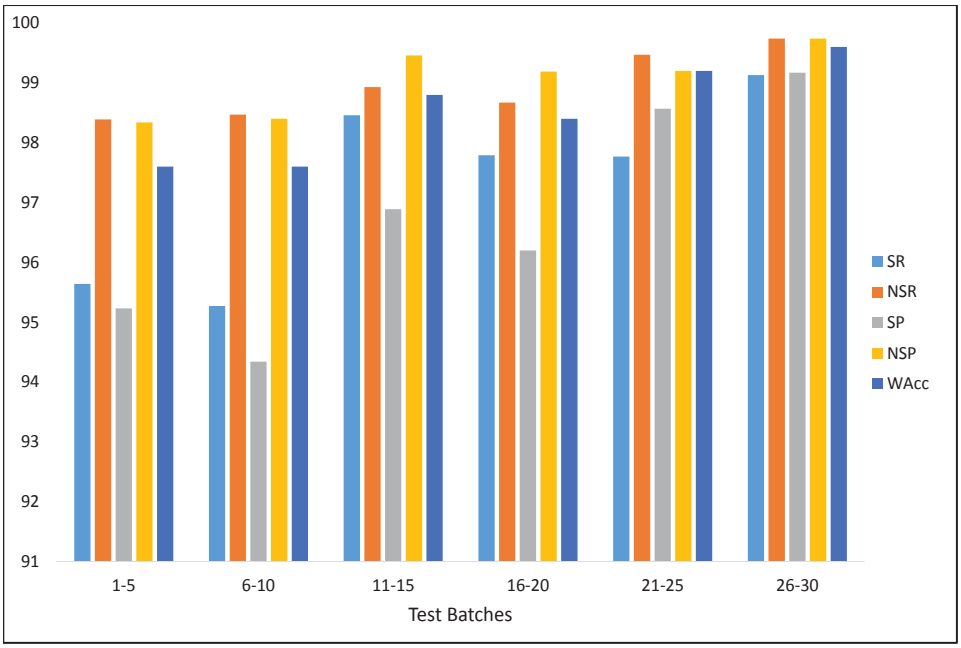


Figure 6.4: Results obtained for each Batch of the Enron-2 dataset using the single-machine PFSM model of Figure 6.1

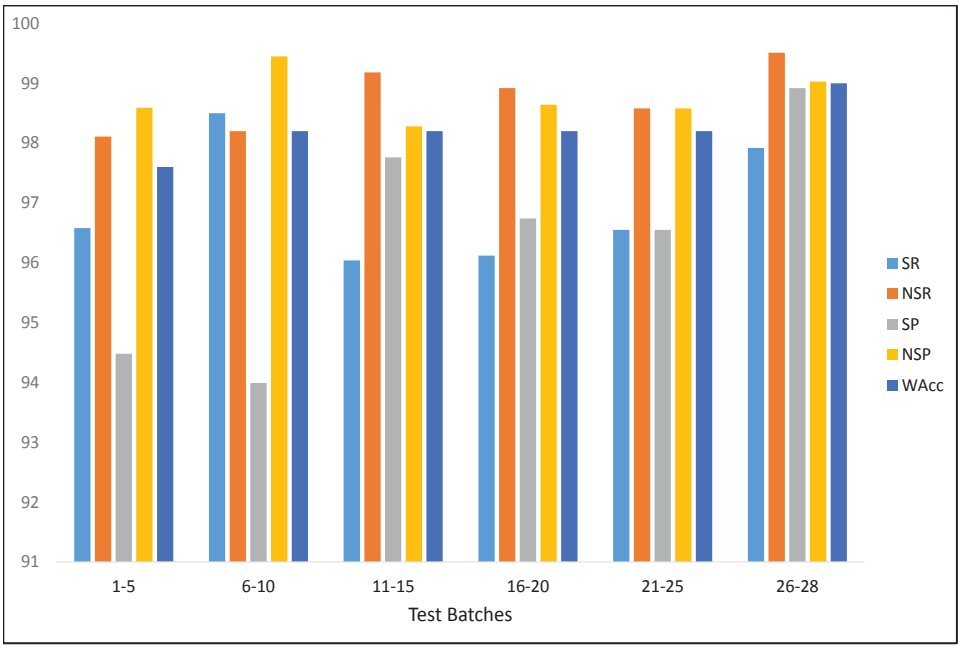


Figure 6.5: Results obtained for each Batch of the Enron-3 dataset using the single-machine PFSM model of Figure 6.1

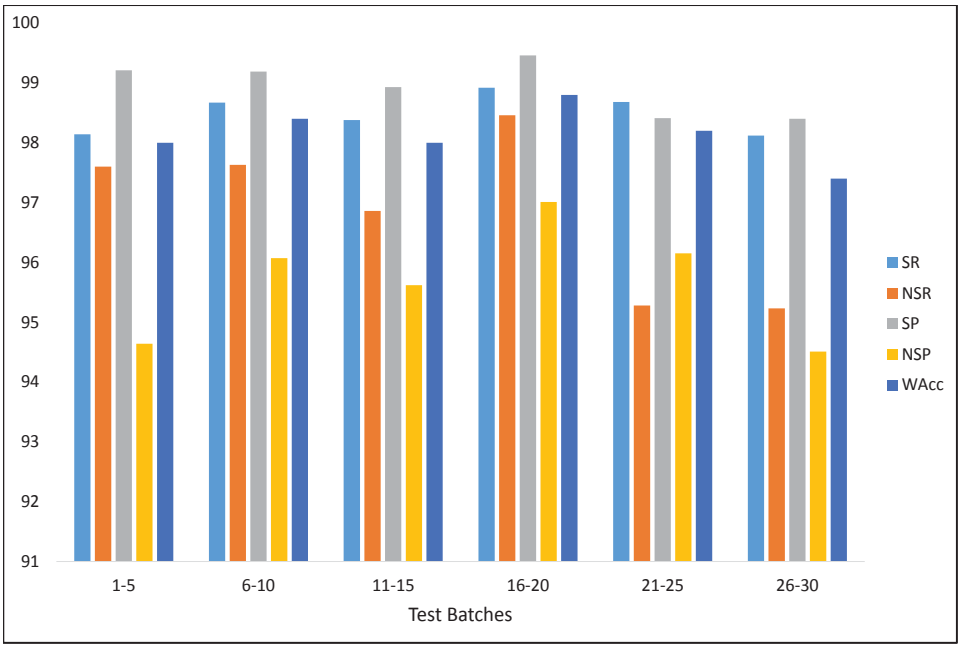


Figure 6.6: Results obtained for each Batch of the Enron-4 dataset using the single-machine PFSM model of Figure 6.1

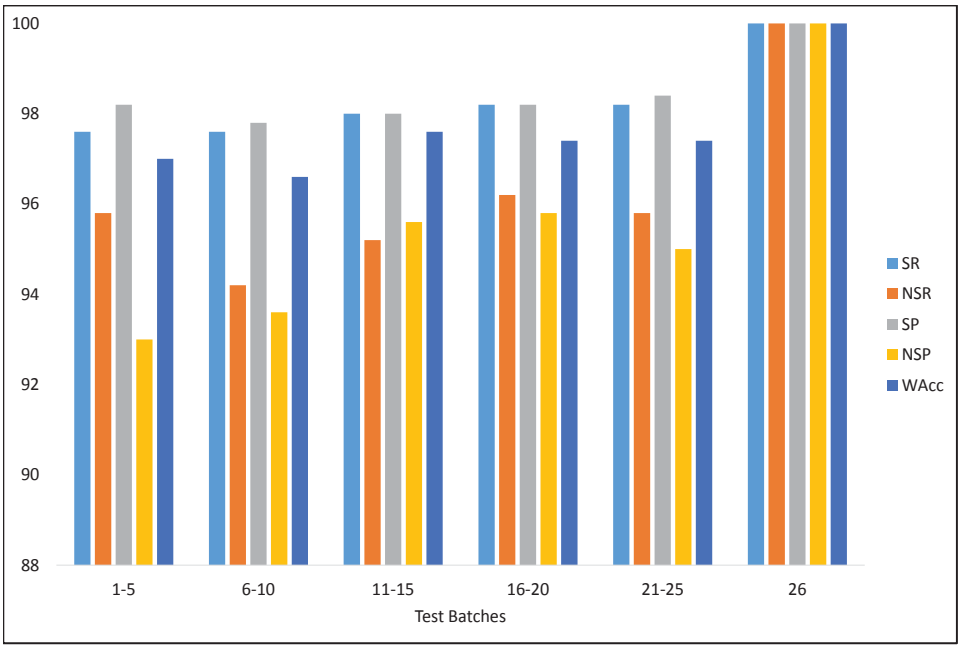


Figure 6.7: Results obtained for each Batch of the Enron-5 dataset using the single-machine PFSM model of Figure 6.1

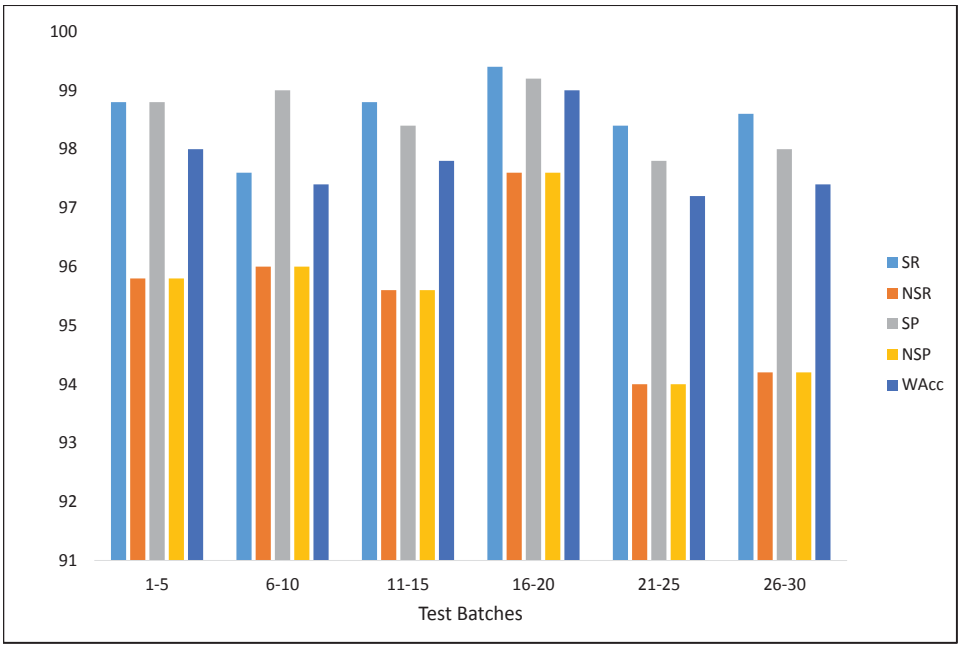


Figure 6.8: Results obtained for each Batch of the Enron-6 dataset using the single-machine PFSM model of Figure 6.1

The results across all the tested batches are averaged and shown in Table 6.4 for the six Enron categories.

Table 6.4: Results across all Enron datasets using Single-Machine model

Category	SR(%)	NSR(%)	SP(%)	NSP(%)	WAcc(%)
Enron-1	97.62	98.90	97.33	99.00	98.52
Enron-2	97.34	**98.95**	96.73	**99.05**	**98.53**
Enron-3	96.88	96.23	96.23	98.74	98.18
Enron-4	97.33	92.00	97.33	92.00	96.00
Enron-5	98.00	95.62	98.19	94.81	97.31
Enron-6	**98.60**	95.53	**98.53**	95.53	97.80

An analysis into Table 6.4 reveals that the average misclassification seen when trying to classify the spam emails across all Enron categories is 2.37%. Average misclassification resulted when classifying the non-spam emails is 3.79%. The overall accuracy % across all Enron categories is 97.72 in the single-machine model. The two-machine model on the other hand, from Table 5.6, resulted in 99.74% average accuracy when summarized for all the Enron categories. The average misclassification rates obtained were 0.28% and 0.25% in the two-machine model, when classifying spam and non-spam emails respectively. This leads to a conclusion that the two-machine model results in better accuracy as compared to the single-machine model. But then we pay more in terms of space and time in the two-machine model. The time factor is attributed to the fact that it takes time to converge the two PFSM models into PFSM-MML models in the two-machine design. Whereas the single-machine model has no such requirements.

We also compare the results generated by the single-machine PFSM model with the different versions of the Naive Bayes classifiers. The various Naive Bayes classifiers are briefly reviewed here.

6.5.1 Naive Bayes Classification methods

The Naive Bayesian methods rely on Bayes's theorem to calculate the probability of a new message with k tokens belonging to either $Class1$ or $Class2$. If the k tokens

are represented by message vector D, then the probability of the message belonging to class $Classj \in \{Class1, Class2\}$ is given by:

$$Pr(Classj|D) = \frac{Pr(Classj).Pr(D|Classj)}{Pr(D)}$$

The message is classified into the class $Classj$ where the product $Pr(Classj)$ $.Pr(D|Classj)$ is maximum, as the denominator is independent of class. A message is classified in $Class1$ category if the following holds true:

$$\frac{Pr(Class1).Pr(D|Class1)}{Pr(Class1).Pr(D|Class1) + Pr(Class2).Pr(D|Class2)} > T$$

where T is a threshold and set to 0.5 value in all experimental evaluations. The probability $Pr(D|Classj)$ is calculated differently in different forms of the Naive Bayesian Classifiers.

6.5.1.1 Basic Naive Bayes

Proposed by Sahami et al. (Sahami et al., 1998), the method of calculating probability $Pr(D|Classj)$ is given by:

$$Pr(D|Classj) = \prod_{i=i}^{k} Pr(T_i|Classj)$$

The probabilities $Pr(T_i|Classj)$ are estimated by:

$$Pr(T_i|Classj) = \frac{M_{T_i,Classj}}{Classj_{instances}}$$

where $M_{T_i,Classj}$ are the number of messages that contain token T_i in $Classj$ category and $Classj_{instances}$ are the number of $Classj$ messages.

6.5.1.2 Multinomial Term Frequency Naive Bayes

The Basic Naive Bayes considers the occurrence of a token in a message only once. In other words, the message D is a Boolean vector. Whereas in the Multinomial Term Frequency Naive Bayes, a message D with k tokens considers the number of occurrences of tokens (McCallum and Nigam, 1998). The occurrence of each token T_i is represented as M_i. The probability follows a multinomial distribution and is given by:

$$Pr(D|Classj) = Pr(D).D!.\prod_{i=1}^{k} \frac{Pr(T_i|Classj)^{M_i}}{M_i!}$$

The probabilities $Pr(T_i|Classj)$ are estimated by Laplacian prior:

$$Pr(T_i|Classj) = \frac{1 + M_{T_i,Classj}}{k + Classj_{instances}}$$

6.5.1.3 Multinomial Boolean Naive Bayes

This method is similar to the above Multinomial Term Frequency Naive Bayes except for the change that each token is Boolean. The above two methods take into consideration the absence of the tokens, but this method does not considers tokens that are not present. The assumption in the Multinomial Term Frequency Naive Bayes is that the attributes follow a Poisson distribution in each category. But in reality, the assumption does not follow and so in that case, working with Boolean attributes gives better results (Schneider, 2004).

6.5.1.4 Multivariate Bernoulli Naive Bayes

In this method, each message D is seen as result of k Bernoulli trials, where at each trial, it is decided whether or nor T_i will appear in D (Losada and Azzopardi, 2008). The probabilities $Pr(D|Classj)$ are computed by:

$$Pr(D|Classj) = \prod_{i=1}^{k} Pr(T_i|Classj)^{M_i}.(1 - Pr(T_i|Classj)^{(1-M_i)}$$

The probabilities $Pr(T_i|Classj)$ are estimated by Laplacian prior:

$$Pr(T_i|Classj) = \frac{1 + M_{T_i,Classj}}{2 + Classj_{instances}}$$

where $M_{T_i,Classj}$ are the number of messages that contain token T_i in $Classj$ category and $Classj_{instances}$ are the number of $Classj$ messages.

6.5.1.5 Boolean Naive Bayes

The Boolean Naive Bayes works similar to Multivariate Bernoulli Naive Bayes with the difference that it does not take into account the absence of terms. The probabilities $Pr(D|Classj)$ are given by:

$$Pr(D|Classj) = \prod_{i=1}^{k} Pr(T_i|Classj)$$

The probabilities $Pr(T_i|Classj)$ are estimated the same way as in Multivariate Bernoulli Naive Bayes.

6.5.1.6 Multivariate Gauss Naive Bayes

Each token T_i in the Multivariate Gauss Naive Bayes is considered as a real-valued attribute that follows a Gaussian distribution $g(T_i; \mu_{i,Classj}, \sigma_{i,Classj})$ for each category $Classj$. The mean $\mu_{i,Classj}$ and the standard deviation $\sigma_{i,Classj}$ are estimated from the training dataset. The probabilities $Pr(D|Classj)$ are computed by:

$$Pr(D|Classj) = \prod_{i=1}^{k} g(T_i; \mu_{i,Classj}, \sigma_{i,Classj})$$

6.5.1.7 Flexible Bayes

The Flexible Bayes works similar to Multivariate Gauss Naive Bayes with the only change that instead of using a single normal distribution for each token T_i, the Flexible Bayes represents the probabilities $Pr(D|Classj)$ as the average of $L_{i,Classj}$ normal distributions with different values of $\mu_{i,Classj}$ (John and Langley, 1995). Therefore the probabilities $Pr(D|Classj)$ are computed by:

$$Pr(D|Classj) = \frac{1}{L_{i,Classj}} \sum_{i=1}^{L_{i,Classj}} g(T_i; \mu_{i,Classj}, \sigma_{i,Classj})$$

$L_{i,Classj}$ is the amount of different values that the token T_i has in the training set of class $Classj$.

6.5.1.8 Support Vector Machines

Support Vector Machines (SVMs) are one of the most powerful techniques employed in text classification (Cormack, 2007)(Drucker et al., 1999). In this method of classification, the data point is viewed as $p - dimensional$ vector and the points are separated with $p - 1$ hyperplanes. Among the different hyperplanes separating the data points, the hyperplane that represents the largest separation or margin, is chosen. The details of the SVM method of classification can be found in references (Hidalgo, 2002)(Kolez and Alspector, 2001)(Sculley and Wachman, 2007).

6.5.2 Comparison of the Single-Machine model with the Naive Bayes Classifiers

We compare the results generated by the single-machine PFSM model with the different versions of the Naive Bayes classifiers. The results generated by the classifiers were computed on 3000 attributes and on the same threshold value of 0.5. Tables 6.5 - 6.10 show the comparison of the single-machine PFSM model (abbreviated as SM in the tables) with the different version of Naive Bayes classifiers.

[1]

Table 6.5: Enron-1 Results achieved by each classifier

Measures	Basic	Bool	MN TF	MN Bool	MV Bern	MV Gauss	Flex Bayes	SVM	SM
SR(%)	91.33	96	82	82.67	72	78.67	87.33	83.33	97.62
NSR(%)	93.48	63.32	88.86	79.35	81.79	95.38	94.29	95.11	98.9
SP(%)	85.09	51.61	75	62	61.71	87.41	86.18	87.41	97.33
NSP(%)	96.36	97.49	92.37	91.82	87.76	91.64	94.81	93.33	99
WAcc(%)	92.86	72.78	86.87	80.31	78.96	90.54	92.28	91.7	98.52

Table 6.6: Enron-2 Results achieved by each classifier

Measures	Basic	Bool	MN TF	MN Bool	MV Bern	MV Gauss	Flex Bayes	SVM	SM
SR(%)	80	95.33	75.33	74	65.33	62.67	68.67	90.67	97.34
NSR(%)	99.31	92.45	**99.8**	99.54	94.97	98.86	99.54	96.8	98.95
SP(%)	97.57	81.25	96.58	**98.23**	81.67	94.95	98.1	90.67	96.73
NSP(%)	93.54	98.3	92.13	91.77	88.87	88.52	90.25	96.8	99.05
WAcc(%)	94.38	93.19	93.2	93.2	87.39	89.61	91.65	95.23	98.53

Table 6.7: Enron-3 Results achieved by each classifier

Measures	Basic	Bool	MN TF	MN Bool	MV Bern	MV Gauss	Flex Bayes	SVM	SM
SR(%)	57.33	99.33	57.33	62	**100**	52.67	52	61.33	96.88
NSR(%)	**100**	99.75	**100**	**100**	93.28	97.76	99.25	98.76	96.23
SP(%)	**100**	99.33	**100**	**100**	84.75	89.77	96.3	96.48	96.23
NSP(%)	86.27	99.75	86.27	87.58	**100**	84.7	84.71	96.83	98.74
WAcc(%)	88.41	**99.64**	88.41	89.67	95.11	85.51	86.41	96.74	98.18

Table 6.8: Enron-4 Results achieved by each classifier

Measures	Basic	Bool	MN TF	MN Bool	MV Bern	MV Gauss	Flex Bayes	SVM	SM
SR(%)	94.67	98	93.78	96.89	98.22	94.44	94.89	**98.89**	97.33
NSR(%)	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	92
SP(%)	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	97.33
NSP(%)	86.21	94.34	84.27	91.46	**94.94**	85.71	86.71	86.77	92
WAcc(%)	96	98.5	95.33	97.67	98.67	95.83	96.17	**99.7**	96

[1]Basic (Basic Naive Bayes), Bool (Boolean Naive Bayes), MN TF (Multinomial Term Frequency), MV Bern (Multivariate Bernoulli), MV Gauss (Multivariate Gaussian), Flex Bayes (Flexible Bayesian), SVM (Support Vector Machines)

Table 6.9: Enron-5 Results achieved by each classifier

Measures	Basic	Bool	MN TF	MN Bool	MV Bern	MV Gauss	Flex Bayes	SVM	SM
SR(%)	89.67	78.23	88.86	94.29	98.1	96.68	88.86	89.4	98
NSR(%)	97.33	**100**	**100**	**100**	80.67	92	97.33	99.33	95.62
SP(%)	98.8	**100**	**100**	**100**	92.56	96.37	98.79	99.7	98.19
NSP(%)	79.35	76.14	78.53	87.72	94.53	73.89	78.7	79.26	94.81
WAcc(%)	91.89	90.93	93.8	95.95	93.5	88.22	91.31	92.28	97.31

Table 6.10: Enron-6 Results achieved by each classifier

Measures	Basic	Bool	MN TF	MN Bool	MV Bern	MV Gauss	Flex Bayes	SVM	SM
SR(%)	86	66.86	76.67	92.89	96.22	92	89.78	89.78	98.6
NSR(%)	97.33	**99.33**	98.67	92	76	85.33	95.33	86.67	95.53
SP(%)	98.98	**99.67**	99.42	97.21	92.32	94.95	98.3	95.28	98.53
NSP(%)	69.86	50	58.5	81.18	87.2	78.5	75.66	73.86	95.53
WAcc(%)	88.33	75	82.17	92.67	91.17	90.33	91.17	90.5	97.8

In the Enron-1 dataset, the single-machine (SM) model gives the best performance in all performance categories. In the Enron-2 dataset, the SM method achieves the best in classifying the spam messages and also achieving the best in the overall classification accuracy measure. The near competitor in terms of overall classification accuracy, is the Support Vector Machine (SVM) classifier in this dataset. The accuracy achieved by the SM classifier is again the best in Enron-5 and Enron-6 datasets. The near competitors in this category are Naive Bayes (Basic), Boolean Naive Bayes (Bool), Multinomial Term Frequency (MN TF) and Multinomial Boolean (MN Bool). In the Enron-4 dataset, the accuracy achieved by the SM method is comparable to the Basic Naive Bayes classifier, whereas the other classifiers did exceptionally better than the SM classifier. On the whole, the SM method did well in most of the cases.

6.6 Summary

The chapter is summarized as follows.

The method of learning single-machine PFSM model is first discussed. Given two classes of data, the model is first trained on the training dataset of the two classes. The attributes or tokens belonging to the two classes are modelled as a Prefix Tree Acceptor (PTA). The method of computing the two-part code length of the PTA of the attributes is done the same way as discussed in Chapter 4 in Section

4.3. The single-machine PFSM model (PTA), as it sees the tokens belonging to two separate classes, the states in the PTA have an additional information contained in them. This information is the number of bits required to encode the binomially distributed states as they are reached by reading the tokens from two different classes. The MML calculation for binomial distribution for the states of the single-machine model applies in addition to the encoding of the other parts information of the single-machine PFSM model. This slightly increases the two-part code length of the single-machine model but, when compared to the added two-part code length of the two-machine model design, the single-machine model takes less number of bits. In other words the single-machine is cheaper than the two-machine model for the same set of training data. This space efficiency of the single-machine design is understood in both datasets (one shown for the artificially created dataset and the other one shown in the publicly available Enron spam datasets).

The method of learning the single-machine PFSM model is followed by classification. The test tokens whose class label is unknown, are input to the single-machine model. The MML probabilities of the test tokens are then combined by the Bayesian method of combining multiple evidences. The combined probability is measured against a threshold. The values greater than the threshold result in the tokens being classified into one class and the values less than threshold classifies the tokens into the other category.

Experiments with real datasets and comparison with the existing methods follow after learning a classifier with the artificial dataset. For experimentation, the single-machine PFSM model is learnt on high frequency 3000 attributes of the spam and non-spam emails in the Enron spam datasets. We show a comparison of the single-machine PFSM model versus the two-machine PFSM model and observe that in every category of the Enron spam datasets, the single-machine model always results in a less code length machine than the two-machine model. The trained model is tested against the test dataset in each category of the Enron spam datasets. A threshold of 0.5 is considered in the experiments. The performance of the model is

evaluated in terms of precision, recall and accuracy. The model performance is then compared with the different threshold sensitive Naive Bayes classifiers. Overally, the single machine model resulted in better results when compared to the other classifiers on the Enron spam dataset.

We also discuss the advantages and disadvantages of the single-machine model over the two-machine model. Although the single-machine is more space efficient than the two machine-model, the criteria for classification is subjected to thresholds. This results either in high false positive rate or high false negative rate. The threshold needs to be optimally chosen. The two-machine model, as it is not dependent on any threshold setting, is capable of resulting in low false positive and low false negative rates at the same time. Also, the accuracy resulted by the two-machine model is more appealing than that resulted by the single-machine model.

In the following chapter, we will discuss a more precise method representing the PFSM models suitable for a particular class of data. That is, we will talk about learning the Hierarchical PFSM (HPFSM) models. We also do an analysis on the HPFSM models learnt in the chapter ahead.

Chapter 7

Learning Hierarchical PFSM (HPFSM) model

7.1 Introduction

The conventional method of learning non-hierarchical PFSMs or simply PFSMs is extended to the case of learning Hierarchical PFSMs or HPFSMs in this chapter and is the crux of the complete research work. The HPFSMs represent the behaviour of PFSMs more concisely by identifying the inherent hierarchy in the data that is a sequence of tokens. This chapter introduces a method of learning HPFSM models from classes of data and the HPFSM models learnt are used in doing some meaningful and useful analysis of the data.

We discuss in this chapter the method of encoding HPFSMs by extending the traditional coding scheme that uses the MML principle. To explain the coding scheme, we consider two different artificially created HPFSM models. The coding scheme is in line with the two-part encoding of a PFSM model as discussed in Chapter 4. The first part encodes the HPFSM model and the second part encodes the data generated by the model. The two-part MML code length is the sum of the code lengths of the first part and the second part of the HPFSM model. From the artificially created HPFSM model, random strings or data tokens are generated

and we compare the cost of encoding the HPFSM model for those tokens with the traditional PFSM model.

We then discuss the experiments performed on the UCI gathered, Activities of Daily Living (ADL) datasets. As discussed before in Chapter 5, the dataset is a collection of daily activities performed by two individuals in their home settings that are captured through sensors. We discuss a method of encoding the dataset in the form of an HPFSM model. The initial models learnt are then inferred using Simulated Annealing and we do an analysis by using those models. The analysis is the prediction of individuals using the models. We do training on nearly half of the datasets and use the other half for testing purpose. Here we use accuracy of prediction as the evaluating criteria for the models.

The model two-part code length is compared with the traditional PFSM model and also with the one-state model. The results indicate that the HPFSM model gives the best compression when compared to the other models.

7.2 Hierarchical Probabilistic Finite State Machine (HPFSM)

The research work done earlier that encodes a hypothesized PFSM, considers the PFSM to be structurally non-hierarchical. Various induction methods using MML have been proposed and they all aim at resulting in a minimal PFSM which is again structurally non-hierarchical. But, a careful observation at the data observations would reveal that, some of the subsets can be generated by small internal PFSMs without the need to get generated out of the big single PFSM (non-hierarchical). And the small PFSMs are tied together in the outer structure resulting in some kind of hierarchy.

This view-point is extremely essential to understand the things happening in ground reality. If we consider an example, where we have a set of observations and the observations are the words belonging to different languages. In those sets

of observations if we try constructing a PFSM that best describes those sets of observations, the code length would be enormous as it has to include the vocabularies of all the languages. Also, if the observations result in heavy use of one language than the other, then there are infrequent transitions from one language to the other language. So constructing a single non-hierarchical PFSM would turn out to be more expensive in this case. We can do this in a cheaper way by constructing a hierarchical PFSM for the set of observations. For enormous enumerations of the data sequence, the cost of encoding the hypothesis H and the data D generated by the hypothesis would be cheaper than the cost of encoding a non-hierarchical PFSM. But, on the other hand, if we have very less enumerations (which is quite improbable), we would be unnecessarily paying for the complex hierarchical structure.

Another example why coding in hierarchy would turn out to be beneficial could be understood this way. Let us say we have a big machine that represents our movements in a day, in the form of state transitions, from one place to another . Broadly speaking, the places that we may visit are, "university or work place", "home" and "city". The machine representing this situation is seen in Figure 7.1. We start from a local place say home. Home represents one small internal PFSM. At "home" we do state transitions locally more often by visiting different places at "home". There is one point of entry (start state) to "home" and the same point can be used as the exit point from this "home" local machine. Then we transit from "home" to another internal PFSM, say "university". We enter into the start state of the "university" local PFSM and again we perform state transitions my moving around different places in the "university". The start state of the "university" internal PFSM is also the exit state from it, as we had in the "home" internal PFSM. If we are to specify a record of the daily movements that involves moving around various places, in the form of code length in bits, then considering the whole picture as a non-hierarchical PFSM and thereafter calculating the code length would turn to be very expensive. Whereas, if we do the same through hierarchical coding

mechanism, we would definitely get a cheaper encoding of the daily movements of the person.

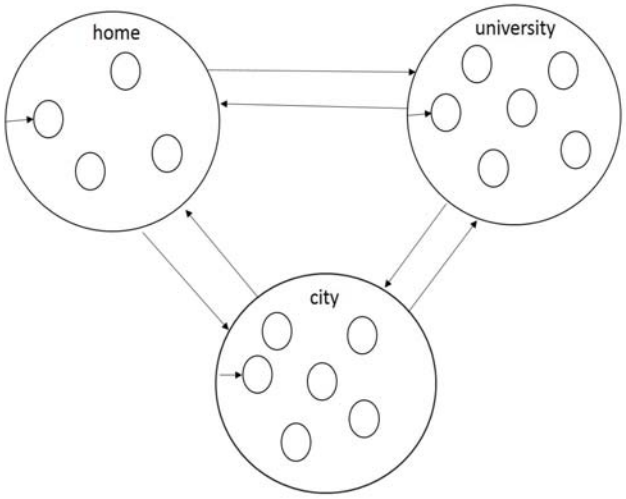


Figure 7.1: An Example of Hierarchical PFSM

7.2.1 Defining an HPFSM

A hierarchical PFSM consists of an outer PFSM whose states can internally contain inner PFSMs (or inner HPSMs). We refer to the hierarchical PFSM as HPFSM to keep it short. An example of such an HPFSM is shown in Figure 7.2. The probability of transition on a particular symbol from any state in the HPFSM of Figure 7.2 is not shown but from every state the transitions on the different symbols are seen equally probable. That, a multinomial distribution with a uniform prior is considered from each state on all symbols.

The behaviour of a simple PFSM model can be understood in a very concise manner in terms of an HPFSM and the obvious benefit is a less two-part code length machine that still represents the same grammar. We refer to the HPFSM model in Figure 7.2, which is a special case, where there are three outer states and each of the outer PFSMs internally contains PFSMs with three states inside, but in general, we can have as many outer and inner states as we may like in the HPFSM model. The structure is explained like this. The HPFSM has three states in the outer structure labelled $S1$, $S2$ and $S3$. The outer states internally contain

PFSMs with three states in each PFSM. The outer PFSM has an initial state $S1$ and this is the string generation point in the HPFSM. Each internally contained PFSM has a starting state and the same state is used as an exit state to transit to other PFSMs in the outer structure. Each internally contained PFSM can independently generate strings separated by the delimiter symbol # or it can communicate with the other PFSMs through the outer transitions and can generate strings that include the outer symbols. Like, for example, the kind of strings produced by the HPFSM of Figure 7.2 can look like $D = \{AB\#BAC\#\#C\#DBC\#BC\#A\#\#...\}$. Each internally contained PFSM has its own input alphabet symbols and similarly the outer structure also has one.

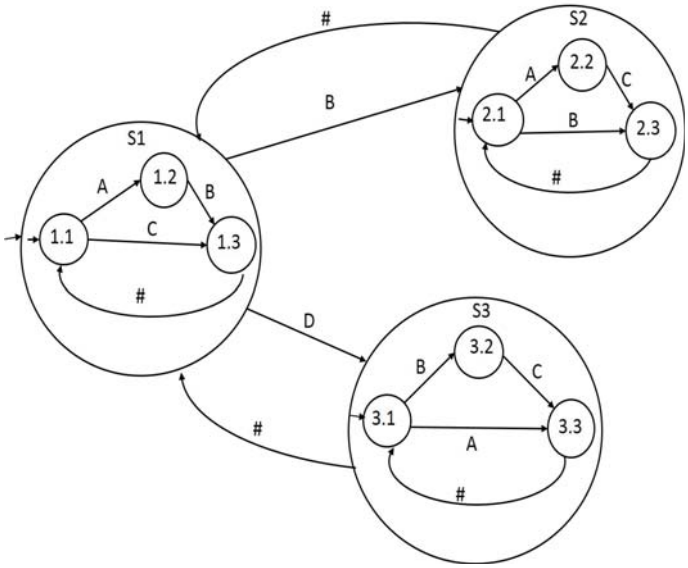


Figure 7.2: Hierarchical PFSM

7.2.2 MML assertion code for the hypothesis H of HPFSM

Before starting with the two-part code length calculation for the HPFSM, we refer to the following definitions:

- S_{outer} is the number of states in the outer PFSM.
- V_{outer} is the cardinality of the input alphabet set in the outer PFSM including the delimiter symbol #.
- The states in the outer PFSM are labelled as $S_1, S_2, ..., S_{S_{outer}}$.

- The number of states in the inner PFSMs are denoted as $S_{internal_1}, S_{internal_2}$, ..., $S_{internal_{S_{outer}}}$.

- The cardinalities of the input alphabets in the inner PFSMs are denoted as $V_{internal_1}, V_{internal_2}, ..., V_{internal_{S_{outer}}}$.

- The number of arcs leaving any state in internal PFSM Sj is denoted by $a_{s_{ji}}$, where $1 \leq j \leq S_{outer}$ and $1 \leq i \leq S_{internal_j}$.

- The number of transitions on symbol k from any current state i in any internal PFSM S_j is denoted as n_{ji_k}, where k can be a symbol from the input alphabet set of S_j or the input alphabet set of outer PFSM.

The coding scheme described below is explained in reference to $S1$ internal PFSM. The other PFSMs are encoded in a similar fashion.

1. The code begins with the number of states in internal PFSM, which is $S_{internal_1}$, and the code length is calculated as $\log_2 S_{internal_1}$ bits.

2. For each state in the inner PFSM S_1, the number of arcs leaving the state are coded. The number of arcs leaving any state depends on the cardinality of the input alphabet set. Since state 1 of internal PFSM S_1 has outgoing arcs to internal PFSM $S1$ and also outgoing arcs to other states in the outer structure, therefore a selection from 1 to $V_{outer} + V_{internal_1}$ is made giving a code length of $\log_2(V_{outer} + V_{internal_1})$ bits. We calculate the combined cardinality denoted by the expression $V_{outer} + V_{internal_1}$ by considering unique input alphabets in the two alphabet sets. So, if # appears in the input alphabet set of the both the internal and the outer structure, it is counted as 1 and likewise any other character other than #.

3. For each state, the labels on the arcs are coded. This is done by making a selection of $a_{s_{j1}}$ symbols from the set of $(V_{outer} + V_{internal_1})$ symbols giving a code length of $\log_2 \binom{(V_{outer}+V_{internal_1})}{a_{s_{j1}}}$ bits, if the state is the starting state of the

internal PFSM. Otherwise for the other states, the code length is $\log_2 V_{internal_1}$ bits. Here $1 \leq j \leq S_{outer}$.

4. The code for the destination states is calculated from each state. From the starting state, the destination can be any of the internal states of that PFSM or it can be of the states in outer structure. In the outer structure the destination is always the initial state of other internal PFSMs. Therefore for the starting state, the code length encoding the destinations is $a_{s_{j1}} \log_2(S_{outer} + S_{internal_1})$ bits. For states other than the starting state, the coding can be done in $\log_2 S_{outer}$ bits. For the arcs labelled with # delimiter symbol the destination is already known and the coding can be done in $(a_{s_{j1}} - 1) \log_2 S_{internal_1}$ bits.

The coding scheme above generates Table 7.1 which shows encoding of the first part code length of the $S1$ internal PFSM. The other internal PFSMs are encoded similarly and are shown in Table 7.2 and Table 7.3.

Table 7.1: Code Length of internal PFSM S1 from Figure 7.2

State	$a_{s_{j1}}$	*Cost*	*Label*(s)	*Cost*	*Dest.*(s)	*Cost*
1.1	4	$\log_2(V_{outer} + V_{internal_1})$	(A, B, C, D)	$\log_2 \binom{V_{outer}+V_{internal_1}}{4}$	(1.2, 1.3, S2, S3)	$4\log_2(S_{outer} + S_{internal_1})$
1.2	1	$\log_2 V_{internal_1}$	(B)	$\log_2 \binom{V_{internal_1}}{1}$	(1.3)	$\log_2 S_{internal_1}$
1.3	1	$\log_2 V_{internal_1}$	$(\#)$	$\log_2 \binom{V_{internal_1}}{1}$	(1.1)	0

Table 7.2: Code Length of internal PFSM S2 from Figure 7.2

State	$a_{s_{j2}}$	*Cost*	*Label*(s)	*Cost*	*Dest.*(s)	*Cost*
2.1	3	$\log_2(V_{outer} + V_{internal_2})$	$(A, B, \#)$	$\log_2 \binom{V_{outer}+V_{internal_2}}{3}$	(2.2, 2.3, S1)	$2\log_2(S_{outer} + S_{internal_2})$
2.2	1	$\log_2 V_{internal_2}$	(C)	$\log_2 \binom{V_{internal_2}}{1}$	(2.3)	$\log_2 S_{internal_2}$
2.3	1	$\log_2 V_{internal_2}$	$(\#)$	$\log_2 \binom{V_{internal_2}}{1}$	(2.1)	0

Table 7.3: Code Length of internal PFSM S3 from Figure 7.2

State	$a_{s_{j3}}$	*Cost*	*Label*(s)	*Cost*	*Dest.*(s)	*Cost*
3.1	3	$\log_2(V_{outer} + V_{internal_3})$	$(B, A, \#)$	$\log_2 \binom{V_{outer}+V_{internal_3}}{3}$	(3.2, 3.3, S1)	$2\log_2(S_{outer} + S_{internal_3})$
3.2	1	$\log_2 V_{internal_3}$	(C)	$\log_2 \binom{V_{internal_3}}{1}$	(3.3)	$\log_2 S_{internal_3}$
3.3	1	$\log_2 V_{internal_3}$	$(\#)$	$\log_2 \binom{V_{internal_3}}{1}$	(3.1)	0

The final equation that encodes the first part code length of the HPFSM is given in Equation 7.1. The start state of each internally contained PFSM is handled

separately in the equation as it has transitions to outer states and that is the reason the variable i starts with 2 in the equation.

$$
\begin{aligned}
CodeLength(H) = \sum_{j=1}^{S_{outer}} \Bigg(& \sum_{i=2}^{S_{internal_j}} \log_2 \binom{V_{internal_j}}{a_{s_{ji}}} + \log_2 \binom{V_{outer} + V_{internal_j}}{a_{s_{ji}}} \\
& + \ (S_{internal_j} - 1) \log_2 V_{internal_j} + \log_2 (V_{outer} + V_{internal_j}) \\
& + \sum_{i=2}^{S_{internal_j}} a_{s_{ji}} \log_2 S_{internal_j} \ + a_{s_{j1}} \log_2 (S_{outer} + S_{internal_j}) \\
& + \log_2 S_{internal_j} \Bigg) + \log_2 S_{outer}
\end{aligned} \tag{7.1}
$$

7.2.3 Encoding the transitions of HPFSM

Again assuming a uniform prior over a multinomial distribution case, the probability of transition on any symbol k from current state i in any internal PFSM Sj, is denoted as $\frac{(n_{ji_k}+1)}{(n_{ji}+a_{s_{ji}})}$, where n_{ji_k} represents the number of transitions from current state i in current internal PFSM Sj on symbol k and n_{ji} represents the total number of transitions on all symbols from the current state. The number of bits required to code the transitions on symbol k is negative logarithm of the transition probability. If we sum over the probabilities for all the symbols from the current state, then the total number of bits required to encode the transitions is given by the following equation:

$$
\frac{(n_{ji} + a_{s_{ji}})!}{\Pi_k (n_{ji_k})!} \tag{7.2}
$$

where $1 \leq j \leq S_{outer}$, $1 \leq i \leq S_{internal_j}$ and k is a symbol from the input alphabet sets of $S_{internal_j}$ and S_{outer}.

Equation 7.3 calculates the second part code length of the HPFSM in Figure 7.2.

$$CodeLength(D|H) = \sum_{j=1}^{S_{outer}} \left(\sum_{i=1}^{S_{internal_j}} \log_2 \frac{(n_{ji} + a_{s_{ji}})!}{\Pi_k n_{ji_k}!} \right) \tag{7.3}$$

The complete two-part code length for the HPFSM is calculated by adding Equation 7.1 and Equation 7.3

7.3 Experiments

The experiments are performed on the artificial datasets generated by the HPFSM of Figure 7.2 and on the UCI gathered Activities of Daily Living (ADL) datasets. In both the experimental situations, we compare the cost of the hierarchical model with the non-hierarchical model and the one-state model. In the experiments performed with the ADL datasets, we first describe a method of learning the initial HPFSM models from the datasets of the individuals. This learning is followed by induction using the Simulated Annealing (SA) search to obtain the optimal HPFSM models. We then finally perform a prediction of individuals based on the sequence of movements from the HPFSM models learnt.

7.3.1 Experiments on Artificial datasets

7.3.1.1 Example-1

We use the HPFSM of Figure 7.2 to generate random data strings or tokens of variable string lengths. As a recap, the HPFSM in Figure 7.2 has three outer states and each of the three outer states has a internal vocabulary of four characters including the delimiter symbol #. The outer PFSM has an input alphabet size of three characters. The strings are generated by setting up initial transition probabilities on all the transition arcs of the HPFSM model. The probabilities for this data-generating process have been set, in this first example, to be uniform in nature. For example, let us consider the state 1.1 in the HPFSM model. State 1.1 has 4 number of

transitions to other states. That is, on characters A and C, states 1.2 and 1.3 are reached respectively from state 1.1 in the same internal PFSM $S1$. On characters B and D, outer states $S2$ and $S3$ are reached from state 1.1. If all the possibilities are considered equally probable in the process of string generation, then all the transition arcs are initially set to probabilities of 0.5 from state 1.1. Similarly the probabilities on other transition arcs are set in the HPFSM model and strings get generated eventually. This constitutes an artificial dataset that is hierarchical as it is generated from the hierarchical structure. The smallest number of strings that we generate are 5 and going upto a maximum of 5000 strings. The string length of the strings that get generated from the HPFSM model are usually 1 or 2. This is because, each internal PFSM has three states internally

and the final states in the internal PFSMs either read single character from the start state or 2 characters from the start state of the internal PFSMs.

Figures 7.3-7.5 show a comparison of the two-part code lengths computed using MML between the various models. We do a comparison of the HPFSM model with the initial non-hierarchical PFSM that is simply shown as PFSM (PTA) in the figures. The PFSM (PTA) models are inferred and what we obtain is the inferred PFSM model. The inferred PFSM model is results in shorter two-part code length model than the PFSM (PTA) model, as expected. We also compare with the one-state model.

Figure 7.3 shows the comparison on number of random strings from 2 in number to 80 strings. For the number of random strings from 2-20 in Figure 7.3, the one-state PFSM model gives the best compression when compared to the other models. The HPFSM model on the other hand results in the highest two-part code length for such less number of strings. This is obvious as the structure encoding is expensive for such a small number of strings. But as the number of strings further increases, the HPFSM model starts showing the least two-part code length when compared to other models.

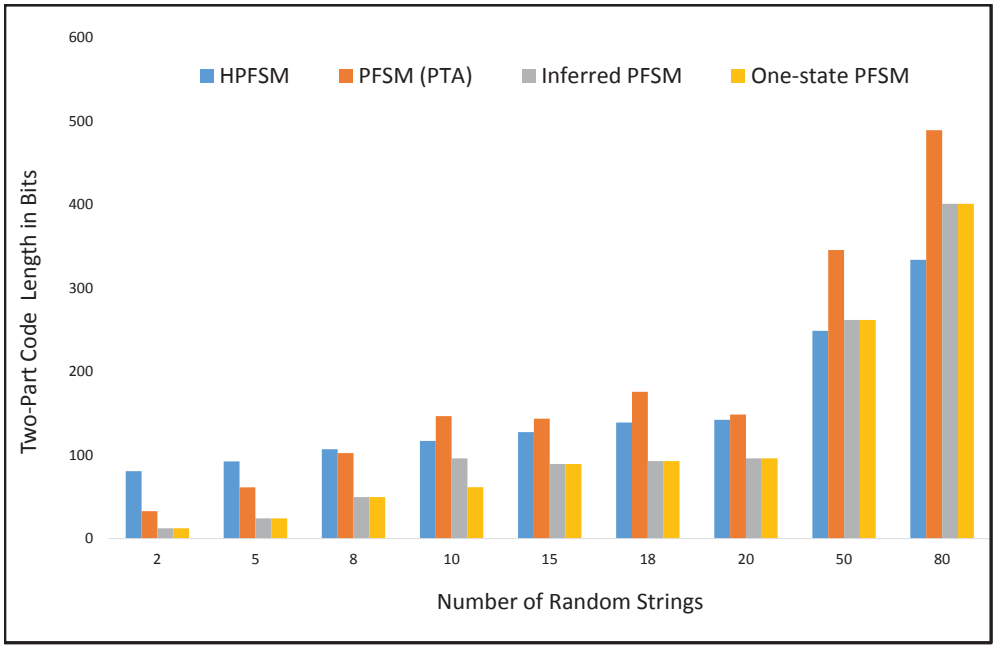


Figure 7.3: Code Length comparison between HPFSM, PFSM (PTA), inferred PFSM and one-state PFSM for random strings 2-80

Figure 7.4 shows the comparison on random strings from 110-800 and Figure 7.5 on random strings from 900-5000. The two figures show a common trend where the HPFSM model shows the best compression and the one-state model shows the worst compression for the dataset comprising the random strings in all the cases. The amount of compression achieved by the HPFSM model for 5000 number of random strings is 28.54% more than the inferred PFSM model. Whereas the same HPFSM model shows 43.57% more compression than the one-state model for 5000 number of random strings. We calculate these percentages by calculating the difference in the two-part code length code length of the HPFSM model with the other models in comparison. The difference in code lengths is then divided by the two-part code length of the compared model and shown as percentage.

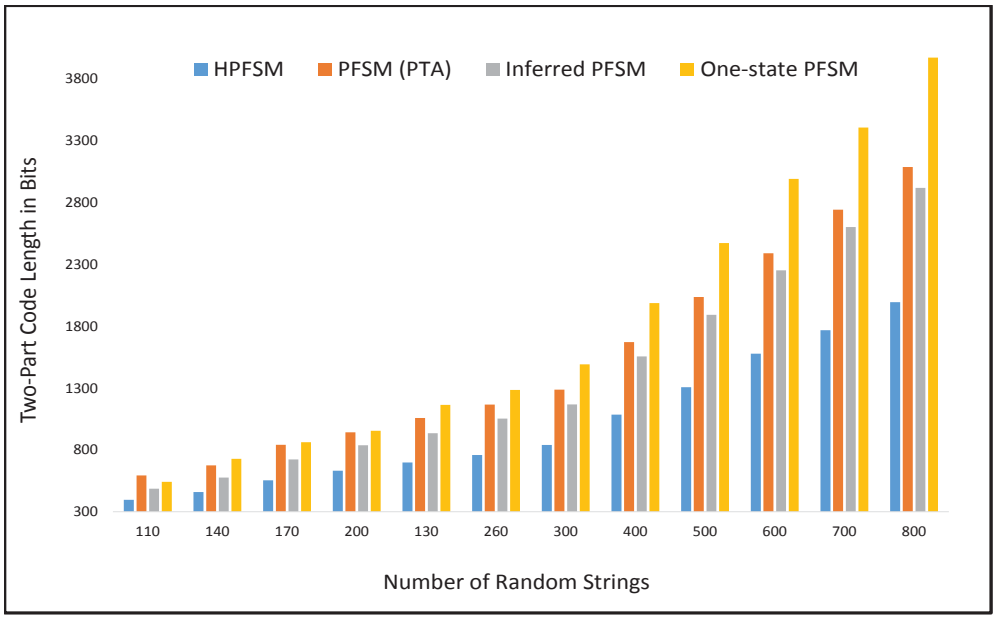


Figure 7.4: Code Length comparison between HPFSM, PFSM (PTA), inferred PFSM and one-state PFSM for random strings 110-800

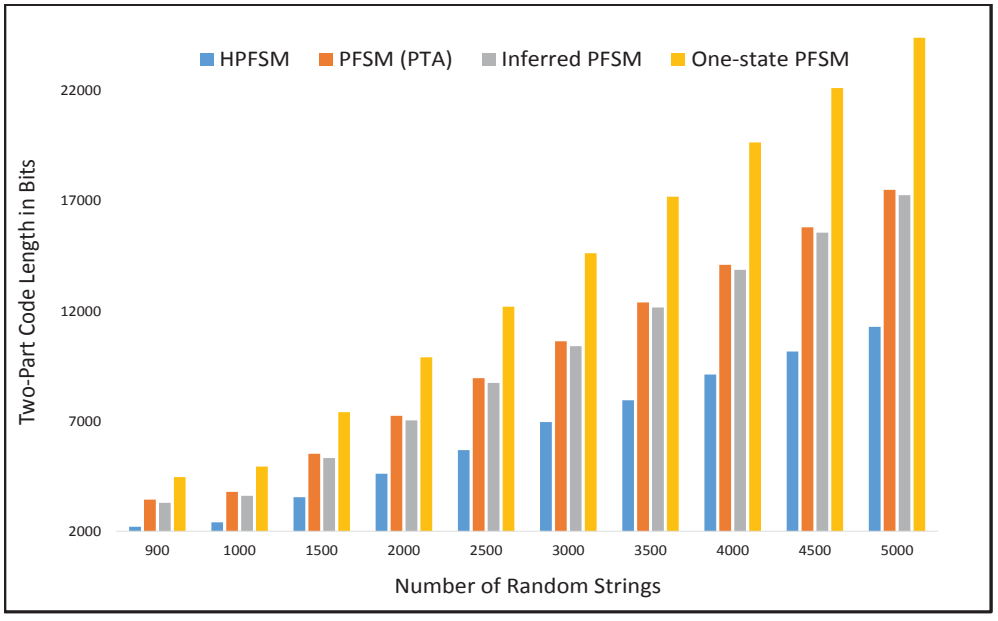


Figure 7.5: Code Length comparison between HPFSM, PFSM (PTA), inferred PFSM and one-state PFSM for random strings 900-5000

7.3.1.2 Example-2

We consider another example of an HPFSM as shown in Figure 7.6. In this example, the outer structure of the HPFSM model has two outer states and the two outer states internally contain 5 and 4 states respectively in their inner PFSMs. The internal vocabulary of the two internal PFSMs are the same. This time, instead of setting uniform prior probabilities in the transition arcs, we set non uniform probabilities. The reason for doing so comes from the motivation that made us think about the idea of HPFSMs. We set to high, the probabilities of transition in the internal arcs and likewise, in the outer transitions the probability values are set to low value. This is because the internal transitions are more frequent than the outer transitions. Variable number of random strings get generated out of the HPFSM model in Figure 7.6 and we do a similar analysis of the model two-part code length for the random strings generated. The two-part code length of the HPFSM model is then compared against the inferred PFSM and the one-state models.

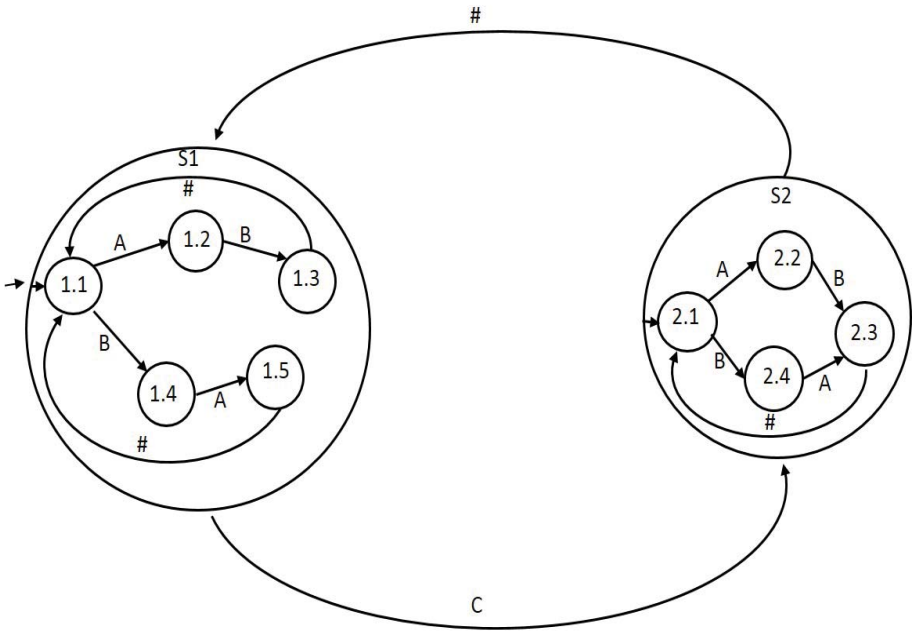


Figure 7.6: Hierarchical PFSM

Figures 7.7 and 7.8 show the comparison of the two-part code lengths of the HPFSM model with other models.

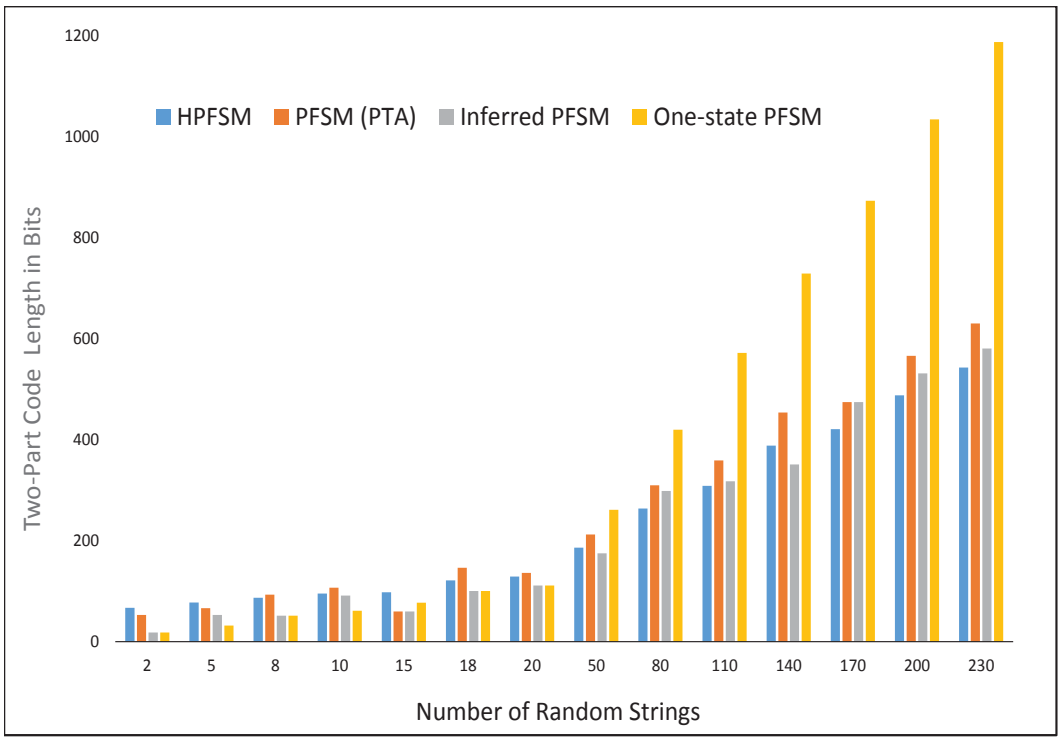


Figure 7.7: Code Length comparison between HPFSM, PFSM (PTA), inferred PFSM and one-state PFSM models for random strings 2-230 for the PFSM of Figure 7.6

In Figure 7.7, we show a comparison on the number of random strings from 2 to 230. The HPFSM model shows the largest two-part code length for the number of random strings less than 8. This is understandable as the model is fairly complex in structure for such small number of strings. The one-state model on the other hand shows the least two-part code length for such small number of random strings. But as the number of strings further increase, the trend completely reverses. The HPFSM model gives the least two-part code length and the one-state model results in the largest two-part code length model.

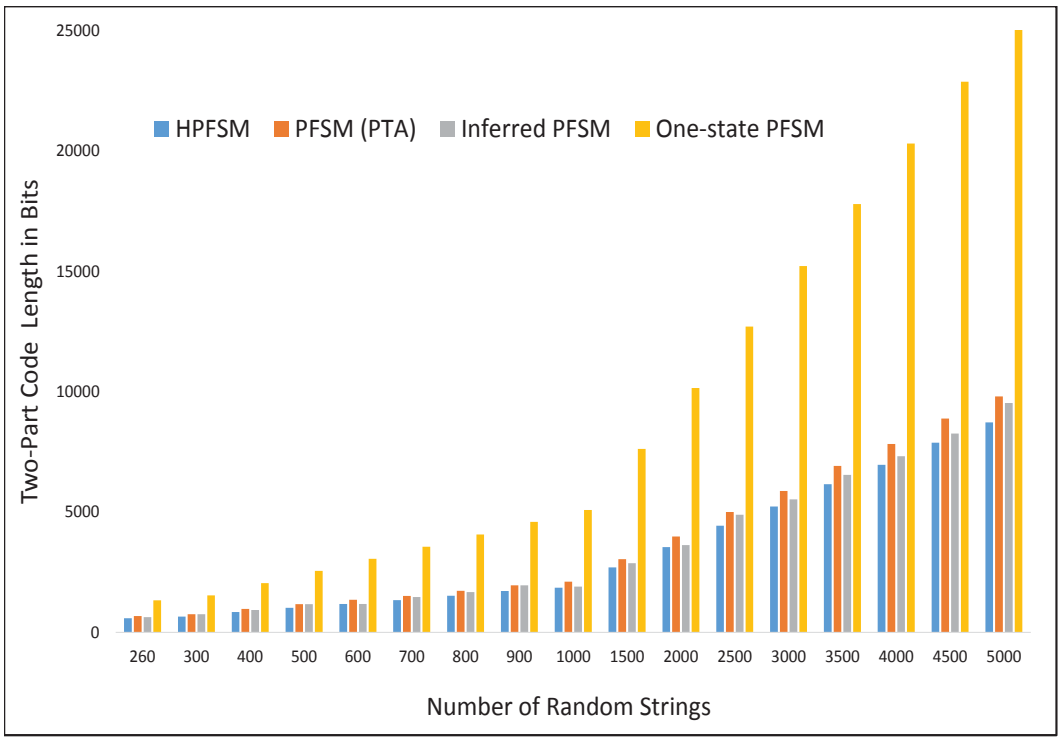


Figure 7.8: Code Length comparison between HPFSM, PFSM (PTA), inferred PFSM and one-state PFSM models for random strings 260-5000 for the PFSM of Figure 7.6

In Figure 7.8, the comparison is shown on the number of random strings from 260 to 5000 and a similar observation follows. The two-part code length in bits obtained from the HPFSM model for 5000 number of random strings is 8728.45 bits and for the inferred PFSM model, the two-part code length is 9534.76 bits. The one-state model on the other hand results in 25421.8 bits for the same dataset. Therefore, for 5000 number of random strings, the probability of the HPFSM model of having generated those strings as compared to the inferred PFSM model is quantified as $\frac{2^{9534.76}}{(2^{9534.76}+2^{8728.45})}$. Aso, the HPFSM model is $\frac{2^{25421.8}}{(2^{25421.8}+2^{8728.45})}$ more probable than the one-state model in generating the dataset of 5000 strings.

7.3.2 Experiments on ADL datasets

The results are also computed using the Activities of Daily Living (ADL) datasets gathered from the UCI Machine Learning Repository (see Table 7.4). A short version of the dataset is presented in Table 7.4. A more detailed version can be seen in Chapter 5 in Section 5.4.2.1. This dataset comprises information regarding the ADL performed by the two users on daily basis in their own home settings and summing up to 35 days of fully labelled data (Ordonez et al., 2013). We call the individuals "Person-A" and "Person-B". Each individual dataset gives a description of the start time and end time of the event, the location of the event captured using sensors and the place where it happened. The five different places (Bathroom, Kitchen, Bedroom, Living and Entrance) are initially considered as five outer states in the HPFSM, each of which has its own internal PFSM. The sequence of transitions is captured as strings and encoded using the HPFSM. The conversion of the sequence of transitions into strings is done by assigning unique symbols to distinct locations in Table 7.4 and the change of place in the table is noted down as the end of sentence. This change of place inserts a delimiter symbol into the sequence formed so far. The HPFSM is inferred and we show as an example in Figure 7.9, the inferred HPFSM model for "Person-A" with three outer states. Three outer states (Bedroom, Living and Entrance) get merged, resulting, finally in three outer states, as seen in the HPFSM of Figure 7.9. We similarly learn the HPFSM model for "Person-B" and the inferred HPFSM model for "Person-B" is shown in Figure 7.10

Table 7.4: Person-A Activities of Daily Living (ADL)

Start time	**End time**	**Location**	**Type**	**Place**
28/11/2011 2:27	28/11/2011 10:18	Bed	Pressure	Bedroom
28/11/2011 10:21	28/11/2011 10:21	Cabinet	Magnetic	Bathroom
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
12/12/2011 0:31	12/12/2011 7:22	Bed	Pressure	Bedroom

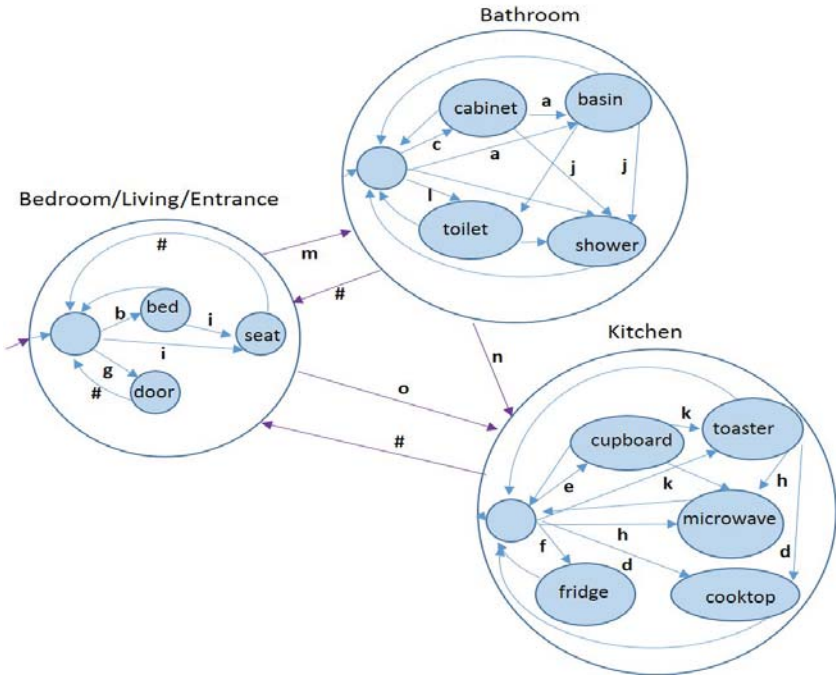


Figure 7.9: Inferred HPFSM for Person-A

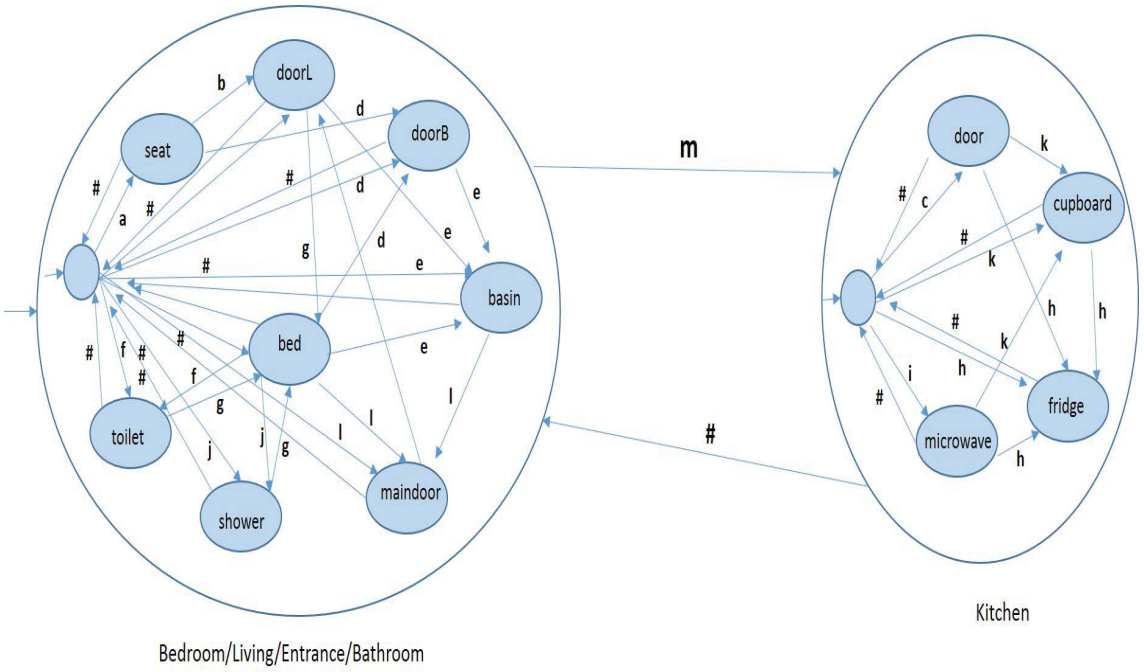


Figure 7.10: Inferred HPFSM for Person-B

Table 7.5 compares the two-part code length of the HPFSM and the non-hierarchical PFSM. The table reveals that, if the tokens generated by the activities of individuals, are encoded using HPFSM, the cost of encoding is less. This is evident from the two-part code length of the HPFSM and simple PFSM (or simply PFSM or non-hierarchical PFSM). We also compare with the one-state PFSM code length.

Table 7.5: Code length in bits given by various models for the ADL of individuals

	Initial HPFSM	**Inferred HPFSM**	**Initial PFSM (PTA)**	**Inferred PFSM**	**1-state PFSM**
Person-A	1235.10	1095.70	1691.02	1143.39	1143.39
Person-B	3062.52	3022.64	4098.80	3374.84	4036.83

A more useful analysis of the HPFSM models learnt from the training datasets is done and that is, prediction of the individuals. As mentioned above, the ADL datasets consist of 35 days of labelled data. This includes 14 days of labelled for "Person-A" and 21 days of labelled data for "Person-B". We use 50% of the total available datasets for learning the models, the remaining 50% is used for testing purpose. The testing datasets are divided into sequence of transitions corresponding to each day and hence we get 7 days in the test dataset for "Person-A" and 10 days in the test dataset for "Person-B". The sequence of transitions corresponding to each test day in test dataset is input to both the models and the amount of increase in code length is noted down in bits. The model that minimally increases the code length is more probable to have generated that sequence.

The increase in code length is quantified in the Figures 7.11 and Figure 7.12 on each HPFSM model on the test dataset belonging to the two individuals for each observed day. Inferred HPFSM (Person-A) model shows a lesser increase in code length for its own test data of every test day, whereas the inferred HPFSM (Person-B) shows a greater amount of increase in code length for "Person-A's" test data of each test day. This results to arrive at the conclusion that inferred HPFSM (Person-A) model correctly predicts "Person-A's" movements for each test day as belonging to inferred HPFSM (Person-A) model, than what the other model would do for test day of "Person-A". Similarly inferred HPFSM (Person-B) model correctly predicts the model from the movements, if the movements belong to "Person-B", by showing a lesser increase in two-part code length.

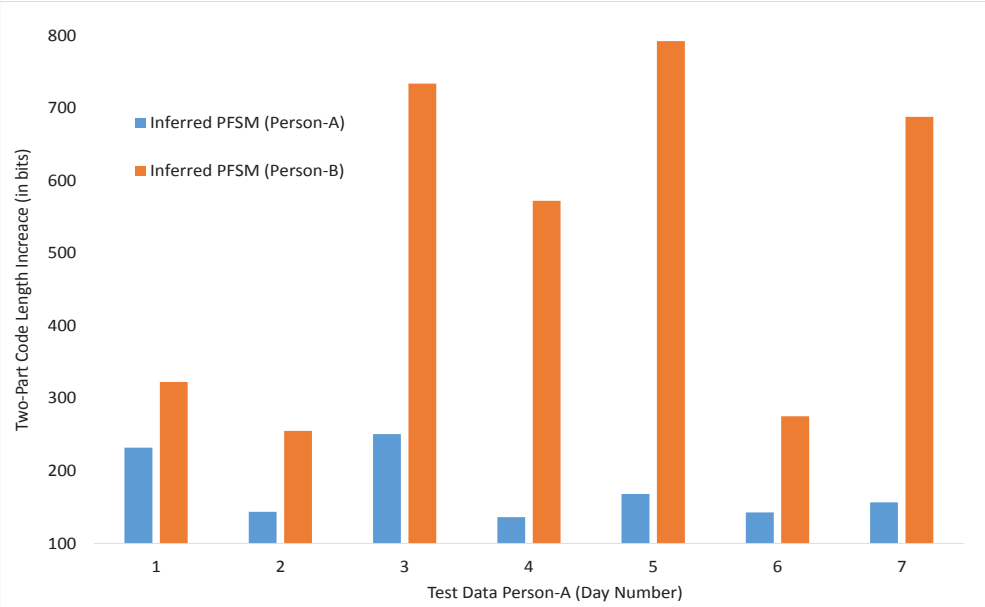


Figure 7.11: Code Length increase in bits in the inferred HPFSM models on test data of Person-A

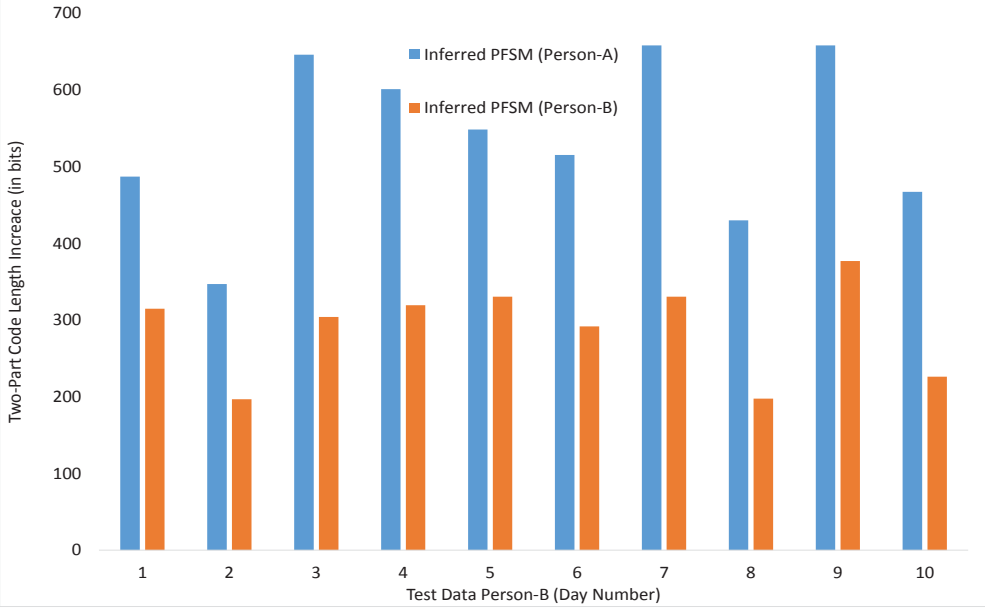


Figure 7.12: Code Length increase in bits in the inferred HPFSM models on test data of Person-B

7.4 Summary

This chapter investigated a new learning method based on hierarchical construction of a simple PFSM model called as an HPFSM model. The HPFSM model encoding of both the structure and the data were discussed. The benefits of the construction were understood in viewpoint of the real things happening around us. We discussed the example of a multilingual person and the activities performed by an individual in his daily life if that involves moving around to various places. Most of the time the movements are local and then there is absolutely no need to encode them using one big machine. We can rather construct small machines and combine them somehow so that the purpose is achieved. This is how the construction of hierarchical PFSMs was driven by the need of representing things in a more concise manner and the the obvious advantage is a less code length model that still represents the same grammar.

The need for constructing the HPFSM model was then followed by an artificial design of an HPFSM model to understand what the model looks like. Using the same design we discussed the two-part code length calculation using MML. The first part code length that encodes the model was summarized by Equation 7.1 and the second part code length that encodes the data using the model was summarized by Equation 7.3.

Finally two different experiments were performed to show the benefits of hierarchical encoding over the non-hierarchical style of encoding. Under the first experimental set up, we considered two artificially created HPFSM models. Different datasets were generated out of the artificially created HPFSM models. The datasets consisted of variable number of random strings from 2 to 5000. In the first artificially created HPFSM model (Example-1 Figure 7.2) , the initial probabilities of transition assumed for string generation were considered uniform. Whereas in the second artificially created HPFSM model (Example-2 Figure 7.1, the initial probabilities of transition were considered variable with the internal transitions set to high probabilities and outer transitions set to low probabilities. The two-part

encoding cost was then compared with the non-hierarchical PFSM (inferred PFSM) model and the single-state model. The datasets with strings greater than 50 were encoded in a cheaper way by the HPFSM model as compared to the other models, eventually leading to the conclusion that the HPFSM models offered a cheaper way of encoding if the datasets were hierarchical.

The second experiment was performed on the real datasets from the UCI repository. The datasets were the Activities of Daily Living (ADL) datasets. The HPFSM model construction from the datasets was discussed followed by an analysis using the models. The HPFSM model code length, again, in this case also gave a better performance in terms of encoding as compared to the other models. The model learning or encoding was followed by an analysis using the models. We used prediction as the evaluating criteria to test the models. The HPFSM models were correctly able to predict the class from the instances of test data from the ADL datasets.

Chapter 8

Conclusions and Future Work

8.1 Summary and Conclusions

I conclude the thesis by summarizing the contributions made in relation to the central underlying question: *how can a model be developed that can model regularities and pattern of text data, is easy to build, can be further used in doing some meaningful analysis and whose performance is measured by comparing it with the other models?* The thesis answers the central question by setting out various sub-goals which are briefly described here in the chapter. But the final goal in the thesis was to develop a model for a particular kind of data and the success of the model was measured through comparative studies.

Probabilistic Finite State Machines (PFSMs) are models that can model text data that contain regularities and patterns. Now, such text data can be obtained from various sources such as a natural language corpus, a DNA sequence or an email text corpus. If we recollect the concept of a finite state machine, then it is best described as a machine that represents a regular grammar. From the regular grammar, originate languages, that are termed as regular languages. The elements of a regular language can be termed as words or tokens or sentences depending upon the context of the application areas. Any text data can be viewed as a regular language and consequently Finite State Machines (FSMs) can model them effectively. A PFSM is an extended FSM that describes a population of sentences.

In the thesis we learned PFSM models from various sources of text data. According to Wallace (Wallace, 2005), the model is an assertion on the observed data and the inductive inference of the model tells how probable the model is in generating the data. For the inductive inference Minimum Message Length (MML) was used. MML has emerged as a powerful tool, not only in providing a coding mechanism for structures as PFSMs but also plays an important role in the inductive inference of such structures. The sub-goals set out to achieve the main goal are summarized as below:

In Chapter 2, we discussed the underlying theory of FSMs and PFSMs. Chapter 3 discussed the Minimum Message Length (MML) principle in general. MML is a Bayesian approach premised on Bayes's theorem. We discussed in this chapter the MML calculation for a discrete Binomial distribution case and MML calculation for a discrete Multinomial distribution case. The calculations were later referred in the MML encoding of a PFSM model.

Chapter 4 discussed the MML two-part code length calculation of a PFSM model. The information-theoretic MML first provides an encoding of the structure of PFSM model by making use of some discrete information in the PFSM. MML does the second part encoding by encoding the data generated by the PFSM model. This is done by encoding the transition probabilities. The two-part MML encoding is followed by a search to find the best model that describes the data. For this we discussed a novel induction method where the two-part MML code length was used as a objective function to search the best model among the competing models. We also discussed in this chapter another search method based on Simulated Annealing (SA) to obtain an optimal MML PFSM model. In all the experiments, we have used SA as the search method to get MML PFSMs.

The contributions arising out of the research work were discussed in the subsequent chapters. We list the contributions below. The first sub-goal set out was accomplished by learning the PFSM models for the text kind of data. We explored two application areas that contained text data with specifically two classes and

the learning was carried out under a supervised learning environment. The contributions were made in relation to the approaches that we developed for learning and classification using the PFSM models and finally measuring the performance of the approaches by comparing with Minimum Description Length (MDL) and Naive Bayes Classifiers. The two application areas explored were, the publicly available Enron spam dataset and the Activities of Daily Living (ADL) datasets. Both the examples considered had two instances of classified data contained in them.

- The first approach was learning the two-machine PFSM models from the two classes of data. The training datasets of the two classes were used to learn the initial PFSM models. The initial PFSM models were the Prefix Tree Acceptors (PTA) of the training datasets of the two classes. The initial PTAs were inferred by the SA search method to get minimum two-part code length PFSM models. The two models were then used for classification of the test datasets of the two classes. The two-machine model approach was experimented on the Enron spam datasets and the ADL datasets. In both the datasets, the training of the models was done on 50% of the datasets and the remaining 50% was used for testing. Precision, weighted accuracy and recall were used as performance measures in the Enron spam datasets whereas we used prediction of class in the ADL datasets. The performance of the 6 Enron spam datasets were compared against the MDL performance on the same datasets. The two-machine PFSM-MML method resulted in better results than the MDL method by giving 99.75% average classification accuracy. The MDL method on the other hand resulted in 98.60% average classification accuracy. On the ADL datasets also, the two-machine PFSM model approach resulted in correct prediction of class for every test dataset of the two individuals.

  The main advantage as seen from the design of the two-machine approach was that the method was insensitive to thresholds and as a consequence of which it was possible to achieve both precision and recall values high at the same time.

- The two-machine design was modified to single-machine design in the second approach. The benefits as seen in the single-machine design as compared to the two-machine design were: the cost of building model was less, the model was built in less time as compared to the two-machine design as it avoided any inference of the PFSM model and lastly the single-machine design had more suitability in multiple class classification scenario as compared to the two-machine design.

  The experiments were done on the Enron spam datasets. We compared the cost of building the single-machine model with the cost of building two-machine models on the training dataset of the Enron spam dataset. The single-machine model in every case resulted in a cheaper code length machine than the combined code length of the two machines in two-machine design approach. The performance of the single-machine model was compared against the different versions of the Naive Bayes classifiers and the general conclusion was that in almost all the cases the single-machine design performed better than the different versions of the Naive Bayes classifiers.

- The third approach was developed in view of obtaining an even more concise representation of grammar by encoding through the means of Hierarchical PFSM (HPFSM). In an HPFSM, several small PFSMs were connected in the outer structure or the external PFSM and each of the small PFSMs or internal PFSMs independently generated their own language by making use of the internal transitions and internal input alphabet sets. Here we discussed the two-part MML calculation for an HPFSM model that was in line with the two-part MML calculation for a PFSM model. The traditional coding scheme of a PFSM model was modified and we came up with a new coding scheme for HPFSMs. One benefit of encoding using an HPFSM model was shown in terms of generating data from an artificial HPFSM model and then showing that, when trying to infer the model from which the data came, for datasets above a certain size, the shortest code length was obtained from the

HPFSM model. Datasets with variable number of strings were generated by the artificial HPFSM model and it was proved experimentally that for strings greater than 50, the HPFSM model resulted in giving the best compression when compared to the non-hierarchical PFSM model and the one-state model.

Because of the nature of the datasets in the ADL datasets, the HPFSM model learning and prediction of classes was possible. We did experimentation on the ADL datasets by building two HPFSM models from the training dataset of the two individuals and the test dataset was used in prediction of classes. The test datasets, when input to both the HPFSM models, resulted in giving accurate results.

8.2 Future Work

The list of things that can be further taken up and implemented for future research include the following:

- We came up with the approach of two-machine model in our research to classify a test data that belonged to one of the two classes. The idea of two-machine model can be generalized to multiple class classification scenario where multiple PFSM models can be built from the multiple classes of data. This method is expected to give good classification results but then, the individual classes need to be learnt and inferred individually. This may be applied for datasets of size that are relatively small but for large size datasets, converting into single machine concept will be more practical. Although we did not attempt to try classification into multiple classes either with multiple machines or with single machine, but this is one possibility where the current research can be taken further.

- The current research was focussed on training the PFSM models with text data. Text data is too simplistic assumption for a dataset when classification is concerned. Even the emails, where classification into spam and non-spam

category is required, they might contain data in image form or any other multimedia form. If we desire to use the PFSM modelling for such data source, appropriate conversion must be done in order to model them as a PFSM model. So in future, possibilities arise to model a multimedia data source.

- We also assumed simplistically that the occurrence of tokens or words in the email text corpus were independent events. In reality, there may be dependency of tokens on one other which can be taken into account in future.

- In the Enron spam datasets, the emails were first tokenized and then, with the appropriate feature selection method, the attributes were selected. The criterion for feature selection was the term frequency of the attribute in the classes of emails. We worked on 3000 high frequency attributes in all our experimental evaluations just to make our work comparable with the other methods on the same dataset. For example, all the Naive Bayes classifiers in the previous studies worked on 3000 high frequency attributes and considered a threshold of 0.5 in their experiments. We kept the experimental conditions the same. In future, different feature selection methods can be employed and also, experiments can be tried with attributes more than 3000 in number and the impact on the results can be seen.

- The PFSM models were trained on word level in our experiments with the Enron spam datasets. The sentence in an email was broken down into tokens by removing all the stop words and the delimiter symbols. In future the models can be trained on phrase level or even sentence level.

- For training the models, we considered 50% of the dataset and performed testing on the other half. In future, a more robust training and testing procedure like the *k-fold* cross validation method can be employed on the datasets.

- For performance evaluation, we used precision and recall as the evaluating metrics. Although precision and recall are the two widely used evaluation

criteria in classification, but in future other criteria like Receiver Operations Characteristics (ROC) curve can be used for comparisons.

- We chose two application areas where PFSM models could be learnt. If modelling with text data is desired, then there are other application areas where PFSM models can be learnt like the natural language corpus or DNA sequences. The PFSM models can be used in DNA sequence clustering.

- We compared the results obtained using the MML (single-machine and two-machine) methods with the Minimum Description Length (MDL) classifier and the Naive Bayes classifiers. Comparison with more text classification methods can be done in future.

Publications

The publications done as part of this research work are listed below:

- **V. Saikrishna**, S. Ray, "Improved Approximate Multiple-Pattern String Matching using Consecutive N-Gramm," *International Journal of Computer Applications (IJCA)*, vol. 81, no. 1 , pp 26-31, November 2013.

- **V. Saikrishna**, D. L. Dowe, S. Ray, "MML Inference of Finite State Automata for Probabilistic Spam Detection," *In Proceedings of International Conference on Advances in Pattern Recognition (ICAPR)*, pp 1-6, IEEE Computer Society Press, 2015.

- **V. Saikrishna**, D. L. Dowe, S. Ray, "Statistical Compression-Based Models for Text Classification," *In Proceedings of IEEE International Conference on Eco-Friendly Computing and Communication Systems (ICECCS)*, pp 1-6, 2016.

The manuscripts under preparation are the following:

- **V. Saikrishna**, D. L. Dowe, S. Ray, "Learning Hierarchical Probabilistic Finite State Automata using MML".

- **V. Saikrishna**, D. L. Dowe, S. Ray, "Learning Probabilistic Finite State Machines using MML with applications to Text Classification".

Bibliography

Enron. `http://www.iit.demokritos.gr/skel/i-config` and `http://www.aueb.gr/users/ion/publications`.

Y. Agusta and D. L. Dowe. MML clustering of continuous-valued data using Gaussian and t distributions. In B. McKay and J. Slaney, editors, *Proceedings of the 15th Australian Joint Conference on Artificial Intelligence*, volume 2557 of *Lecture Notes in Artificial Intelligence (LNAI)*, pages 143–154, Berlin, Germany, 2002. Springer-Verlag.

Y. Agusta and D. L. Dowe. Unsupervised learning of gamma mixture models using minimum message length. In M. H. Hamza, editor, *Proceedings of the 3rd IASTED Conference on Artificial Intelligence and Applications*, pages 457–462, Benalmadena, Spain, 2003a. ACTA Press.

Y. Agusta and D. L. Dowe. Unsupervised learning of correlated multivariate Gaussian mixture models using MML. In T. D. Gedeon and L. C. Fung, editors, *Proceedings of the 16th Australasian Joint Conference on Artificial Intelligence*, volume 2903 of *Lecture Notes in Artificial Intelligence*, pages 477–489, Berlin, Germany, 2003b. Springer-Verlag.

T. Almeida and A. Yamakami. Advances in Spam Filtering Techniques. *Computational Intelligence for Privacy and Security*, 394:199–214, 2012.

S. D. Anderson. Combining Evidence using Bayes' Rule. *Technical report, Wellesley College*, 2007.

Andrej Bratko, Gordon V. Cormack, Bogdan Filipič, Thomas R. Lynam, Blazupan. Spam Filtering using Statistical Data Compression Models . *Journal of Machine Learning Research*, 7:2673–2698, 2006.

I. Androutsopoulos, J. Koutsias, K. V. Chandrinos, and C. D. Spyropoulos. An Experimental Comparison of Naive Bayesian and Keyword-Based Anti-Spam Filtering with Personal E-mail Messages. In *Proc. of 23rd Annual International ACM SIGIR Conference on Research and Developement in Information Retrieval*, pages 160–167, 2000.

D. Angluin. On the complexity of minimum inference of regular sets. *Information and Control*, 39(3):337–350, 1978.

R. A. Baxter and J. J. Oliver. MDL and MML: Similarities and Differences. *Dept. Comput. Sci. Monash Univ., Clayton, Victoria, Australia, Tech. Rep*, 207, 1994.

D. M. Boulton. *Numerical Classification based on Information Measure.* Master's thesis, Department of Computer Science, Monash University, Clayton, Australia, 1970.

D. M. Boulton. *The information measure criterion for intrinsic classification.* PhD thesis, Department of Computer Science, Monash University, Clayton, Australia, 1975.

D. M. Boulton and C. S. Wallace. The information content of a multistate distribution. *Journal of Theoretical Biology*, 23(2):269–278, 1969.

D. M. Boulton and C. S. Wallace. A program for numerical classification. *The Computer Journal*, 13(1):63–69, 1970.

D. M. Boulton and C. S. Wallace. A comparison between information measure classification. In *Proceedings of ANZAAS Congress, Perth*, 1973a.

D. M. Boulton and C. S. Wallace. An information measure for hierarchic classification. *The Computer Journal*, 16(3):254–261, 1973b.

D. M. Boulton and C. S. Wallace. Occupancy of a rectangular array. *The Computer Journal*, 16(1):57–63, 1973c.

D. M. Boulton and C. S. Wallace. An information measure for single link classification. *The Computer Journal*, 18(3):236–238, 1975.

P. Cheeseman. On finding the most probable model. In *Computational models of scientific discovery and theory formation/edited by Jeff Shrager and Pat Langley.* Morgan Kaufmann Publishers, 1990.

N. Chmait. *Understanding Collective Intelligence in Agent-Based Systems: an Information-Theoretic Approach to the Measurement and Comparison of Intelligence in Groups.* Faculty of Information Technology, Monash University, Australia, 2017.

C. H. Clelland. Assessment of candidate PFSA models induced from symbol datasets. In *Tech Report TR-C95/02.* Deakin University, Australia, 1995.

C. H. Clelland and D. A. Newlands. PFSA modelling of behavioural sequences by evolutionary programming. In *Proceedings of the Conference on Complex Systems: Mecahnism for Adaptation*, pages 165–172. Rockhampton, Queensland: IOS Press, 1994.

M. S. Collins and J. J. Oliver. Efficient induction of finite state automata. In *Proceedings of the Thirteenth conference on Uncertainty in artificial intelligence*, pages 99–107. Morgan Kaufmann Publishers Inc., 1997.

J. W. Comley and D. L. Dowe. General Bayesian networks and asymmetric languages. In *Proceedings of the 2nd Hawaii International Conference on Statistics and Related Fields*, pages 1–18, 2003.

J. W. Comley and D. L. Dowe. Minimum Message Length and generalized Bayesian nets with asymmetric languages. *Advances in Minimum Description Length: Theory and Applications, Chapter 11, In P. D. Grünwald and I. J. Myung and M.*

A., Pitt (Eds), pages 265–294, 2005. Final camera ready copy was submitted in October 2003.

G. V. Cormack. Email spam filtering: A systematic review. *Foundations and Trends in Information Retrieval*, 1(4):335–455, 2007.

D. L. Dowe. Foreword re C. S. Wallace. *The Computer Journal*, 51(5):523–560, 2008a. doi: 10.1093/comjnl/bxm117. URL +http://dx.doi.org/10.1093/comjnl/bxm117.

D. L. Dowe. Minimum Message Length and statistically consistent invariant (objective?) Bayesian probabilistic inference—from (medical)"evidence". *Social Epistemology*, 22(4):433–460, 2008b.

D. L. Dowe. MML, hybrid Bayesian network graphical models, statistical consistency, invariance and uniqueness. In P. S. Bandyopadhyay, M. R. Forster (Eds.), editor, *Handbook of the Philosophy of Science*, volume 7 of *Philosophy of Statistics*, pages 901–982. Elsevier, 2011.

D. L. Dowe. Introduction to ray solomonoff 85th memorial conference. In *Algorithmic Probability and Friends. Bayesian Prediction and Artificial Intelligence*, pages 1–36. Springer, 2013.

D. L. Dowe and A. R. Hajek. A computational extension to the Turing Test. In *Proceedings of the 4th Conference of the Australasian Cognitive Science Society, University of Newcastle, NSW, Australia*, volume 1. Citeseer, 1997a. URL http://users.monash.edu/~dld/Publications/1997/DoweHajek1997a.pdf.

D. L. Dowe and A. R. Hajek. A computational extension to the Turing Test. Technical Report #97/322, Department of Computer Science, Monash University, Melbourne, Australia, 1997b. URL http://users.monash.edu/~dld/Publications/1997/DoweHajek1997b.pdf.

D. L. Dowe and A. R. Hajek. A non-behavioural, computational extension to the Turing Test. In *International conference on computational intelligence & multimedia applications (ICCIMA'98), Gippsland, Australia*, pages 101–106, 1998. URL `http://users.monash.edu/~dld/Publications/1998/DoweHajek1998.pdf`.

D. L. Dowe, J. J. Oliver, and C. S. Wallace. MML estimation of the parameters of the spherical Fisher distribution. In *International Workshop on Algorithmic Learning Theory*, pages 213–227. Springer, 1996.

D. L. Dowe, R. A. Baxter, J. J. Oliver, and C. S. Wallace. Point estimation using the Kullback-Leibler loss function and MML. *Research and Development in Knowledge Discovery and Data Mining*, pages 87–95, 1998.

H. Drucker, D. Wu, and V. N. Vapnik. Support vector machines for spam categorization. *IEEE Transactions on Neural networks*, 10(5):1048–1054, 1999.

T. Edgoose and L. Allison. MML Markov classification of sequential data. *Statistics and Computing*, 9(4):269–278, 1999.

T. Edgoose, L. Allison, and D. L. Dowe. An MML classification of protein structure that knows about angles and sequence. In *Pacific Symposium on Biocomputing*, volume 3, pages 585–596. World Scientific Publishing, 1998.

R. T. Edwards and D. L. Dowe. Single factor analysis in MML mixture modelling. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, volume 1394 of *Lecture Notes in Artificial Intelligence (LNAI)*, pages 96–109. Springer, 1998.

J. Feldman. Some decidability results on grammatical inference and complexity. *Information and control*, 20(3):244–262, 1972.

M. A. T. Figueiredo and A. K. Jain. Unsupervised learning of finite mixture models. *IEEE Transactions on pattern analysis and machine intelligence*, 24(3):381–396, 2002.

L. J. Fitzgibbon, D. L. Dowe, and F. Vahid. Minimum Message Length autoregressive model order selection. In M. Palanaswami, C. C. Sekhar, G. K. Venayagamoorthy, S. Mohan, and M. K. Ghantasala, editors, *Proceedings of International Conference on Intelligent Sensing and Information Processing*, pages 439–444, Chennai, India, 2004. IEEE. Catalogue Number: 04EX783.

B. R. Gaines. Behaviour/Structure Transformation under Uncertainity. *International Journal of Man-Machine Studies*, 8:337–365, 1976.

R. G. Gallager. *A Book on Information theory and reliable communication*, volume 2. Springer, 1968.

M. P. Georgeff and C. S. Wallace. A general selection criterion for inductive inference. In *European Conference of Artificial Intelligence (ECAI)*, volume 84, pages 473–482, 1984.

E. M. Gold. Complexity of automaton identification from given data. *Information and control*, 37(3):302–320, 1978.

J. M. G. Hidalgo. Evaluating cost-sensitive unsolicited bulk email categorization. In *Proceedings of the 2002 ACM symposium on Applied computing*, pages 615–620. ACM, 2002.

P. Hingston. Inference of regular languages using model simplicity. *Australian Computer Science Communications*, 23(1):69–76, 2001.

D. A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952.

G. H. John and P. Langley. Estimating continuous distributions in Bayesian classifiers. In *Proceedings of the Eleventh conference on Uncertainty in artificial intelligence*, pages 338–345. Morgan Kaufmann Publishers Inc., 1995.

P. Kasarapu and L. Allison. Minimum Message Length estimation of mixtures of multivariate Gaussian and von Mises-Fisher distributions. *Machine Learning*, 100 (2):333–378, 2015.

A. Kolez and J. Alspector. SVM-based Filtering of E-mail Spam with Content-specific Misclassification Costs. In *Proceedings of the First International Conference on Data Mining*, pages 1–14. San Jose, CA, USA, 2001.

E. L. Lehmann and G. Casella. *Theory of point estimation*. Springer Science & Business Media, 2006.

R. Lenhardt. *Probabilistic Automata with Parameters*. Oriel College University of Oxford, 2009.

D. E. Losada and L. Azzopardi. Assessing multivariate Bernoulli models for information retrieval. *ACM Transactions on Information Systems (TOIS)*, 26(3):17, 2008.

E. Makalic and D. F. Schmidt. MDL multiple hypothesis testing. In *Proceedings of the 4th Workshop on Information Theoretic Methods in Science and Engineering (WITMSE-11)*, pages 45–48, Helsinki, Finland, 2011. Citeseer. Invited paper.

A. McCallum and K. Nigam. A comparison of event models for Naive Bayes text classification. In *AAAI-98 workshop on learning for text categorization*, volume 752, pages 41–48. Menlo Park, CA, USA, 1998.

V. Metsis, I. Androutsopoulos, and G. Paliouras. Spam Filtering with Naive Bayes - Which Naive Bayes? In *Third Conference on Email and Anti-Spam (CEAS)*, pages 28–69, 2006.

S. B. Molloy, D. W. Albrecht, D. L. Dowe, and K. M. Ting. Model-Based Clustering of Sequential Data. In *Proceedings of the 5th Annual Hawaii International Conference on Statistics, Mathematics and Related Fields*, January 2006.

S. Musgrave and D. L. Dowe. Kinship, optimality, and typology. *Behavioral and Brain Sciences (BBS)*, 33(5):397–398, 2010.

J. J. Oliver. Decision graphs – an extension of decision trees. In *Proceedings of the 4th International Workshop on Artificial Intelligence and Statistics*, pages 343–350, 1993. Extended version available as TR173, Department of Computer Science, Monash University, Clayton, Australia.

J. J. Oliver and D. Hand. Introduction to minimum encoding inference. *Technical Report No. 94/205, Department of Computer Science, Monash University, Australia*, 1994.

J. J. Oliver and C. S. Wallace. Inferring decision graphs. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI-91), workshop 8*, 1991.

J. J. Oliver, D. L. Dowe, and C. S. Wallace. Inferring decision graphs using the Minimum Message Length principle. In *Proceedings of the Australian Joint Conference on Artificial Intelligence*, pages 361–367, 1992.

F. J. Ordonez, P. de Toledo, and A. Sanchis. Activity recognition using hybrid generative/discriminative models on home environments using binary sensors. *Sensors*, 13(5):5460–5477, 2013.

A. Raman and J. Patrick. 14 Linguistic similarity measures using the minimum message length principle. *Archaeology and Language I: Theoretical and Methodological Orientations*, page 262, 1997a.

A. V. Raman. *An information theoretic approach to language relatedness: a dissertation submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy in Information Systems at Massey University.* PhD thesis, Massey University, 1997.

A. V. Raman and J. D. Patrick. Beam search and simba search for PFSA inference. Technical report, Tech Report 2/97, Massey University Information Systems Department, Palmerston North, New Zealand, 1997b.

A. V. Raman, J. D. Patrick, and P. North. The sk-strings method for inferring PFSA. In *Proceedings of the workshop on automata induction, grammatical inference and language acquisition at the 14th international conference on machine learning (ICML97)*, 1997.

A. V. Raman, P. Andreae, and J. Patrick. A Beam Search Algorithm for PFSA Inference. *Pattern Analysis and Applications*, 1:121–129, 1998.

K. Rice. Bayesian Statistics (a very brief introducion). *Biostat*, Epi. 515, 2014.

J. Rissanen. Modeling by Shortest Data Description. *Automatica*, 14(5):465–471, 1978.

J. Rissanen. *Stochastic complexity in statistical inquiry*, volume 15. World scientific, 1998.

J. Rissanen. Hypothesis Selection and Testing by the MDL Principle. *The Computer Journal*, 42(4):260–269, 1999.

M. Sahami, S. Dumais, D. Heckerman, and E. Horvitz. A Bayesian approach to filtering junk e-mail. In *Learning for Text Categorization: Papers from the 1998 workshop*, volume 62, pages 98–105, 1998.

V. Saikrishna, D. L. Dowe, and S. Ray. MML Inference of Finite State Automata for Probabilistic Spam Detection. In *Proceedings of International Conference on Advances in Pattern Recognition (ICAPR)*, pages 1–6. IEEE Computer Society Press, 2015.

V. Saikrishna, D. L. Dowe, and S. Ray. Statistical Compression-Bases Models for Text Classification. In *Proceedings of International Conference on Eco-Friendly Computing and Communication Systems (ICECCS)*, pages 1–6. IEEE, 2016.

D. F. Schmidt. *Minimum Message Length inference of autoregressive moving average models (PhD. thesis).* Faculty of IT, Monash University, 2008.

K. M. Schneider. On word frequency information and negative evidence in Naive Bayes text classification. In *Advances in Natural Language Processing*, pages 474–485. Springer, 2004.

D. Sculley and G. Wachman. Relaxed Online SVMs in the TREC Spam Filtering Track. In *Proceedings of the 30th International ACM SIGIR Conference on Research and Development in Information Retrieval, Amsterdam, Netherlands*, pages 415–422, 2007.

C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, 1948.

M. Sipser. *Introduction to the Theory of Computation*, volume 2. Thomson Course Technology Boston, 2006.

P. J. Tan and D. L. Dowe. MML inference of decision graphs with multi-way joins. In *Australian Joint Conference on Artificial Intelligence*, pages 131–142. Springer, 2002.

P. J. Tan and D. L. Dowe. MML inference of decision graphs with multi-way joins and dynamic attributes. In *Australasian Joint Conference on Artificial Intelligence*, volume 2903 of *Lecture Notes in Artificial Intelligence (LNAI)*, pages 269–281. Springer, 2003.

P. J. Tan and D. L. Dowe. MML inference of oblique decision trees. In *Australasian Joint Conference on Artificial Intelligence*, volume 3339 of *Lecture Notes in Artificial Intelligence (LNAI)*, pages 1082–1088. Springer, 2004.

A. Torsello and D. L. Dowe. Learning a generative model for structural representations. In *Proceedings of the 21st Australasian Joint Conference on Artificial Intelligence*, volume 5360 of *Lecture Notes in Artificial Intelligence (LNAI)*, pages 573–583, Auckland, NZ, 2008a. Springer.

A. Torsello and D. L. Dowe. Supervised learning of a generative model for edge-weighted graphs. In *Proceedings of the 19th International Conference on Pattern Recognition, 2008 (ICPR 2008)*, pages 1–4, Tampa, Florida, U.S.A., 2008b. IEEE Catalog Number: CFP08182.

G. Visser and D. L. Dowe. Minimum Message Length Clustering of Spatially-Correlated Data with Varying Inter-Class Penalties. In *Computer and Information Science, 2007. ICIS 2007. 6th IEEE/ACIS International Conference on*, pages 17–22. IEEE, 2007.

G. Visser, P. Dale, D. L. Dowe, E. Ndoen, M. Dale, and N. Sipe. A novel approach for modeling malaria incidence using complex categorical household data: The Minimum Message Length (MML) method applied to Indonesian data. *Computational Ecology and Software*, 2(3):140–159, 2012.

C. S. Wallace. An improved program for classification. In *Proceedings of the 9th Australian computer science conference*, volume 8, pages 357–366, 1986.

C. S. Wallace. Classification by Minimum Message Length Inference. In *Advances in Computing and Information—ICCI'90*, pages 72–81. Springer, 1990.

C. S. Wallace. Intrinsic classification of spatially correlated data. *The Computer Journal*, 41(8):602–611, 1998.

C. S. Wallace. *Statistical and Inductive Inference by Minimum Message Length*. Information Science and Statistics. Springer Science and Business Media, Spring Street, NY, USA, 2005.

C. S. Wallace and D. M. Boulton. An information measure for classification. *The Computer Journal*, 11(2):185–194, 1968.

C. S. Wallace and D. M. Boulton. An invariant Bayes method for point estimation. *Classification Society Bulletin*, 3(3):11–34, 1975.

C. S. Wallace and D. L. Dowe. *MML estimation of the von Mises concentration parameter*. Monash University, Technical Report No. 93/193, Department of Computer Science, 1993.

C. S. Wallace and D. L. Dowe. Estimation of the von Mises concentration parameter using Minimum Message Length. In *Proc. 12th Australian Statistical Soc. Conf., Monash University, Australia*, 1994a.

C. S. Wallace and D. L. Dowe. Intrinsic classification by MML-the Snob program. In *Proceedings of the 7th Australian Joint Conference on Artificial Intelligence*, pages 37–44. World Scientific, 1994b.

C. S. Wallace and D. L. Dowe. MML Mixture Modelling of Multi-State Poisson von Mises Circular and Gaussian Distribution. *COMPUTING SCIENCE AND STATISTICS*, pages 608–613, 1997.

C. S. Wallace and D. L. Dowe. Minimum Message Length and Kolmogorov Complexity. *The Computer Journal*, 42(4):270–283, 1999a.

C. S. Wallace and D. L. Dowe. Refinements of MDL and MML coding. *The Computer Journal*, 42(4):330–337, 1999b.

C. S. Wallace and D. L. Dowe. Rejoinder. *The Computer Journal*, 42(4):345–347, 1999c.

C. S. Wallace and D. L. Dowe. MML clustering of multi-state, Poisson, von Mises circular and Gaussian distributions. *Statistics and Computing*, 10(1):73–83, 2000.

C. S. Wallace and P. R. Freeman. Estimation and inference by compact coding. *Journal of Royal Statistical Society series B*, 49(3):240–252, 1987.

C. S. Wallace and P. R. Freeman. Single-factor analysis by Minimum Message Length estimation. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 195–209, 1992.

C. S. Wallace and M. Georgeff. A General Objective for Inductive Inference. *Technical Report No. 32, Department of Computer Science, Monash University, Australia*, 1983.

C. S. Wallace and J. Patrick. Coding decision trees. *Machine Learning*, 11(1):7–22, 1993.

C. S. Wallace, K. B. Korb, and H. Dai. Causal discovery via MML. In *ICML*, volume 96, pages 516–524, 1996.

D. R. Wright. *Finite State Machines*. CSC215 Class Notes. Prof. David R. Wright website, N. Carolina State University, July 14 2012.