FREQUENT SUBSTRING-BASED SEQUENCE CLASSIFICATION USING REDUCED ALPHABETS

A Thesis
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By

Charith Devinda Chitraranjan

In Partial Fulfillment of the Requirements
for the Degree of
MASTER OF SCIENCE

Major Department:
Computer Science

June 2011

Fargo, North Dakota

North Dakota State University
Graduate School

Title

FREQUENT SUBSTRING-BASED SEQUENCE CLASSIFICATION

USING REDUCED ALPHABETS

By

CHARITH DEVINDA CHITRARANJAN

The Supervisory Committee certifies that this ***disquisition*** complies with North Dakota State University's regulations and meets the accepted standards for the degree of

MASTER OF SCIENCE

ABSTRACT

Chitraranjan, Charith Devinda, M.S., Department of Computer Science, College of Science and Mathematics, North Dakota State University, June 2011. Frequent Substring-Based Sequence Classification Using Reduced Alphabets. Major Professor: Dr. Anne M. Denton.

In recent years, various disciplines have generated large quantities of sequence data which has necessitated automated techniques for classifying these sequences into different categories of interest. Especially with the rapid rate at which biological sequence data has been emerging out of high throughput sequencing efforts, the need to interpret these large quantities of raw sequence data and gain deeper insights into them has become an essential part of modern biological research. Understanding the functions, localization and structure of newly identified protein sequences in particular has become a major challenge and is seeking the aid of computational techniques to keep up with the pace. In this thesis, we[1] evaluate frequent pattern-based algorithms for predicting aforementioned attributes of proteins from their primary structure (amino acid sequence). We also apply our algorithms to datasets containing wheat Expressed Sequence Tags (ESTs) as an attempt to predict ESTs that are likely to be located near the centromere of their respective chromosomes. We use frequent substrings mined from the training sequences as features to train a classifier. Our evaluation includes SVM and association rule-based classifiers. Some amino acids have similar properties and may substitute one another without altering the topology

[1]Co-authors: Charith Chitraranjan and Anne Denton, Ph.D.
Charith Chitraranjan contributed with algorithm design and implementation, experimental evaluation and writing the manuscript. Anne Denton contributed with the initial research idea, concepts and editing the manuscript.

or function of a protein. Therefore, we use a combination of reduced amino acid alphabets in an attempt to capture patterns that may contain such substitutions. Frequent substrings mined from different alphabets are treated as features resulting from multiple sources and we evaluate both feature fusion and classifier fusion approaches towards multiple source prediction. We compare the performance of the different approaches using protein sub-cellular location, protein function and EST chromosomal location datasets. Pair-wise sequence-alignment-based Nearest Neighbor and basic SVM k-gram classifiers are also included as baseline algorithms in the comparison. Results show that frequent pattern-based SVM classifiers demonstrate better performance compared to other classifiers on the sub-cellular location datasets and they perform competitively with the nearest neighbor classifier on the protein function datasets. Our results also show that the use of reduced alphabets provides statistically significant performance improvements for the SVM-based classifier fusion algorithm, for half of the classes studied.

ACKNOWLEDGEMENTS

First and foremost, I would like to express my gratitude to my thesis advisor, Dr. Anne Denton, for her valuable advise, continuous encouragement and providing me with an opportunity to be a part of her research group, all of which means a lot to me. I thank Dr. Kendall Nygard for his guidance as the graduate coordinator of the Computer Science Department, Dr. Shahryar Kianian for letting me take part in the Wheat Radiation-Hybrid mapping project and Dr. Saeed Salem for introducing useful research tools and sharing his research ideas with me. I also thank them all for being in my supervisory committee and for their interest in my work.

I thank all the members of our research group for their valuable input and pleasant companionship. I thank the staff of the Computer Science Department at NDSU for helping me with various administrative tasks.

I would also like to extend my gratitude to all my teachers and university professors in Sri Lanka who laid the foundation for my academic career.

Last but certainly not least, I thank my parents, my brother, other family members and friends whose support and encouragement made this thesis possible.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

SVM	-	Support Vector Machine
ASSO_FF	-	Association rule-based feature fusion
SVM_FF	-	Support Vector Machine-based feature fusion
SVM_CF	-	Support Vector Machine-based classifier fusion
OM	-	Outer membrane
Cyto	-	Cytoplasmic
CM	-	Cytoplasmic membrane
PP	-	Periplasmic
EC	-	Extracellular
CW	-	Cell wall

CHAPTER 1. INTRODUCTION

Large amounts of sequence data has been emanating from various different fields in recent years. Biological sequences contribute to a vast majority of this data while a considerable portion has originated from other sources such as software execution traces, medical informatics, stock market trends, weather patterns and market basket research as well.

With the abundance of data, sequence classification has become a very important task with a broad range of real world applications such as protein function prediction, classification of ECG time series data as normal or abnormal, classification of software execution traces as erroneous or successful and classification of customers into different groups based on their purchase history.

There has been a steady increase in the amounts of biological sequence data largely due to recent advances in automated high-throughput sequencing technology. Genomes of many species have been fully sequenced resulting in a wealth of sequence data. However, sequencing a genome in only the first step in an attempt towards understanding a species. In order to gain deeper insights, attributes such as structure, function and sub-cellular location of the corresponding proteins need to be determined. The molecular structures of newly identified proteins are being determined at a rapid rate thanks to structure genomics projects [22]. However, experimental determination and manual curation of protein function and sub-cellular location has been lagging way behind the exponential rate at which sequence data has been emerging. Experimental evaluation of these attributes is expensive and time consuming. Therefore, there is a great incentive for making automated predictions about protein function and sub-cellular location in order to characterize and understand the roles of newly sequenced proteins in an efficient manner.

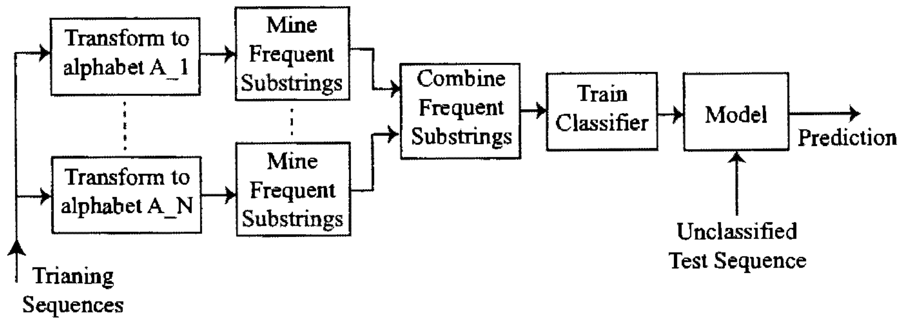
In general, classification is the process of predicting the class membership a novel data point from a finite set of classes. E.g. predicting the functional family of a protein sequence. It is well known that sequence similarity of biological sequences often translates into structural or functional similarity. Therefore it is possible to use sequence classification algorithms to predict the structure or function of a protein [19]. Furthermore, sequence information can also be used to predict the sub-cellular location of a protein. Biological experiments have revealed that information required to guide a protein to a specific sub-cellular location is encoded in its amino acid sequence [46]. As another potential application of sequence classification techniques for biological data, we also attempt to predict the chromosomal location of a set of Expressed Sequence Tags(ESTs) from wheat. As explained in chapter 9, the goal is to predict whether a given EST is located close to the centromere of its chromosome or not. This can be useful in the development of Radiation-Hybrid maps for wheat, which will in turn assist in sequencing the wheat genome.

In this thesis, we study the use of frequently occurring substrings as features for sequence classification. A sequence is an ordered list of elements drawn from a set of distinct elements called an alphabet (A formal description is given in chapter 3). However depending on the classification problem at hand, some elements in the alphabet may be considered similar to each other so that substitution of one such element for another may not have an effect on the class specific properties of the sequence. Therefore, the pattern mining algorithm needs to be aware of these substitutions in order to discover useful patterns. Protein sequences are a good example where this situation arises.

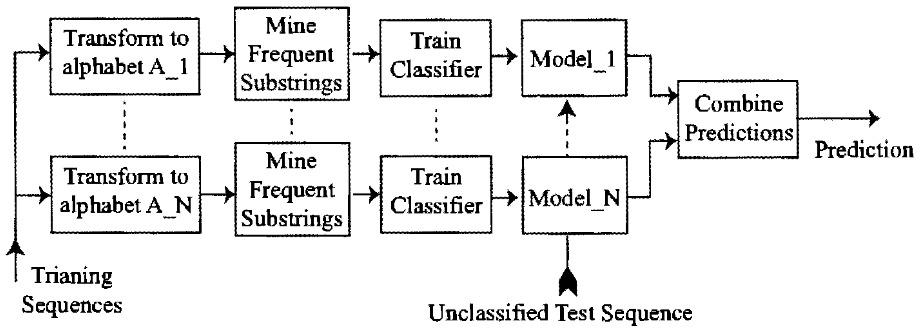
Proteins are made up of chains of amino acids each drawn from an alphabet of 20 distinct elements (naturally occurring amino acids). It's well known that certain sequence patterns are responsible for certain functions or localization of proteins and can be used to make predictions about them [46, 22]. It is also known that over the course of evolution, some amino acids in a protein could be substituted by others. Some of these substitutions change the properties of the protein while some do not. The latter are referred to as *conservative* substitutions. Typically, conservative substitution occur when an amino acid is substituted by another which has similar physiochemical properties [54]. From a pattern mining perspective, these substitutions present a challenging situation because sequence patterns that determine the function of a protein can contain substituted amino acids thereby causing difficulties in extracting useful patterns. As a solution to this problem, we propose the use of reduced alphabets formed by grouping similar amino acids together. We use alphabet reduction schemes presented in literature [39, 21] that are derived from a hierarchical clustering tree of the amino acids. The tree is formed by groping amino acids based on a pair-wise similarity measure so that those with very similar properties are grouped earlier than those with more distant properties. A more detailed description about alphabet reduction schemes is presented in chapter 4.

We introduce three frequent substring-based sequence classification algorithms which make use of reduced alphabets in addition to the full alphabet of 20 naturally occurring amino acids. Our motivation is to capture conserved sequence patterns even amidst different amino acids that may have substituted one another while preserving the properties of interest of a protein. Other sequence classification algorithms that take the similarity between amino acids in to account include, sequence distance-based algorithms which compute a pairwise sequence alignment using similarity matrices to score the alignment and Hidden Markov Model (HMM)-based algorithms. HMM-based algorithms usually require a multiple alignment of the input sequences which is a computationally expensive process and can also lead to ambiguities often requiring manual adjustment [10].

All three algorithms presented in our work consists of two main steps. 1) Mining frequently occurring substrings from the training sequences and transforming the sequences into a feature space using the substrings. 2) Training a classifier using the feature representation of sequences. As explained later in detail, frequent substrings are mined using different reduced alphabets and the three algorithms differ in the way these features are employed or depending on the classifier (Associative or SVMs) used. The first algorithm combines features mined from different alphabets and trains an association rule-based classifier on this combined feature space. The second algorithm trains an SVM-based classifier on the same feature space. Both of these approaches fall into the category of feature fusion in multiple source prediction. The third algorithm trains individual classifiers using frequent substrings mined from each alphabet and then combines the predictions made by all individual classifiers. This approach falls into the category of classifier fusion. The two basic approaches are shown in Figure 1.1. The details of combining frequent substrings mined under different alphabets and combining the predictions made by different classifiers are presented in chapter 7.


Transform to alphabet A_1
Mine Frequent Substrings
Transform to alphabet A_N
Mine Frequent Substrings
Combine Frequent Substrings
Train Classifier
Model
Prediction
Unclassified Test Sequence
Trianing Sequences

(a) Feature Fusion approach.



(b) Classifier Fusion approach.

Figure 1.1. Basic schematic diagrams of our algorithms. A_1, ... A_N are the different alphabets used.

CHAPTER 2. RELATED WORK

Previous work on sequence classification can be divided into three broad areas [52]. 1) Feature-based classification techniques that first transform the sequences into feature vectors and then apply conventional classification algorithms on them. 2) Sequence distance based techniques that utilize some distance function to compute the pair-wise similarity between sequences and then employ a nearest neighbor classification method. 3) Techniques that use models such as Hidden Markov Models and other statistical models for sequence classification.

Deshpande and Karypis [19] investigated sequence classification methods that fall into each of the three categories mentioned above for the classification of biological sequences. Their work includes SVM-based classification of sequences transformed into feature vectors, K-nearest neighbor classification using sequence alignment score as the pair-wise similarity measure and Markov Model-based classification. Their experiments reveal that SVM-based classifiers generally achieve higher accuracy compared to other methods.

Several studies have been performed on feature-based sequence classification. She et al. [46] used a feature space based on frequent substrings for the prediction of outer membrane proteins. They compared the performance of several conventional classification methods including SVMs, See5 and Association rules on this feature space. PSORTB [23] is an extension to this work which can make predictions for other sub-cellular locations as well. It also achieved better accuracy through the use of a Motif and Profile module and a BLAST module in addition to the SVM module used in [46]. A feature space consisting of more general subsequences was proposed by Lesh et al [34]. However, mining for frequent subsequences with unconstrained gaps is a very expensive process and does not scale well for long sequences. Leslie et al. [36, 35] used a K-gram based feature space to develop string kernels for use with

SVMs and Chuzhanova et al. [15] used K-grams as features for use with decision trees for genetic sequence classification.

Agrawal [8] approached the problem of sequence classification by using wavelet decomposition. Their method is capable of capturing both local and global classification behavior of sequences. It also benefits from the multi-resolution property of wavelet decomposition and therefore it can mine classification characteristics of sequences at different levels of granularity.

Sequence distance based classification schemes use a pair-wise distance function to measure similarity between sequences and apply a classification method such as the K-nearest neighbor classifier. The selection of the distance function plays a major role in these schemes. Work presented in [30, 51] uses Euclidean distance for classification of time series data. Euclidean distance makes sense only if the two sequences being compared have equal lengths. Therefore it is not useful for biological sequences in particular. Sequence-alignment-based distance functions [27] are more suitable for symbolic sequences such as protein or DNA sequences. String alignment kernel for use with SVMs proposed by Saigo et al. [42] can also be considered as a sequence distance based classification scheme.

Markov and Hidden Markov Models are widely used in sequence classification techniques that fall into the model based category. Yakhnenko et al. [53] applied a discriminatively trained k-order Markov model to classify protein and text sequence data. Srivastava et al. [48] used a profile HMM to classify biological sequences.

In general, using item taxonomies in sequential pattern mining was introduced by Srikant et al. [47]. Given a user defined taxonomy on items, it allows sequential patterns to contain items across all levels in the taxonomy. They modify each transaction by adding to each item, its ancestors in the taxonomy. Then they mine sequential patterns while making an attempt to minimize redundant patterns. Han et al. [24] explored the use of item taxonomies in association rule mining. Even though their focus was on frequent itemset mining, the proposed idea is useful for sequential pattern mining as well. They iteratively mine frequent itemsets (and eventually strong rules) at each level in the taxonomy in a top-down manner while reducing the support threshold as they proceed toward the lower levels.

Andorf et al [12] explored the use of reduced amino acid alphabets for classification of protein sequences into functional families. They constructed decision tree-based classifiers using motifs identified through multiple sequence alignments. Their evaluation shows that classification accuracies achieved by the use of motifs based on certain reduced alphabets are comparable to that of the 20-letter alphabet. However, they have not made an attempt to combine different alphabets. A Genetic Algorithm is used to produce different reduced alphabets in [40]. For each generated alphabet, N-peptide compositions (same as k-grams) are extracted from the training sequences and are used to train an SVM-based classifier. The final prediction is computed according to the mean rule.

Work presented in this thesis falls into the first category of sequence classifiers mentioned above. Our feature space is similar to the one used in [46]. Even though this thesis primarily focuses on protein sequences, our approach can in general be applied to any form of sequence data composed of symbols that have a taxonomic relationship between them so that reduced alphabets can be derived from the taxonomy.

CHAPTER 3. FREQUENT SUBSTRING MINING

Frequent substring mining is a special case of the well known sequential pattern mining problem. Given a collection of sequences, sequential pattern mining addresses the problem of discovering subsequences that occur frequently among them. For example, in a collection of customer purchase records for a computer accessories shop, the sequence, PC→ Modem →Webcam could be a frequent subsequence. A substring is a special subsequence where there are *no* gaps between adjacent elements.

We mine frequent closed substrings from the sequences in the training dataset which will serve as primary features for classification as described later in this thesis.

Definition 3.1. ***(Substring).*** *Let* $\Sigma = \{e_1, e_2 \ldots e_n\}$ *be an alphabet and* $S = a_1a_2 \ldots a_m$ *be a sequence over* Σ *where* $a_i \in \Sigma$ *for all* $1 \leq i \leq m$. *A sequence* $S' = b_1b_2 \ldots b_k$ *over* Σ *is said to be a substring of* S, *denoted by* $S' \subseteq S$, *if there exists some* $0 \leq c \leq m - k$ *so that* $b_j = a_{c+j}$ *for all* $1 \leq j \leq k$.

E.g. For an alphabet $\Sigma = \{A, B, C\}$, $S' = ABBC$ *is a substring of* $S = BCA\boldsymbol{ABBC}AB$

Definition 3.2. ***(Support).*** *Given a sequence dataset D, the support of a substring pattern P is given by,*

$$Support(P) = |\{S \in D | P \subseteq S\}|$$

For a given training sequence dataset D, the problem of frequent substring mining is to find the set of all substring patterns whose support is greater than or equal to a user specified support threshold λ. The goal is to find all elements of the set,

$$FS = \{P \mid (Support(P) \geq \lambda)\}$$

In most applications, the set of frequent substrings can be extremely large, especially when λ is low. Every substring of a frequent substring is also frequent and this is a major contributor towards generating a huge set of patterns. This problem can be alleviated to a great extent by mining for frequent *closed* substrings. A substring P is closed if there is *no* string P' so that $P \subseteq P'$ and $Support(P) = Support(P')$. The set of frequent closed substrings FCS, is defined as follows.

$$FCS = \{P \in FS \mid P\, is\, closed\}$$

FCS for the sequence dataset in Table 3.1 when $\lambda = 2$ is as shown below. Note that we ignore substrings with length=1. (Support is shown within parenthesis).

$$FCS = \{AC(4), ACBDF(3), BAEC(3), BAECE(2), CDBCA(3), DAB(3), EDA(2), FA(2)\}$$

Table 3.1. An example of a sequence dataset.

SID	Sequence
1	ACBDFBAECCDBCA
2	CDBCAFDAB
3	ACBDFABAECEDAB
4	BAECEFACDBCA
5	FEDACBDFEBBDAB

Once the frequent substrings are mined, the training sequences are converted into binary feature vectors or feature sets as shown in Table 3.2, where the rows correspond to sequences and columns (or the set elements in the feature set representation) correspond to frequent substrings. If a given sequence S_i contains the j^{th} frequent substring, then the j^{th} bit in the feature vector of S_i is set to 1 or equivalently, j is included in its feature set. If not, the bit is set to 0. Note that both of these

Table 3.2. Feature vector and feature set representations for the same sequences contained in Table 3.1.

SID	Feature Index							
	1	2	3	4	5	6	7	8
1	1	1	1	0	1	0	0	0
2	0	0	0	0	1	1	0	0
3	1	1	1	1	0	1	1	1
4	1	0	1	1	1	0	0	1
5	1	1	0	0	0	1	1	0

SID	Feature set
1	{1, 2, 3, 5}
2	{5, 6}
3	{1, 2, 3, 4, 6, 7, 8}
4	{1, 3, 4, 5, 8}
5	{1, 2, 6, 7}

Note:Features are indexed in the same order as they are listed in FCS

representations contain the exact same information.

From a pattern mining perspective, the set of closed substrings is a lossless compression of the full set of frequent substrings as the former can be used to derive the latter. It reduces the dimensionality of the feature space used for classification significantly without loosing any information from the *training* data. However, a substring P may be excluded from the feature space due to the existence of a superstring $P^{/}$ of P with the same support, while a test sequence may only contain P but not $P^{/}$. This could lead to misclassification of test sequences. Nevertheless, we use the set of closed substrings for the feature fusion-based classification algorithms as the set of all frequent substrings can become unmanageable large when substrings from all alphabets are combined. On the other hand, we use the set of all frequent substrings for the classifier fusion-based algorithm as it treats substrings from different alphabets individually. The algorithm proposed in [37] is used to mine frequent closed substrings from the sequences in the training database. The original algorithm is for mining gap-constrained closed sequential patterns. Hence we set the gap-constraint to zero for mining substrings. Note that this algorithm directly mines closed frequent subsequences *without* having to generate the frequent subsequences first and then eliminate the non-closed subsequences later. However, we disable the closure checking functionality when we need the set of all frequent subsequences.

CHAPTER 4. REDUCED ALPHABETS

4.1. Introduction

As described previously, sequences are defined over a finite alphabet of distinct elements. However, there are situations where distinct elements may have considerable similarity with each other and could be treated as the same element. Let $\Sigma = \{A, B, C, D, E, F\}$ be an alphabet. For example, if A represents "Margarine" and B represents "Butter", then in most general applications A and B could be treated as identical elements. This means pattern $CD\mathbf{A}F$ could be treated the same as $CD\mathbf{B}F$. Therefore it's desirable to group elements A and B together and construct a reduced version of the alphabet in order to discover such patterns. On the other hand, if one is interested in distinguishing between customers who buy "Margarine" and those who buy "Butter", then this grouping is undesirable. Further, there can be situations where the appropriate level of reduction is not known a priori. One good way of handling these multiple levels of granularity is to consider different reduced alphabets where elements have been grouped through hierarchical clustering.

Provided there is some function to calculate the pairwise similarity between distinct elements, an agglomerative hierarchical clustering scheme can be used to form reduced alphabets as follows. In the first iteration, group the two most similar elements together. In each subsequent iteration, merge the two most similar clusters (a cluster can be a single element) and continue this process until the desired number of groups are formed. Similarity between clusters can be measured using single, complete or average linkage. Theoretically, elements can be clustered until all of them are assigned to a single group. However, in most practical applications, the process will be terminated before that stage because a single element alphabet is not useful.

Figure 4.1 shows a hypothetical hierarchical clustering tree for the alphabet Σ mentioned above. According to this tree, elements A and B are the two most similar elements in Σ, thus they get merged into a group at the lowest level of the tree, followed by D and E. F is distantly similar to either D or E and therefore join the group (DE) at a higher level of the tree. The two groups (ABC) and (DEC) exhibit very little similarity to elements in each other and therefore remain as separate groups until the root of the tree. Table 4.1 shows how a sequence can be represented using the different alphabets.

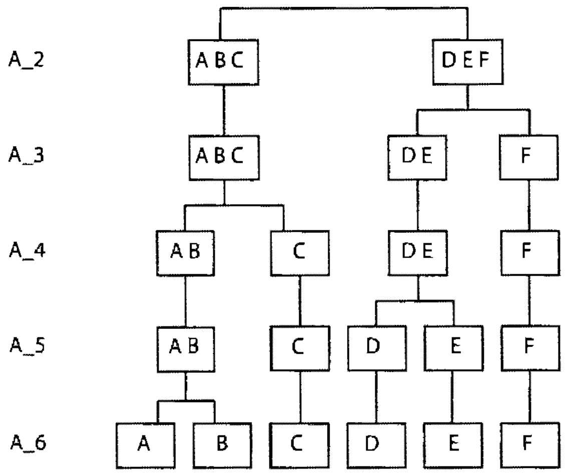


Figure 4.1. Hierarchical clustering tree of reduced alphabets.

Table 4.1. An example of a sequence, transformed into the different alphabets shown in Figure 4.1 (A group is represented by its first element).

Alphabet	**Sequence**
A_6	ACBDFBAECCDBCA
A_5	ACADFAAECCDACA
A_4	ACADFAADCCDACA
A_3	AAADFAADAADAAA
A_2	AAADDAADAADAAA

4.2. Reducing the Amino Acid Alphabet

There are twenty naturally occurring amino acids that serve as building blocks in forming protein chains. However, some of these amino acids have similar properties and have been found substituting one another with only minor effects on the final topology of a protein which can consequently result in conserved function. The multiple Alanine substitutions are among the most well known examples [21]. Substitution matrices such as PAM [18] and BLOSUM [25] provide quantitative values for these similarities. They have been developed based on the frequencies of amino acid substitutions observed in aligned protein sequences and reflect both evolutionary and functional similarity between amino acids [31].

The similarity between amino acids can be used to group similar ones together and derive reduced alphabets. Use of reduced alphabets has been extensively studied in relation to protein folding in particular [39, 16, 32, 41]. Further, Andorf et al. [12] and Albayrak et al. [10] studied the use of reduced alphabets in classification and clustering of proteins based on their functions, respectively. We study the use of reduced alphabets and combinations of them, in mining substring patterns from protein sequences. Our assumption is that it would allow us to find conserved patterns among protein sequences that may not necessarily have identical amino acid compositions under the full alphabet of twenty residues but will become identical when transformed to a reduced alphabet. For the work presented in this thesis, we use the sets of reduced amino acid alphabets derived by Murphy et al. [39] and Etchebest et al. [21] In [39], Similarity between a pair of amino acids is expressed in terms of a correlation coefficient based on the BLOSUM50 substitution matrix. As an example, the correlation coefficient for the two residues W and Y is given by,

$$C_{WY} = \frac{\sum_{i=1}^{20} M_{W,i} M_{Y,i}}{\left(\sum_{i=1}^{20} M_{W,i} M_{W,i}\right)\left(\sum_{i=1}^{20} M_{Y,i} M_{Y,i}\right)}$$

Where, $M_{n,m}$ is the similarity value between the n^{th} and m^{th} amino acids as found in the similarity matrix. The summation of i is taken over the 20 amino acids.

The correlation coefficient is used as a pair-wise similarity measure to group similar amino acids together. First, the two amino acids with the highest correlation coefficient are grouped together. Then the two with the next highest correlation coefficient are considered for grouping. If none of them is already in a group, then they are assigned to a new group. If one of them is already in a group, then the other one is assigned to that same group. This process continues until the desired number of groups is formed. Figure 4.2 shows some of the reduced alphabets derived. Please refer to [39] for a more complete description.

Etchebest et al. [21] derived reduced alphabets based on amino acid distribution observed in structural motifs called protein blocks. Use of protein blocks incorporates the influence of local protein structures into the construction of the alphabets. This is useful since it is well established that protein structures can be represented as a combination of local structures and results in a more complete description than classical secondary structures [21]. The authors define a distance measure which is based on the distribution of different amino acids in the protein blocks and it is then used to generate a hierarchical clustering tree of reduced alphabets as shown in Figure 4.3. please refer to [21] for more details.

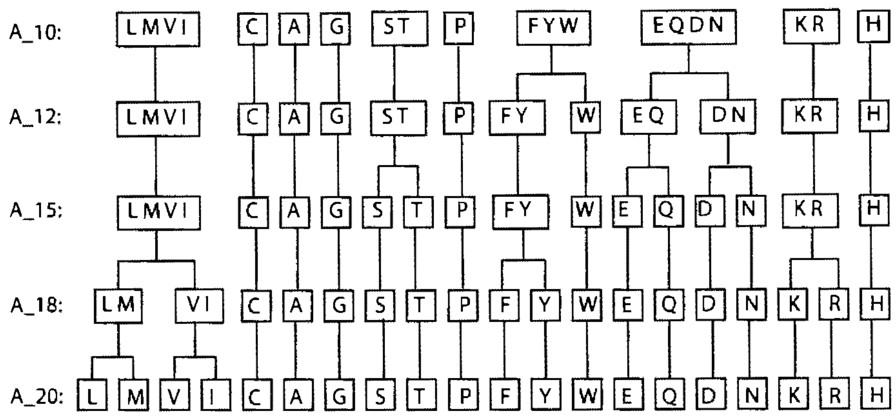


Figure 4.2. Reduced alphabets derived by Murphy et al. [39].

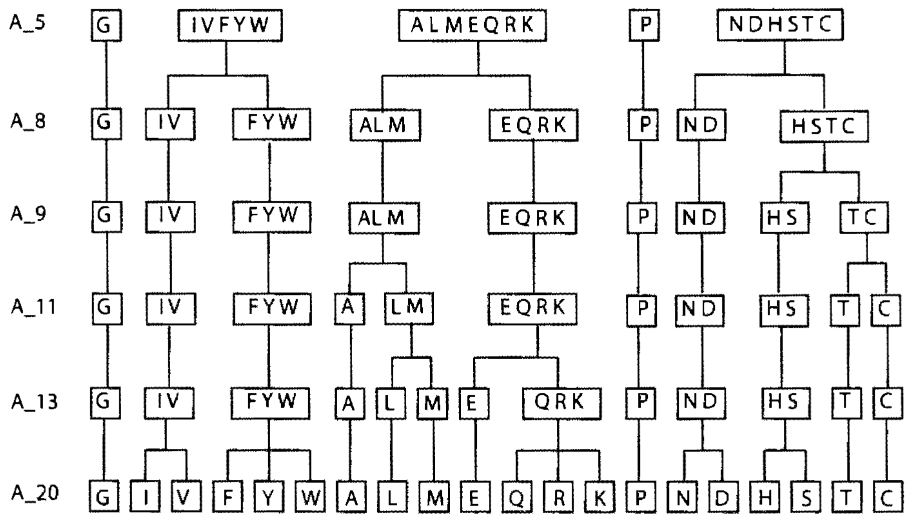


Figure 4.3. Reduced alphabets derived by Etchebest et al. [21].

CHAPTER 5. ASSOCIATION RULE-BASED CLASSIFICATION

5.1. Introduction and Preliminary Concepts

Association rule mining (ARM) is one of the most important tasks in data mining. It has been used in many contexts since its introduction by Agrawal et al. [9]. It was initially proposed to facilitate Market Basket Research (MBR). Since then it has found applications in areas such as Software engineering, Bioinformatics and Precision agriculture as well.

In typical association rule mining, a rule of the form $X \Rightarrow Y$ relates two disjoint sets of items X and Y where X is called the antecedent and Y is called the consequent. For example in MBR this could mean, customers who buy X also tend to buy Y. The statistical strength of a rule is ubiquitously expressed in terms of support and confidence where the former is the number of transactions containing both the antecedent and consequent while the latter is the fraction of transactions that contain the antecedent that also contain the consequent. However, in association rule-based classification, associations are drawn between itemsets and classes.

Conventionally, association rule mining is done in two steps. 1) Generating all itemsets that exceed a minimum support threshold (frequent itemset mining) 2) Deriving association rules from the set of frequent itemsets mined in step 1.

Given a set of distinct items $I = \{i_1, i_2, \ldots i_m\}$ and a transaction database $D = \{T_1, T_2, \ldots T_N\}$ where each $T_i \subseteq I$, the problem of frequent itemset mining is to find the set I_f of subsets of I so that, $I_f = \{X \subseteq I | Support(X) \geq \sigma\}$ where,

$$Support(X) = |\{T_i \in D | X \subseteq T_i\}|$$

and σ is a user defined minimum support threshold.

In our case, the transactions are the training sequences converted into the feature space of frequent closed substrings mentioned in chapter 3. Therefore the items refer to the indices of the frequent substrings. I.e $I = IndicesOf(FCS)$. This means, an item corresponds to a frequent substring and therefore an itemset corresponds to a collection of frequent substrings which can be thought of as a sequence signature. Table 5.1 contains an example transaction database where each row represents a transaction. Note that the transaction database is constructed in the same manner as shown in Table 3.2.2. However, we have constructed a new example different from the one shown in Table 3.2.2 in order to be more illustrative of the specifics of this section.

5.2. Learning Association Rules for Classification

In a binary classification problem the transaction database contains both positive and negative training instances. In general, if we assume D contains transactions belonging to K classes; $c_1, c_2 \ldots c_K$ then, a frequent itemset X can be used construct classification rules of the form,

$$R : X \rightarrow c_i, \quad for\, 1 \leq i \leq K$$

If D^{C_i}, contains all the transactions belonging to class label c_i, the confidence of a rule $R : X \rightarrow c_i$ is defined as,

$$Conf(R : X \rightarrow c_i) = \frac{|\{T_i \in D^{C_i} | X \subseteq T_i\}|}{|\{T_i \in D | X \subseteq T_i\}|}$$

Table 5.2 lists *some* of the frequent itemsets along with the confidence values of the rules they generate, as derived from the transaction database contained in Table 5.1. E.g. $Conf(\{1, 5\} \rightarrow -1) = \frac{2}{3} = 66.7\%$

Table 5.1. An example transaction database.

Transaction	**Items**	**Class**
T_1	{1, 3, 4, 6, 7}	+1
T_2	{1, 4, 5}	+1
T_3	{3, 4, 6, 7}	+1
T_4	{1, 3, 5, 6, 7}	-1
T_5	{1, 3, 6, 7}	-1
T_6	{1, 2, 3, 5, 6}	-1
T_7	{2, 3, 5, 6}	-1

Table 5.2. *Some* of the frequent itemsets and their corresponding classification rules derived from the transaction database in Table 5.1 at $\sigma = 2$. Shown in boldface are *all* the rules generated by HARMONY [50].

	Itemset	**Support**	**Conf(R : X → +1)%**	**Conf(R : X → −1)%**
1	{1, 3, 5, 6}	2	0	100
2	{1, 4}	2	100	0
3	{1, 5}	3	33.3	66.7
4	**{1, 6}**	**4**	**25**	**75**
5	{3, 4}	2	100	0
6	{3, 7}	4	50	50
7	**{4}**	**3**	**100**	**0**
8	{4, 6}	2	100	0
9	{5}	4	25	75
10	**{5, 6}**	**3**	**0**	**100**

Typically, frequent itemset mining generates a large number of itemsets. However, only the high confidence itemsets are useful for constructing effective classifiers. E.g. the itemset {3, 7} is equally associated with both classes with a confidence of 50% and is therfore *not* a useful rule for classification. Further, if a confidence threshold is used (E.g. 90%), then itemsets such as {1, 5}, {1, 6} and {5} are also *not* useful. Therefore, mining all frequent itemsets and then selecting the confident rules is not an efficient solution and is often prohibitive due to the excessive computational cost of frequent itemset mining as well as rule selection from a large number of candidates.

In order to avoid the above mentioned inefficiencies in the conventional approach, we use HARMONY [50], an efficient algorithm which directly mines high-confidence covering rules for classification. Given a minimum support threshold σ, HARMONY will find high quality classification rules ensuring that for each training instance, the highest confidence rule covering that instance is included in the final rule set. By default, HARMONY does not use a minimum confidence threshold. However, users can specify a confidence threshold so that only those rules with confidence exceeding the threshold are included in the final classification rule set(But this may cause some training instances not to have any rules covering them in the final rule set). Please refer to [50] for more details.

The classification rules generated by using the HARMONY algorithm on the same transaction database shown in Table 5.1 are shown in boldface in the same Table. Note that it generates only three itemsets but covers all the training instances and includes the highest confidence rule for each instance. In spite of pruning certain itemsets that would otherwise be generated by the conventional approach, and culminating in a much more compact set of rules, HARMONY still preserves the most useful and representative rules for classification. E.g. the itemset $\{1, 3, 5, 6\}$ covers transactions T_4 and T_6 and generates a rule with confidence of 100% towards the negative class. But both of these transactions are covered by the itemset $\{5, 6\}$ which has the same confidence but a higher support. Therfore $\{5, 6\}$ is prefered over $\{1, 3, 5, 6\}$ and is potentially more useful.

Once the set of classification rules has been mined, they are ordered in the confidence descending order. Support descending order is used to order rules with the same confidence. This ordering will provide efficient access to the classification rules when the developed model is used to classify new test instances.

As mentioned before, the body (itemset) of a classification rule represents a collection of frequent substrings which we refer to as a sequence signature, following a similar concept as in [20]. The usefulness of sequence signatures go beyond pure classification purposes as they can reveal important sequence segments that are responsible for the specific properties of a family of related sequences. For example, for protein sequences, these could mean sequence motifs or functional domains. However, since these signatures do not impose a limit on the gap between two frequent substrings, they can capture patterns that are wide spread along a sequence which generalizes the concept of sequence signatures beyond traditional motifs or domains.

5.3. Classification of a New Instance

A new test instance T_i, is classified according to the most confident classification rule it covers. T_i covers a rule $R : X \rightarrow c_i$ if $X \subseteq T_i$.

$$Classlabelof(T_i) = arg_{c_i}max(Conf(R : X \rightarrow c_i)) : X \subseteq T_i$$

E.g. The transaction; $\{1, 2, 4, 6\}$ will be classified as positive by the rule $R : \{4\} \rightarrow +1$ with 100% confidence.

CHAPTER 6. SUPPORT VECTOR MACHINE-BASED CLASSIFICATION

A support vector machine(SVM) [49] is a supervised machine learning technique that is applicable to both classification and regression tasks. SVMs have gained much popularity in recent years and have found applications in many different fields including, text categorization, image recognition and bioinformatics.

SVMs are extensively used in bioinformatics for sequence classification. Deshpande and Karypis [19] investigated the use of several widely used classification methods on biological sequence data. Their results prove classifiers based on SVMs outperform other popular classification methods such as those based on Markov models and K-nearest neighbors. She et al [46] compared See5 [7] (improved version of C.4.5), association rules, Hidden Markov Model and SVM based classifiers in their study of outer membrane proteins and showed that SVM based classifiers produce the best results. Furthermore, SVMs are particularly suitable for situations with high dimensional feature spaces which is exactly the case in our work.

The following discussion is a very brief introduction to the theory of SVMs. Most of the material in the following section is based on [45].

Given a sample S of training data points (training examples) defined in a space $X \subseteq \mathbb{R}^n$, SVMs attempt to find a hyperplane which separates the positive and negative examples in S. Let $S = \{(x_1, y_1), (x_2, y_2), \ldots (x_l, y_l)\}$, where $x_i \in X$ is the n-dimensional feature vector of the i^{th} training example and $y_i \in \{+1, -1\}$ is its class label, for each $i = 1, 2 \ldots l$. The goal of a linear classifier is to find a hyperplane $\mathbf{w.x} + b = 0$, such that all positive examples in S reside on one side and the negative examples on the other as shown in Figure 6.1. In the previous equation, $\mathbf{w} \in \mathbb{R}^n$ is referred to as the weight vector and it defines a direction perpendicular to the hyperplane where as $b \in \mathbb{R}$ is called the bias and it determines the distance to the hyperplane from the origin. A separating hyperplane must satisfy the following condition. However, this is possible only if the training data is linearly separable.

$$y_i(\mathbf{w}.\mathbf{x_i} + b) \geq 0, \ \forall i = 1, 2, \ldots l \tag{1}$$

When the classifier is being used to predict the class label a new test instance x_i, it is classified as positive if $(< \mathbf{w}.\mathbf{x} > + b) > 0$. Otherwise, it is classified as negative.

Definition 6.1 (Functional margin). *Functional margin of an example (x_i, y_i) w.r.t a hyperplane characterized by $(\boldsymbol{w}, b)$ is given by,*

$$\gamma_i = y_i(\mathbf{w}.\mathbf{x_i} + b)$$

Functional margin of a hyperplane $(\boldsymbol{w}, b)$ w.r.t a training sample S is given by,

$$\gamma = min_{i \in S}\{y_i(\mathbf{w}.\mathbf{x_i} + b)\}$$

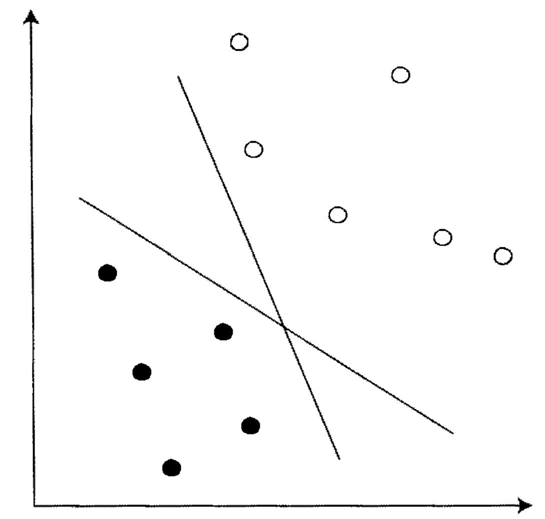
Definition 6.2 (Geometric margin). *Geometric margin of an example (x_i, y_i) w.r.t a hyperplane $(\boldsymbol{w}, b)$ is given by,*

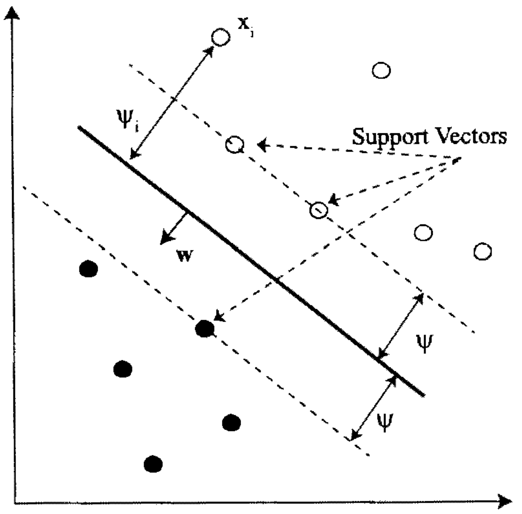
$$\psi_i = \frac{y_i(\mathbf{w}.\mathbf{x_i} + b)}{|w|}$$

Geometric margin of a hyperplane $(\boldsymbol{w}, b)$ w.r.t a training sample S is given by,

$$\psi = min_{i \in S}\left\{\frac{y_i(\mathbf{w}.\mathbf{x_i} + b)}{|w|}\right\}$$

There are many different separating hyperplanes for a linearly separable training dataset as shown in Figure 6.1. 1(a). SVMs look for the one with the maximum geometric margin which is termed as the *maximal margin hyperplane.* A hyperplane


(a) Two of many possible separating hyperplanes for linearly separable data.



(b) Maximal margin hyperplane. $\psi = 1/|\mathbf{w}|$ for a canonical hyperplane.

Figure 6.1. Demonstration of linear SVMs for 2-dimensional data.

characterized by $(\mathbf{w}, b)$ has an inherent degree of freedom so that it can be rescaled into $(\lambda\mathbf{w}, \lambda b)$ for some $\lambda \in \mathbb{R}^+$ without changing its geometry. The rescaling has no effect on the geometric margin but it does affect the functional margin. Therefore, the maximal margin hyperplane is found by fixing the functional margin to 1 and then minimizing $|\mathbf{w}|$. Hyperplanes with a functional margin of 1 are referred to as *canonical hyperplanes.* The solution to the following optimization problem yields the maximal margin hyperplane, $(\mathbf{w}, b)$ with geometric margin $\psi = 1/|\mathbf{w}|$ [45].

$$minimize_{\mathbf{w},b} \quad < \mathbf{w}.\mathbf{w} >$$
$$subject\ to\ constraint: \quad y_i(\mathbf{w}.\mathbf{x_i} + b) \geq 1$$

Through out the above discussion we assumed that the training data is linearly separable. However, this is typically not the case with real datasets as they can contain outliers due to noise. Therefore, the classification model may need to ignore such training examples. This leads to the development of *soft margin* SVMs that allow a few training examples to be on the wrong side of the separating hyperplane.

This is achieved by introducing slack variables(ξ_i) into the optimization problem as shown below.

$$minimize_{\xi,\mathbf{w},b} \quad <\mathbf{w}.\mathbf{w}> + C\sum_{l}^{i=1}\xi_i^2$$

$$subject\ to\ constraint: \quad y_i(\mathbf{w}.\mathbf{x_i} + b) \geq 1 - \xi_i$$

C is called the regularization parameter and it controls the trade-off between margin size and training errors. In practice, a value for C is obtained by assessing the performance of the classifier by using a separate validation set or through internal cross-validation within the training set.

While soft margin SVMs handle outliers in noisy data, kernel functions can be used to map linearly inseparable data into a higher dimensional space where it could become linearly separable. This is sometimes referred to as the *kernel trick.* Popular kernel functions include; polynomial, Radial Basic Function (RBF) and Sigmoid.

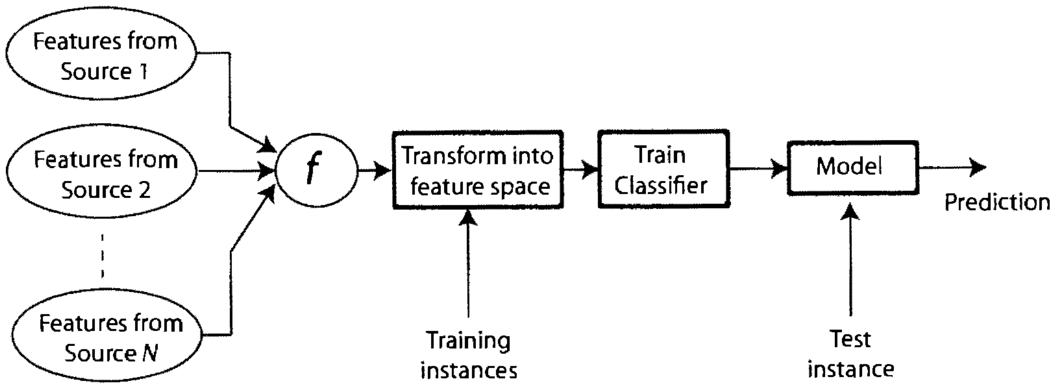
CHAPTER 7. FEATURE VS CLASSIFIER FUSION APPROACHES TOWARDS SEQUENCE CLASSIFICATION

7.1. Introduction

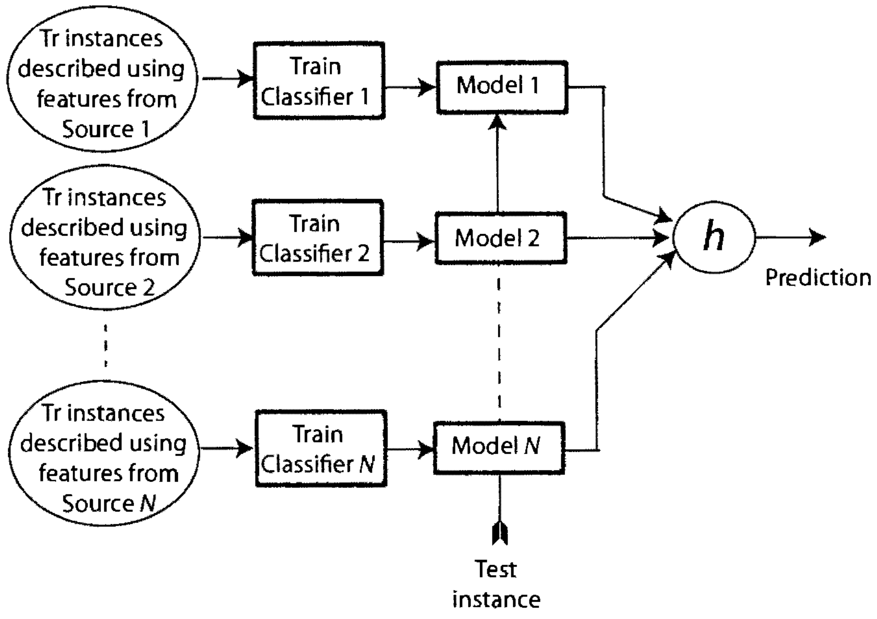
In situations where information from multiple sources are used for building predictive models, there must be some way of combining this information in order to make the final prediction. Feature fusion and classifier fusion are two fundamental approaches towards addressing this problem. Feature fusion refers to combining the sets of features obtained from individual sources into a single feature set, describing each training instance using this combined feature set and then training a single classifier. In contrast, classifier fusion refers to training a set of individual classifiers using training instances described by features obtained from each individual source and then combining the predictions made by these classifiers to generate the final prediction for a test instance. In our work, the different alphabets serve as multiple sources of information.

Figures 7.1 (a) and (b) demonstrate the difference between the two approaches. f and h are the functions that perform the actual feature or classifier fusion respectively. In our case, f performs the operation of concatenating feature vectors from selected reduced alphabets and h is a meta classification model. Both of these will be explained in detail in their respective sections.

Notation: For the rest of this thesis, we will use A_f to denote the complete set of alphabets used in a given experiment. In our experiments A_f is equal to the set of alphabets listed in either Figure 4.2 or Figure 4.3. E.g. referring to Figure 4.2, $A_f = \{A_{20}, A_{18}, A_{15}, A_{12}, A_{10}\}$.



(a) Feature Fusion. f is some function which combines the features.


Tr instances described using features from Source 1
Train Classifier 1
Model 1
Tr instances described using features from Source 2
Train Classifier 2
Model 2
h
Prediction
Tr instances described using features from Source N
Train Classifier N
Model N
Test instance

(b) Classifier Fusion. h is some function which combines the predictions.

Figure 7.1. A diagrammatic view of feature Vs classifier fusion.

7.2. Feature Fusion Approach

While following the feature fusion approach to multiple source classification, we combine features (frequent substrings) mined from different reduced alphabets and train a classifier using the combined set of features.

The most straight forward way of combining features would be to concatenate feature vectors corresponding to each alphabet in A_f. However, we learned that this is not an effective solution as addition of features from certain alphabets aggravated the performance of the classifier. Further, the influence of features from a given alphabet on the performance of the classifier is dataset or even class dependent. Therefore, in a given classification experiment, we use a subset A_s of A_f. The procedure for selecting alphabets and corresponding support thresholds to mine frequent substrings from sequences transformed into them, is explained in the next section. For each selected alphabet, we reduce the sequence database to the corresponding alphabet and mine for frequent substrings. Once mining is complete for all selected alphabets, we collect all the frequent substrings and remove duplicates. Using this combined set of frequent substrings as features, we then represent the sequences in the training dataset as binary feature vectors or feature sets in the same manner as described in Chapter 3. Finally, a classification model is developed using these feature vectors or feature sets. Any feature-based classification algorithm can be used to build the classification model. In this thesis we used either Association rules or Support Vector Machines (SVM). Figure 7.2 contains the pseudo code for this process.

7.2.1. Parameter Estimation

Selection of which alphabets to use and estimation of a suitable minimum support threshold plays an important role in the final classification accuracy achieved by our algorithm. We refer to this process as the parameter estimation procedure. By parameters, we mean the set of selected alphabets(A_s) and a support threshold λ_i for each $a_i \in A_s$. Suppose we have a total of N different alphabets to choose from, then we have a total of $2^N - 1$ possible alphabet selections. E.g. there are 1023 different alphabet selections available for N=10. In the most flexible case, the support threshold can take a value anywhere between 0-100% of the dataset size and can be independantly chosen for each selected alphabet. Therfore selecting a subset from all available alphabets and slecting minimum support thresholds for them is an extremely difficult task and exploring all possible combinations is not an option.

SUBROUTINE 1: Learn (D, A_{sp})

Input: D - Training database
Input: A_{sp} - A set of tuples (a_i, λ_i), where a_i is an alphabet and λ_i is the minimum support threshold used to mine frequent substrings from D transformed into a_i.
Output: *Model* - The classification model learned

```
1. $FCS = \phi$
2. For each $(a_i, \lambda_i) \in A_{sp}$
3.     $D_{a_i} = D$ transformed into alphabet $a_i$
4.     $FCS_{a_i}$ = Frequent Substrings mined from $D_{a_i}$ with min support = $\lambda_i$
5.     $FCS = FCS \cup FCS_{a_i}$
6. $FCS = Remove\_Duplicates(FCS)$
7. $feature\_vectors$ = sequences in $D$ transformed into feature vectors using $FCS$
8. $Model$ = Classifier trained with feature_vectors
9. return $Model$
```

Figure 7.2. Building the classification model.

In order to make this alphabet selection and support threshold estimation process more manageable, we use a greedy approach as described by the pseudo code in Figure 7.3. This process runs in several rounds. In the first round, it selects the alphabet that yields the best classification result and adds it to a set of selected alphabets(A_s). In each subsequent round, it finds the alphabet which gives the best result when combined with the alphabets already in the selected set and adds it to the selected set. As explained in the next paragraph, by "classification result", we mean the F1 value obtained by internal cross validation within the training set. The greedy nature of this process may cause the selected parameter values to deviate from the absolute optimal values. However, as for our experience, it provides parameter values that result in reasonably good classification accuracy.

The optimization sub-routine outlined in Figure 7.4 is responsible for selecting a near optimal minimum support threshold (λ) for a given alphabet (either by itself or in combination with already selected alphabets). It performs a k-fold cross validation (E.g. 3-fold) within the training dataset in order to obtain classification results for parameter estimation. Please note that this cross validation is separate from the cross validation we use in the experimental evaluation of our algorithm in section 8.2 and should not be confused with each other. We use the F1 measure as the objective measure and try to maximize it using a hill climbing approach with multiple restarts. Lines 1-3 are used to skip the optimization if required and its use will be explained later in this section.

Definition 7.1. *(Precision, Recall and F1 measure)*
Let, TP(True Positives) - Number of test instances (test sequences in our case) predicted as positive and are actually positive.
FP(False Positives) - Number of test instances predicted as positive but are actually negative.
TN(True Negatives) - Number of test instances predicted as negative and are actually negative.
FN(False Negatives) - Number of test instances predicted as negative but are actually positive. Then,

$$Precision = \frac{TP}{TP + FP} \quad Recall = \frac{TP}{TP + FN}$$
$$F1 = \frac{2.Precision.Recall}{Precision + Recall}$$

The loop starting at line 4 of Figure 7.4 varies the value of λ for each iteration until the F1 value reaches a maximum. We use a hill climbing approach to update λ. Note that there is a possibility of the F1 value reaching a local maxima due to the inherent nature of the hill climbing approach. However, we use multiple restarts as an attempt to reduce the possibility of a local maxima. For each fold in the cross validation, line 6 builds the classification model and line 7 makes the predictions for the sequences in the secondary testing dataset. Line 8 calculates the overall F1 value using the predictions made in all folds.

Figure 7.3 contains the pseudo code for the complete algorithm, including parameter estimation and construction of the final classification model. Lines 4-9 describe the process of discovering the best alphabet in a given round. In the very first round where there are no alphabets selected yet, i.e. $A_s = \phi$, this refers to finding the alphabet in A_f which gives the best result (highest F1 value) on its own. In subsequent rounds, it finds from the set of alphabets which haven't been selected yet $(A_f - A_s)$, the one that gives the best result when combined with the alphabets already in the selected set(A_s). Line 10 checks if the best alphabet found in the current round actually improves the overall best result achieved so far. If it does, it is added to the selected set and its support threshold is saved for future use and the process moves on to the next round. If it doesn't, the parameter estimation process terminates. Even though our parameter estimation process is capable of determining a near optimal support threshold for each selected alphabet independently of others, for efficiency reasons, we fix the support threshold at a constant value after the initial round. More specifically, in the very first round we select the best alphabet and its optimal support threshold. Then we keep the support threshold constant at this value and prohibit the optimization subroutine from changing it in subsequent rounds. This is achieved at line 14, where λ_c is assigned the optimal support threshold of the first selected alphabet which will cause the optimization sub-routine not to change it thereafter. Line 16 calls the *Learn* subroutine along with the parameters estimated as described above, to build the final classification model.

ALGORITHM 1:

Input: D_{tr} - Training sequence database.
Input: A_f - Set of all alphabets.
Output: $Model$ - Classification Model.

```
1.  $A_s = \phi$,    $A_{sp} = \phi$        //Initialization
2.  $\lambda_c = 0$
3.  $global\_best\_F1 = 0$
4.  $current\_best\_F1 = 0$
5.  **For** each alphabet $a \in A_f - A_s$
6.      $(F1, \lambda) = Optimize(a, A_{sp}, \lambda_c, D_{tr})$
7.      **If** $F1 > current\_best\_F1$
8.          $current\_best\_F1 = F1$
9.          $(a_s, \lambda_s) = (a, \lambda)$
10. **If** $current\_best\_F1 > global\_best\_F1$
11.     $global\_best\_F1 = current\_best\_F1$
12.     $A_s = A_s \cup \{a_s\}$
13.     $A_{sp} = A_{sp} \cup \{(a_s, \lambda_s)\}$
14.     $\lambda_c = \lambda_s$
15.     Goto line 4
16. Model=Learn($A_{sp}, D$)
17. return Model
```

Figure 7.3. Main algorithm.
Lines 1-15 account for parameter optimization. Line 16 builds the final classification model

SUBROUTINE 2: Optimize(a, A_{sp}, λ_c, D)

Input: a - Alphabet for which the optimal support threshold is being estimated.
Input: A_{sp} - The set of already selected alphabets and their respective support thresholds.
Input: λ_c - The value of the support threshold if it's fixed at some value. 0 if not.
Input: D - Input sequence database.
Output: $(F1, \lambda)$ - Where $F1$ is the best result achieved and λ is the support threshold that produced the best result.

```
1. If $\lambda_c \neq NULL$
2.     $\lambda = \lambda_c$
3.     Goto line 5  //No optimization required. $\lambda$ is fixed.
4. Loop while changing $\lambda$ until $F1$ reaches maximum
5.     For $i = 0$ to Folding
6.         $Model = Learn(D_{tr}^i, A_{sp} \cup \{(a, \lambda)\})$
7.         $Predictions = Predictions \cup Test(D_{tst}^i, Model)$
8.     $F1 = Calculate\_F1(Predictions)$
9. return $(F1, \lambda)$
```

Figure 7.4. Optimizing the support threshold for a given alphabet.

7.3. Classifier Fusion Approach

In this section we describe an ensemble of SVM-based classifiers, each trained with features mined with respect to a distinct alphabet. The predictions made by individual classifiers are then combined using a meta classifier to obtained the final result. This approach of combining classifiers is commonly referred to as *stacking* in the Machine Learning community. Seewald [44] showed that other schemes of classifier fusion such as Voting, Selection by cross-validation, Grading and Bagging can also be reformulated as stacking.

Figure 7.5 shows the schematic diagram of our approach. First, the training sequences are transformed into the different alphabets used in the experiment. Note that unlike in the feature fusion approach, in this case we do not perform any alphabet selection. Instead, we use the complete set of alphabets(A_f). In phase-1, for each alphabet, the primary training sequences are further split into secondary training and testing sets and frequent substrings are mined from the secondary training set. Using the frequent substrings as features, an SVM model is trained on the secondary training set and predictions are made for the sequences in the testing set. We use k-fold cross validation to split the primary set of training sequences into secondary training and testing sets mentioned above. Therefore, this cross validation process provides predictions for each sequence in the primary training dataset. If there are N alphabets, this results in N predictions for each sequence. A prediction is expressed as the distance from the separating hyper-plane of the corresponding SVM model. Higher the distance, the more likely the prediction is correct.

We use the predictions of the Phase-1 SVMs to train a meta SVM model as follows. The predictions made by the Phase-1 SVMs transform each sequence in the primary training set into an N-dimensional feature vector where the i^{th} element is the prediction made by the i^{th} Phase-1 SVM model. For a given training sequence, if all N predictions made by the Phase-1 SVMs are in favor of the same class, we consider it as a confident prediction and do not further process it. On the other hand, if at least one prediction differs from the rest, we note there is some disagreement between the classifiers and use these vectors to train the meta SVM model. Our belief is that the meta SVM model will learn how to treat instances where the individual Phase-1 classifiers have provided conflicting predictions. (Please note that the filter for detecting conflicting predictions is not shown in the figure due to space limitations)

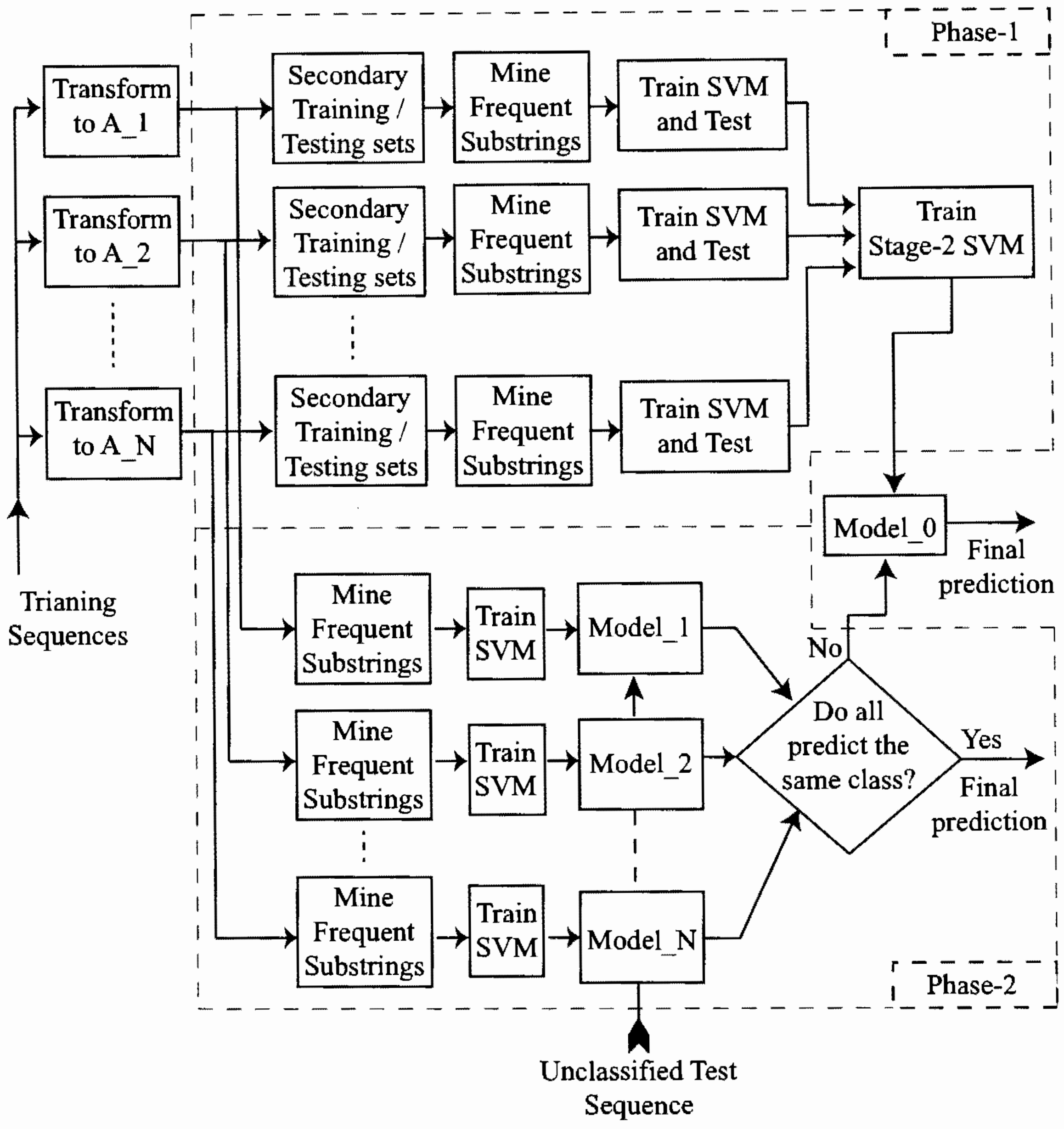


Figure 7.5. Schematic diagram of the ensemble of classifiers.

In Phase-2, for each alphabet, we mine frequent substrings from the complete set of primary training sequences and train N, SVM models. When a prediction needs to made for an unclassified test sequence, we consult each of the Phase-2 SVM models. Following the same reasoning as in Phase-1, if all models have predicted in favor of the same class, we let this be the final prediction. If at least one model has predicted a different class compared to the rest, then, using the N-dimensional feature vector produced by the Phase-2 SVM models, we consult the meta SVM model built in Phase-1 ($Model_0$ as shown in Figure 7.5) to generate the final prediction.

In this approach, we do *not* try to find an optimal support threshold for frequent substring mining. Instead, we use a constant support threshold for all alphabets. If necessary, the same techniques mentioned in the previous sections can be used to optimize the support threshold for each alphabet. However it will significantly increase the running time of the algorithm. Our experiments show that having a constant support threshold produces acceptably good results when using the classifier fusion approach.

CHAPTER 8. EXPERIMENTAL EVALUATION

8.1. Datasets

We used four datasets for the experimental evaluation of our algorithms. The first two are protein sub-cellular location datasets corresponding to Gram-negative and Gram-positive bacteria respectively. The third and fourth datasets contain protein sequences assigned with Gene Ontology terms based on their molecular function.

The two sub-cellular location datasets were obtained from the PSORTb v.2.0 [23] database. The location of each protein in these datasets has been experimentally verified and reported in literature, thereby making them very reliable. The Gram-negative bacteria dataset contains 1444 singly located proteins residing at five different location sites where as the Gram-positive dataset contains 541 singly located proteins at four location sites.

The third protein dataset consists of subfamilies of peptidase. Peptidase, also known as protease is a clinically important group of enzymes (proteins that catalyze chemical reactions) whose function is to break down proteins and are utilized in many metabolic processes of an organism. E.g. digestion of proteins contained in foods. Study of protease is also important in developing protease inhibitors for antiretroviral therapy. E.g for HIV. There are six broad categories of peptidase; Metallopeptidase, Cysteine, Serine, Threonine, Aspartate and Glutamic acid. We extracted a set of 516 experimentally annotated peptidase protein sequences from the UniProt database [38], which mainly contained sequences belonging to the first three sub-families mentioned above. (The other sub-families had only a small number of experimentally annotated sequences. Therefore we did not attempt to classify them. However, we included them as negative examples in training and testing of classifiers developed for the three most represented sub-families). A similar dataset has been used in [19].

The fourth dataset consists of 206 proteins from subfamilies of Human kinases, another clinically important family of proteins. Similar to the previous dataset, we downloaded sequences from the UniProt database which have been experimentally annotated to the kinase family. We mainly focused on the two subfamilies; protein Serine/Threonine kinase activity (GO:0004674) and protein Thyrosine activity (GO:0004713) as others did not have a sufficient number of sequences. Similar datasets have been used in [13, 53, 14]. Table 8.1 shows the composition of all four datasets.

8.2. Experiments

For notational convenience we use the terms $ASSO\_FF$, SVM_FF and SVM_CF to refer to association rule-based, SVM-based feature fusion and SVM-based classifier fusion algorithms respectively.

The main objectives of our experiments were, 1) determine the effect of using reduced alphabets on the different algorithms $ASSO\_FF$, SVM_FF and SVM_CF. 2) determine how they compare with other algorithms that make use of the similarity between amino acids. To achieve the first objective, we compare the results achieved by each algorithm with those achieved by the same algorithm when only the alphabet of 20 amino acids is used, i.e. $A_f = \{A_{20}\}$. To achieve the second objective, we compare our algorithms with a sequence-alignment-based K-nearest neighbor classification algorithm. We implemented a k-NN algorithm that uses BLAST [11] to compute a pairwise sequence alignment of the test sequence with each of the training sequences, pick the k nearest neighbors based on the E-values of the alignments and use majority voting to predict a class for the test sequence. BLAST uses an amino acid similarity matrix to score the alignments and therefore this classification algorithm inherently considers the similarity between different amino acids. Further, we also compare our algorithms with an SVM k-gram based[35, 14] method as well.

Table 8.1. Composition of the datasets.

Gram-negative Bacteria	
Location site	**Number of Sequences**
Outer membrane(OM)	391
Cytoplasmic(Cyto)	278
Cytoplasmic membrane(CM)	309
Periplasmic(PP)	276
Extracellular(EC)	190
Total	1444
Gram-positive Bacteria	
Location site	**Number of Sequences**
Cytoplasmic(Cyto)	194
Cytoplasmic membrane(CM)	103
Cell wall(CW)	61
Extracellular(EC)	183
Total	541
Sub-families of Peptidase	
Function/Sub-family	**Number of Sequences**
Metallopeptidase activity	160
Cysteine-type peptidase activity	129
Serine-type peptidase activity	172
Others	55
Total	516
Subfamilies of Human Kinase Activity	
Function/Sub-family	**Number of Sequences**
Serine/Threonine Kinase activity	160
Thyrosine Kinase activity	41
Others	5
Total	206

Each algorithm was tested on a standard one-against-all binary classification setting. We developed a binary classifier for each class in a given dataset. E.g. we constructed five binary classifiers corresponding to the five sub-cellular location sites of the Gram-negative bacteria dataset shown in Table 8.1. For a given class, sequences belonging to that class are treated as positive instances and the rest of the sequences in the dataset are treated as negative instances. Our intention was to train a classifier so that it can identify sequences in the class for which it was trained for, amidst sequences belonging to all other classes of the given dataset.

As mentioned before, frequent substrings were used as features for *SVM_CF* where as frequent *closed* substrings were used as features for *ASSO_FF* and *SVM_FF*. Further, inclusion of very short length substring patterns (length 4 or less) caused *ASSO_FF* to have prohibitively long run times. Therefore we included only those substrings with length 4 or greater in the feature space used for *ASSO_FF*.

For the SVM K-gram based and k-NN algorithms, we experimented with values of $k = 1, 2, 3, 4$ and $k = 1, 3, 5, 7, 20$ respectively and chose the value which resulted in the best $F1$ value for each class. Linear kernels were used for all experiments involving SVMs.

We carried out independent experiments using alphabets derived by Murphy et al. [39] and Etchebest et al. [21] as listed in Figures 4.2 and 4.3 respectively. I.e. $A_f = \{A_{20}, A_{18}, A_{15}, A_{12}, A_{10}\}$ and $A_f = \{A_{20}, A_{13}, A_{11}, A_9, A_8, A_5\}$. Murphy et al. studied more alphabets than those listed in Figure 4.2, but we did not use those with less than 10 distinct elements in them as it was shown in [39] that they cause information loss with regard to protein folding.

All experiments based on SVMs were performed using a linear kernel function. Further, we did not attempt to optimize the parameter C associated with SVMs as any value greater than 1 produced the same best result. We used the SVM^{light} [26] implementation.

8.3. Results and Discussion

Table 8.2 summarizes the classification results of all algorithms in terms of F1 measure. The F1 measure provides a good single measurement of classifier performance combining both precision and recall. Based on these results, it is evident that

Table 8.2. F1 measure achieved by different classification algorithms as averaged over 5-fold cross validation.

Gram-negative Bacteria					
Class	F1 measure (%)				
	ASSO_FF	SVM_FF	SVM_CF	k-NN	k-grams
OM	73.6(66.0)	**92.8**(87)	92.2(88.1)	87.6	87.5
Cyto	36.7(22.5)	78.5(76.5)	**81.7**(76.8)	59.3	71
CytoMem	81.2(57.3)	91.7(91.3)	**93.6**(91)	80.1	90.1
PP	49(46.5)	72.5(71.6)	76.7(71.2)	**77.3**	68.2
EC	58.5(57.7)	75.1(73.2)	75.6(71.1)	**78.8**	69.7

Gram-positive Bacteria					
Class	F1 measure (%)				
	ASSO_FF	SVM_FF	SVM_CF	k-NN	k-grams
Cyto	73.8(70.4)	89.1(88.8)	**90.1**(87.7)	79.7	84.7
CytoMem	76.2(72.7)	91.8(89.6)	**92.3**(89.7)	77.3	89.3
CW	49(24.3)	82.5(78.1)	83.5(77.7)	72.8	**88.8**
EC	57.5(54.1)	81.3(82.1)	**83.2**(81.8)	78.2	80.2

Subfamilies of Peptidase activity					
Class	F1 measure (%)				
	ASSO_FF	SVM_FF	SVM_CF	k-NN	k-grams
Metallo	75.1(70.6)	85(81.6)	87.8(80.6)	**95.0**	80.1
Cysteine	80.5(77.7)	93.2(91.9)	93.9(90.8)	**98.1**	88.2
Serine	84(82.6)	89.7(87)	92.3(89.2)	**95.3**	86.9

Subfamilies of Kinase Activity - Human					
Class	F1 measure (%)				
	ASSO_FF	SVM_FF	SVM_CF	k-NN	k-grams
Serine	95.6(94.9)	**96.7**(94.8)	**96.7**(94.7)	96.3	94.6
Thyrosine	82.7(79.5)	83.3(80)	84.5(75.8)	**87.8**	71.6

Note1: Alphabets listed in Figure 4.2 (Murphy et al.) are used for ASSO_FF, SVM_FF and SVM_CF. Shown within parenthesis is the F1 value achieved by the same algorithm when only the alphabet of 20 amino acids is used.
Note2: For SVM_CF, a constant support threshold of 2% of the class of interest was used in all experiments.

SVM-based classification performs better than Association Rule-based classification on all of our datasets. However, it must be noted that the models learned by SVMs are *not* easily interpretable and therefore they act as a black box from a users point of view. On the other hand, the classification rules learned by an associative classifier are easy to understand. Especially in our case, the body of a rule, i.e. an itemset, is a collection of frequent substrings so that it can be thought of as a sequence signature which is responsible for assigning a given sequence to a particular class.

Comparison of *SVM_FF* and *SVM_CF* reveals that the classifier fusion approach performs better than its feature fusion counterpart and is also much more efficient in terms of execution time. SVM-based algorithms outperform the k-NN classifier on protein sub-cellular location datasets. However, k-NN performs better on the protein function datasets closely followed by *SVM_CF*.

One of our goals was to evaluate the performance improvements gained by using reduced alphabets. We will discuss the results based on alphabets derived by Murphy et al. [39] as they produced better results in general compared to those of Etchebest et al. [21]. Referring to Table 8.2, we could see that all three algorithms showed some degree of improvement w.r.t their versions which do *not* use reduced alphabets, for all classes across all datasets except for the Gram-positive bacteria extracellular class where *SVM_FF* has performed worse. We carried out a two-tailed binomial test [43] to evaluate the statistical significance of the performance improvements gained by *ASSO_FF*, *SVM_FF* and *SVM_CF* compared to the same algorithms when only the standard alphabet of 20 amino acids is used. Table 8.3 contains these p-values where significant results at the 5% level are shown in boldface. *SVM_CF* demonstrates statistically significant performance improvements for 7 out of 14 classes across all datasets. *ASSO_FF* has achieved significant improvements for 3 classes while *SVM_FF* has gained significant improvements for only 1 class.

Figure 8.1 shows the precision and recall achieved by *ASSO_FF* and *SVM_CF* in comparison with those achieved by the same algorithms when only the standard alphabet of 20 amino acids is used. This figure illustrates that for *SVM_CF*, the use of reduced alphabets improves recall for all classes except Gram-positive cytoplasm and cytoplasm membrane. This is the expected behavior because the use of reduced alphabets allows more patterns to be discovered. It also maintains comparable precision with the exception of Gram-negative bacteria cytoplasm and Gram-positive bacteria cell wall, where more precision has been sacrificed for improved recall. However, the overall F1 measure has improved in all classes as mentioned before. *ASSO_FF* also behaves in a similar manner for many of the classes although in some cases it improves precision as well. However, it has also involved in more cases where the recall has *not* improved.

Table 8.4 shows the F1 measures achieved when the alphabets derived by Etchebest et al. [21] are used. The SVM-based classifiers do not show much variation between the two different alphabet schemes. However, the F1 measure for *ASSO_FF* shows greater variation for some classes. E.g. Gram-positive bacteria cytoplasm, cytoplasm membrane and Gram-positive bacteria cell wall. For the remaining classes the variation is not so remarkable but it is generally greater than that of the SVM-based methods. We believe this is due to the fact that *ASSO_FF* uses longer substrings as features and the different types of alphabets used can have a greater influence on discovering longer patterns than shorter patterns. I.e. longer patterns have a higher tendency to be broken in to smaller fragments due to mismatches. Some of these can be recovered to a certain extent by using reduced alphabets. The extent to which they are recovered depends on the alphabets used. This effect is less for shorter patterns.

Figure 8.2 shows the run times of the different algorithms. k-NN is the fastest algorithm for all datasets, followed by *SVM_CF*. *ASSO_FF* appears to be faster than *SVM_FF*. However, note that given identical training data and feature spaces, SVM-based classification is generally faster than association rule-based classification for our data. Yet, due to limiting the feature space of *ASSO_FF* to only those substrings of length 4 or greater, it appears to performs faster than *SVM_FF*.

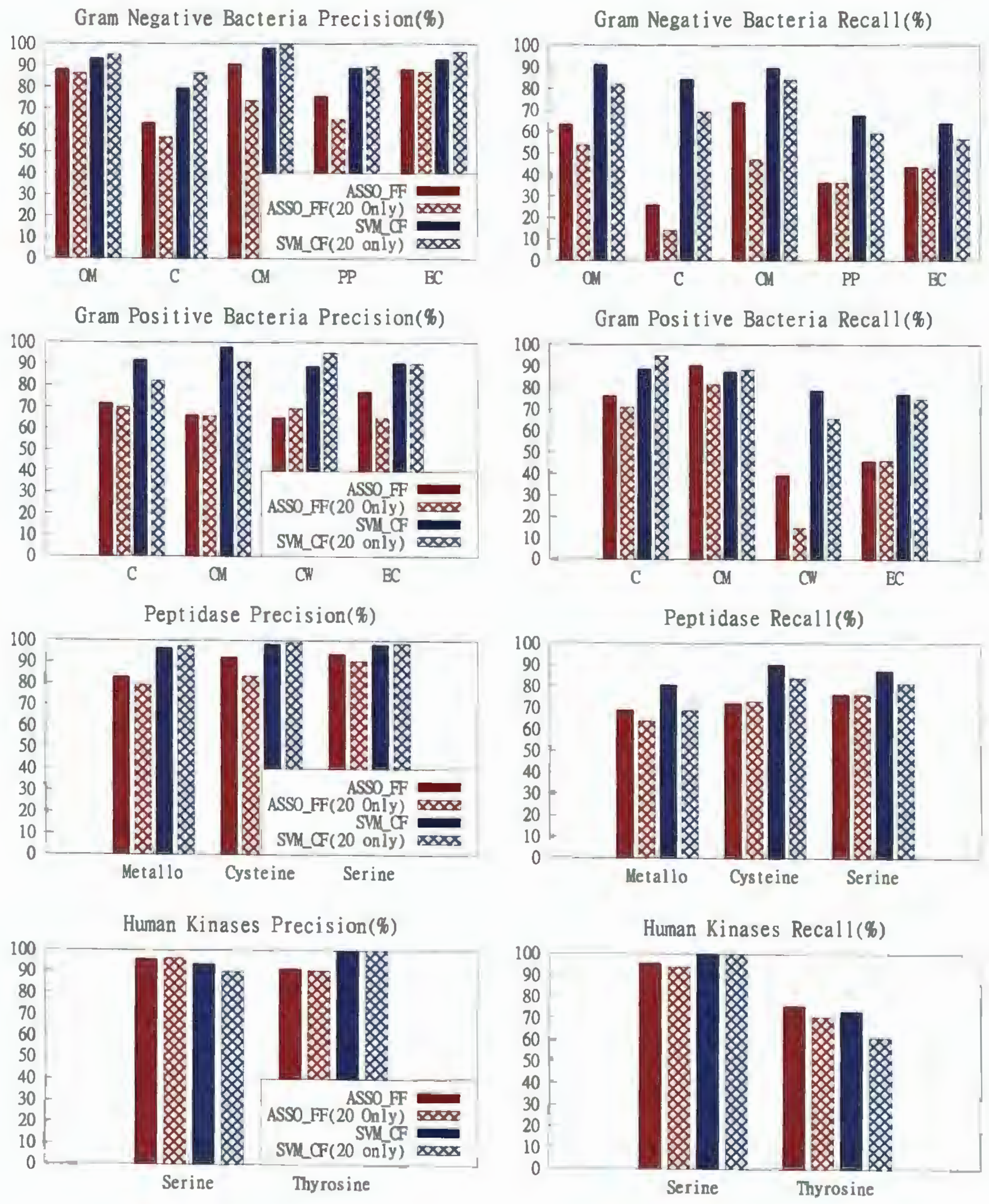


Figure 8.1. Comparison of precision and recall of ASSO_FF and SVM_CF with/without using reduced alphabets.
Alphabets listed in Figure 4.2 (Murphy et al.) are used as reduced alphabets. *20 only* means only the standard alphabet of 20 amino acids is used.

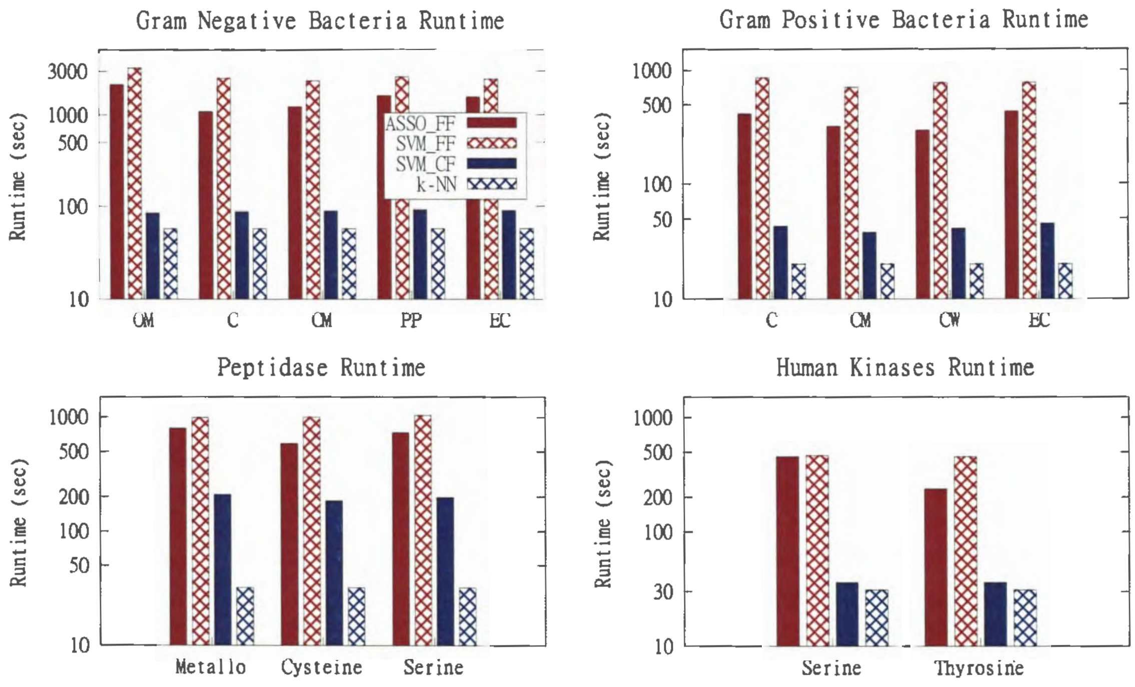


Figure 8.2. Running times of the algorithms as averaged over 5-fold cross-validation.

Table 8.3. P-value for the significance of performance improvements gained by ASSO_FF and SVM_CF w.r.t the same algorithms when only the standard alphabet of 20 amino acids is used.

Gram-negative Bacteria			
Class	**F1 measure (%)**		
	ASSO_FF	**SVM_FF**	**SVM_CF**
OM	**0.0049**	**0.0000**	**0.0005**
Cyto	0.0969	0.4657	0.2543
CytoMem	**0.0000**	1.0000	**0.0106**
PP	0.0777	0.3323	**0.0094**
EC	0.8804	0.7359	0.0931
Gram-positive Bacteria			
Class	**F1 measure (%)**		
	ASSO_FF	**SVM_FF**	**SVM_CF**
Cyto	0.3421	1.0000	0.0730
CytoMem	0.6718	0.4807	0.1094
CW	0.4050	0.5078	0.3877
EC	**0.0091**	NA	0.2891
Subfamilies of Peptidase activity			
Class	**F1 measure (%)**		
	ASSO_FF	SVM_FF	SVM_CF
Metallo	0.1118	0.1839	**0.0002**
Cysteine	0.2801	0.4531	**0.0391**
Serine	0.8026	0.1360	**0.0117**
Subfamilies of Kinase Activity - Human			
Class	**F1 measure (%)**		
	ASSO_FF	**SVM_FF**	**SVM_CF**
Serine	0.7872	0.1796	**0.0156**
Thyrosine	0.6875	0.7539	0.0625

Note1: Alphabets listed in Figure 4.2 (Murphy et al.) were used as reduced alphabets.

Table 8.4. F1 measure achieved by different classification algorithms using the alphabets derived by Etchebest et al. [21].

Gram-negative Bacteria			
Class	**F1 measure (%)**		
	ASSO_FF	**SVM_FF**	**SVM_CF**
OM	73(73.6)	87.6(92.8)	91(92.2)
Cyto	23.2(36.7)	77.1(78.5)	79.7(81.7)
CytoMem	65.5(81.2)	91.7(91.7)	**93.8**(93.6)
PP	47.6(49)	71.6(72.5)	75.5(76.7)
EC	**62.1**(58.5)	74.7(75.1)	**77.8**(75.6)
Gram-positive Bacteria			
Class	**F1 measure (%)**		
	ASSO_FF	**SVM_FF**	**SVM_CF**
Cyto	72.9(73.8)	88.9(89.1)	**90.9**(90.1)
CytoMem	**78.7**(76.2)	89.6(91.8)	91.9(92.3)
CW	**62.7**(49)	80.4(82.5)	83.5(83.5)
EC	55.3(57.5)	81.3(81.3)	82.8(83.2)
Subfamilies of Peptidase activity			
Class	**F1 measure (%)**		
	ASSO_FF	**SVM_FF**	**SVM_CF**
Metallo	69.8(75.1)	82.3(85)	85.5(87.8)
Cysteine	**85.1**(80.9)	91.1(93.2)	93.2(93.9)
Serine	82.2(84)	88.5(89.7)	92.1(92.3)
Subfamilies of Kinase Activity - Human			
Class	**F1 measure (%)**		
	ASSO_FF	**SVM_FF**	**SVM_CF**
Serine	95.3(95.6)	95.4(96.7)	95.5(96.7)
Thyrosine	80.6(82.7)	81.7(83.3)	77.6(84.5)

Note: Shown within parenthesis are the F1 values achieved by using alphabets derived Murphy et al. [39]

CHAPTER 9. PREDICTION OF GENES/MARKERS IN THE CENTROMERE REGION OF CHROMOSOMES IN WHEAT

9.1. Introduction

All the biological information required to construct and maintain a living instance of an organism is contained in what is known as its *genome* [4]. This information is stored in long strands of deoxyribonucleic acid (DNA), which are organized into structure called *chromosomes*. The DNA in a chromosome is divided into discrete segments called *genes*. Figure 9.1 illustrates how DNA, chromosomes and genes are related to each other. Genes are the basic units of heredity of an organism and are responsible for passing genetic traits to off-spring. They also encode the instructions required to produce proteins. Therefore, determination of the DNA sequence of genes and their location on the chromosome is very important in genomic research.

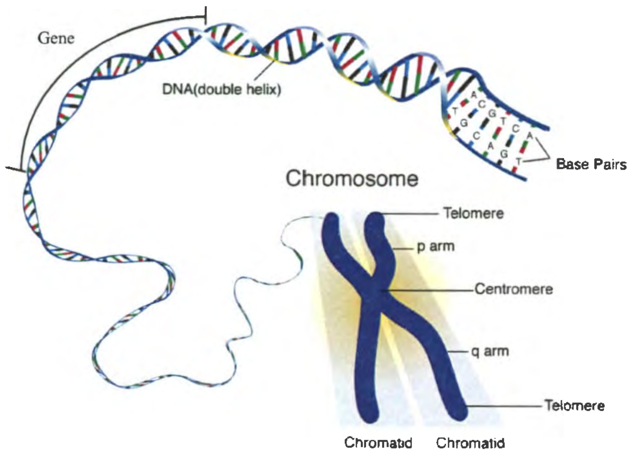


Figure 9.1. DNA, chromosome and genes:How they are related. Original image by [33].

Genome mapping is the process of identifying the locations of genes/markers on their respective chromosomes. Markers are DNA segments that act as landmarks on a map. Figure 9.2 shows an example of how genes/markers are located on a chromosome. Genome maps assist scientists in their study of genes. Examples include, locating a human disease gene on the genome [5, 17] or localizing genes that could be used for improving a crop plant [29]. They also provide a framework for genome sequencing [5, 6, 29]. There are two main types of maps; genetic linkage maps and physical maps. Genetic maps are based on the principle that genes/markers that are close together on a chromosome are less likely to be separated during the process of meiotic recombination and are therefore more likely to be inherited together than those that are father apart (Please refer to [5] for more details). Genetic maps provide the relative order of markers on a chromosome and an indirect estimate of the distance between them. However, the rate of recombination is not uniformly distributed along the length of a chromosome. Therefore, distance in a genetic map may not always correspond to the actual physical distance. On the other hand, physical maps provide an estimate of the true physical distance between markers [5]. There are three general types of physical maps; chromosomal or cytogenetic maps, radiation hybrid (RH) maps, and sequence maps [5]. These different types of maps vary in resolution. Chromosomal maps are low in resolution while RH maps and sequence maps are of higher resolution and provide finer details. In our work, we are especially interested in radiation hybrid mapping of wheat chromosomes.

In Radiation hybrid (RH) mapping, measured doses of radiation is used to break the chromosome of interest into fragments at random points. These fragments are then recovered using recipient cells which are subsequently analyzed for the presence or absence of markers [17]. Similar to genetic mapping, RH mapping also relies on the concept that markers that are physically close on a chromosome tend to stay together on the same fragment because the closer two markers are, the less likely that radiation will strike the chromosome at a point between them and induce a break.

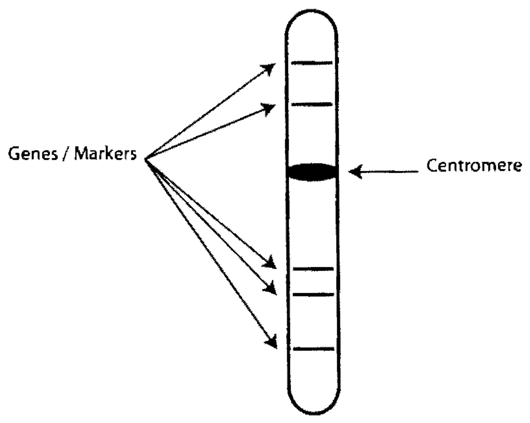


Figure 9.2. Location of genes/markers on a chromosome.

Based on this theory, it is possible to determine the order of markers and estimate the distance between them by observing the frequency of breakage. Since radiation induced chromosomal breakage is random, RH mapping is not affected by the uneven distribution of meiotic recombination as genetic mapping. Please refer to [17, 29] for more details.

9.2. Importance of Genes/Markers Near the Cetromere Region for Radiation-Hybrid Mapping of Wheat

As mentioned above, one of the main problems in genetic mapping is that the rate of recombination is not uniformly distributed along the length of a chromosome. This is a main concern in wheat, where the rate of recombination is approximately proportional to the square of the distance of a given segment from the centromere. Further, it is estimated that one-forth to one-third of the wheat genome is present around the centromeres and this region is subject to less that 1% of total recombination [28]. This causes difficulties in developing high resolution genetic maps for wheat. Since RH mapping does not depend on the natural recombination process, it does not sufer from this problem and is therefore very suitable for developing high-resolution physical maps for wheat chromosomes including the centromere regions.

However, in order to utilize the full potential of RH mapping to map the centromere regions, a sufficient number of marker for these regions should be used in the mapping process. One of the different types of markers used in RH mapping are those that are designed uging Expressed Sequence Tags (ESTs). ESTs are short segments of DNA extracted by sequencing one or both ends of an expressed gene [3]. Our goal is to use the frequent-substring-based sequnce classification techniques mentioned previously in this thesis, to predict ESTs that are likely to be near the centromere of the chromosomes.

9.3. Experimental Evaluation

9.3.1. Datasets and Experiments

We used wheat EST sequences available at [1], for our experiments. It contains ESTs mapped to segments of chromosomes called deletion bins. Since our interest is in predicting ESTs that are probably located near the centromere, we divide the dataset into two classes: ESTs in peri-centromere bins and ESTs in all other bins. Peri-centromere bins are the deletion bins that cover the centromere of a given chromosome. Wheat is a hexaploid species and has three genomes A,B and D. Each of these genomes has 7 chromosomes. We used ESTs mapped to D-genome chromosomes labeled, 1D-7D and focused on one chromosome at a time. This lead to the construction of 7 datasets with two classes each as shown in Table 9.1. Since our algorithms are designed to work with amino acid sequences, we translated the nucleotide sequences of the ESTs into their corresponding amino acid sequences using the Transeq [2] tool.

For each dataset, i.e., chromosome listed in Table 9.1, we employed the classification algorithms *ASSO_FF*, *SVM_FF* and *K-NN* as described in section 8.2 while treating ESTs in peri-centromere bins as the positive class. Experiments were carried out to perform a 5-fold cross validation. However, the automated

Table 9.1. Composition of the wheat EST datasets.

Dataset	**Number of ESTs**	
(Chromosome)	**Peri-centromere**	**Non-peri-centromere**
1D	96	547
2D	150	623
3D	121	591
4D	87	515
5D	218	386
6D	66	411
7D	96	547

parameter estimation process described for *ASSO_FF* and *SVM_FF* in section 7.2.1 did not produce reasonable parameter values. We believe that our classifiers and automatically selected parameter values did not generalize well enough for the wheat EST datasets. Therefore, as an attempt to see how well the classifiers can perform at their best, we selected values for the parameters so as to obtain best classification results based on the final testing data. Note that this contrasts to the way we selected parameter values for the experiments in section 8.2, where we selected them using only the training data *without* the involvement of final testing data.

9.3.2. Results and Discussion

Figures 9.3 and 9.4 show the precision and recall achieved by the three different classification algorithms for the wheat D-genome EST datasets in Table 9.1. According to results shown in Figure 9.3, both *ASSO_FF* and *SVM_FF* demonstrate better precision than what would be achieved due to random chance. The precision achieved by *k-NN* is very low and can be attributed to random chance.

As indicated by Figure 9.4, *k-NN* achieves the best recall. However, this may not be considered as acceptable performance due to its low precision as mentioned before. *SVM_FF* achieves a lower recall compared to *k-NN* but can be nominated as the better of the two classifiers in terms of overall performance due to its higher precision. This

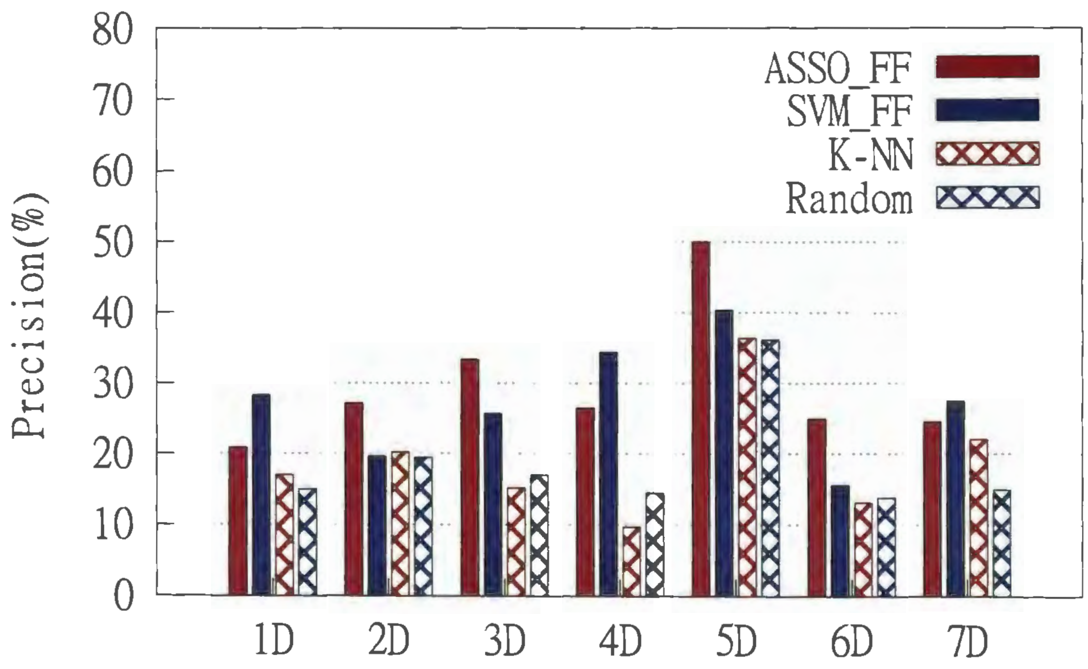


Figure 9.3. Precision achieved by the different classification algorithms for the wheat EST datasets in Table 9.1.
Random is the precision that could be achieved due to random chance.

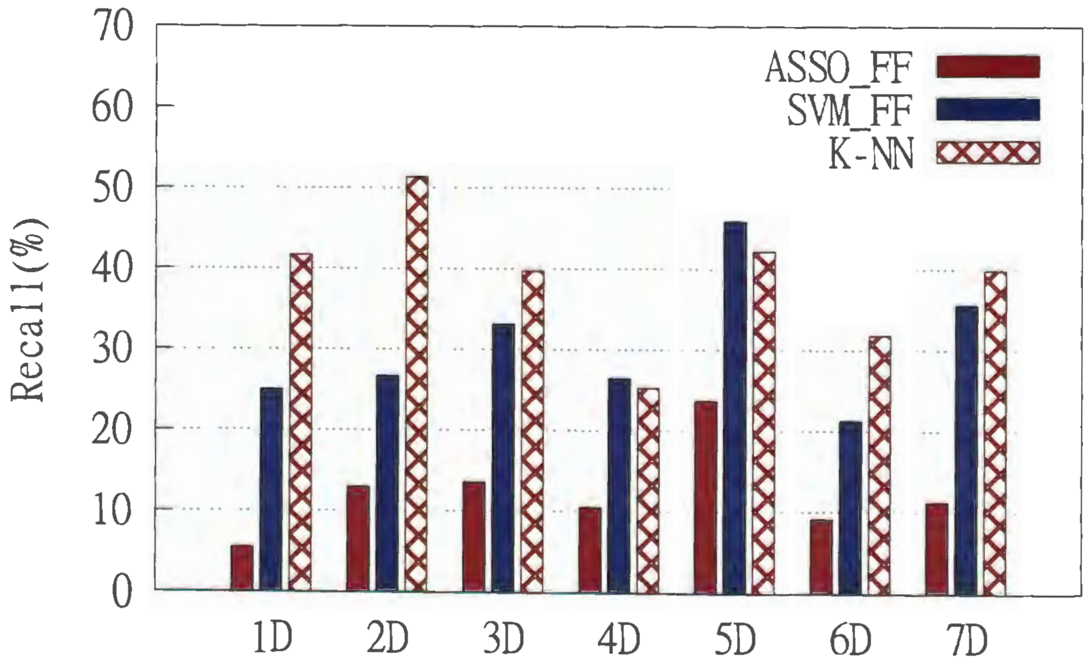


Figure 9.4. Recall achieved by the different classification algorithms for the wheat EST datasets in Table 9.1.

is confirmed by the F1 measures as shown in Table 9.2. The performance of *ASSO_FF* is impaired by its poor recall.

Table 9.2. F1 measure of the different classification algorithms for the wheat EST datasets in Table 9.1.

Dataset	**F1 measure (%)**		
(Chromosome)	**ASSO_FF**	**SVM_FF**	**k-NN**
1D	8.6	**26.5**	24.1
2D	17.5	22.6	**29**
3D	19.3	**28.9**	21.9
4D	15	**29.9**	14.1
5D	32.2	**42.9**	39.1
6D	13.3	18	**18.6**
7D	15.4	**31**	28.4

According to overall results, SVM-based classification seems to show a slight degree of promise on wheat EST data. However, it needs to be considerably improved before it can be used effectively for the prediction of peri-centromere ESTs in wheat. Making use of comparative genomic techniques has been suggested by experts as future direction for improvements.

CHAPTER 10. CONCLUSIONS AND FUTURE WORK

In this thesis, we presented three frequent substring-based sequence classification algorithms using reduce alphabets; An association rule-based classifier adopting a feature fusion approach(*ASSO_FF*), SVM-based classifier adopting a feature fusion approach and an SVM-based classifier adopting a classifier fusion approach(*SVM_CF*). Through our experimental evaluation we showed that *SVM_CF* in particular, demonstrates statistically significant performance improvements with the use of reduced alphabets, for half of the classes studied. Our results show that SVM-based classification performs better that association rule-based classification on all of our datasets. However, interpretabilty of the learned classification rules is an advantage of association rule-based classification using frequent substrings. SVMs also outperform the k-NN classifier on protein sub-cellular location datasets. On the other hand, k-NN classifier performs better on the protein function datasets while the SVMs demonstrate competitive performance. We believe the better performance of the k-NN algorithm on protein function datasets is due to the high pair-wise sequence similarity of these datasets compared to that of the sub-cellular localization datasets. k-NN performs best on the Peptidase function dataset, which has the highest average pair-wise similarity score[1] of 401 whereas for all other datasets the score is lower than 300.

Our algorithms showed limited performance on the wheat EST datasets. *SVM_FF* showed some promise but needs to be considerably improved before it could be employed to effectively identify ESTs in peri-centromere bins.

As future work, we plan to combine the frequent substring-based SVM classifier with the pair-wise sequence alignment-based *k-NN* classifier using the same meta classifier approach described for *SVM_CF*. We believe this will combine the power of

[1]The pair-wise sequence alignment score computed using BLAST for every pair of sequences of the same class and then averaged over the entire dataset.

sequence alignment for homology detection with the usefulness of frequent substrings to discover more specific sequence patterns responsible for the properties of a protein.

It is will also be interesting to extend the same idea of combining frequent substrings as used for association rule-based classification, for the discovery of generalized sequence signatures similar to the ones described in [20]. However, preliminary experiments we carried out in this direction revealed that even with the use of reduced alphabets, frequent substrings tend to break in to smaller fragments due to a few mismatches. This results in a large number of fragmented substrings making it difficult for frequent itemset mining algorithms to operate efficiently. Therefore, we need to explore the use of approximate closed sequential pattern mining algorithms as a solution this problem.

REFERENCES

[1] http://wheat.pw.usda.gov/cgi-bin/westsql/map_locus.cgi.

[2] http://www.ebi.ac.uk/tools/emboss/transeq.

[3] http://www.ncbi.nlm.nih.gov/about/primer/est.html.

[4] http://www.ncbi.nlm.nih.gov/about/primer/genetics_genome.html.

[5] http://www.ncbi.nlm.nih.gov/about/primer/mapping.html.

[6] http://www.ornl.gov/sci/techresources/human_genome/publicat/primer/prim2.html.

[7] Rulequest research, information on see5/c5.0. *http://www.rulequest.com/see5-info.html.*

[8] C. C. Aggarwal. On effective classification of strings with wavelets. In *Proceedings of the 8th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '02, pages 163–172, Edmonton, Alberta, Canada, 2002. ACM.

[9] R. Agrawal, T. Imieliński, and A. Swami. Mining association rules between sets of items in large databases. *SIGMOD Rec.*, 22:207–216, June 1993.

[10] A. Albayrak, H. Otu, and U. Sezerman. Clustering of protein families into functional subtypes using relative complexity measure with reduced amino acid alphabets. *BMC Bioinformatics*, 11:428, 2010.

[11] S. Altschul, W. Gish, W. Miller, E. Myers, and D. Lipmanl. Basic local alignment search tool. *Journal of Molecular Biology*, 215:403–410, 1990.

[12] C. Andorf, D. Dobbs, and V. Honavar. Discovering protein function classification rules from reduced alphabet representations of protein sequences. *In Proceedings*

of the Conference on Computational Biology and Genome Informatics. Durham, North Carolina, USA, 2002.

[13] C. Andorf, D. Dobbs, and V. Honavar. Exploring inconsistencies in genome-wide protein function annotations: a machine learning approach. *BMC Bioinformatics*, 8:284, 2007.

[14] C. Andorf, A. Silvescu, D. Dobbs, and V. Honavar. Learning classifiers for assigning protein sequences to gene ontology functional families. In *Proceedings of the 5th International Conference on Knowledge Based Computer Systems (KBCS)*, page 256, 2004.

[15] N. A. Chuzhanova, A. J. Jones, A. J. Jones, and S. Margetts. Feature selection for genetic sequence classification. *Bioinformatics*, 14:139–143, 1998.

[16] N. Clarke. Sequence minimization: exploring the sequence landscape with simplified sequences. *Current Opinion in Biotechnology*, 6:467–472, 1995.

[17] D. Cox, M. Burmeister, E. Price, S. Kim, and R. Myers. Radiation hybrid mapping: a somatic cell genetic method for constructing high-resolution maps of mammalian chromosomes. *Science*, 250:245–250, October 1990.

[18] M. O. Dayhoff and R. M. Schwartz. Chapter 22: A model of evolutionary change in proteins. In *Atlas of Protein Sequence and Structure*, 1978.

[19] M. Deshpande and G. Karypis. Evaluation of techniques for classifying biological sequences. In *Proceedings of the 6th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining*, PAKDD '02, pages 417–431, London, UK, 2002. Springer-Verlag.

[20] D. H. Dorr and A. M. Denton. A pattern mining approach toward discovering generalized sequence signatures. In *Proceedings of the SIAM International*

Conference on Data Mining, SDM 08', April 24-26, Atlanta, Georgia, USA, pages 353–362, 2008.

[21] C. Etchebest, C. Benros, A. Bornot, A. Camproux, and A. de Brevern. A reduced amino acid alphabet for understanding and designing protein adaptation to mutation. *European Biophysics Journal*, 36:1059–1069, 2007.

[22] I. Friedberg. Automated protein function prediction: the genomic challenge. *Briefings in Bioinformatics*, 7:225–242, 2006.

[23] J. L. Gardy, M. R. Laird, F. Chen, S. Rey, C. J. Walsh, M. Ester, and F. S. L. Brinkman. Psortb v.2.0: Expanded prediction of bacterial protein subcellular localization and insights gained from comparative proteome analysis. *Bioinformatics*, 21:617–623, March 2005.

[24] J. Han and Y. Fu. Discovery of multiple-level association rules from large databases. In *Proceedings of the 21th International Conference on Very Large Data Bases*, VLDB '95, pages 420–431, San Francisco, California, USA, 1995. Morgan Kaufmann Publishers Inc.

[25] S. Henikoff and J. Henikoff. Amino acid substitution matrices from protein blocks. *Proc Natl Acad Sci*, 89:10915–10919, 1992.

[26] T. Joachims. *Making large-scale support vector machine learning practical*, pages 169–184. MIT Press, Cambridge, Massachusetts, USA, 1999.

[27] L. Kajn, A. Kertsz-Farkas, D. Franklin, N. Ivanova, A. Kocsor, and S. Pongor. Application of a simple likelihood ratio approximant to protein sequence classification. *Bioinformatics*, 22(23):2865–2869, 2006.

[28] V. Kalavacharla, K. Hossain, Y. Gu, O. Riera-Lizarazu, M. I. Vales, S. Bhamidimarri, J. L. Gonzalez-Hernandez, S. S. Maan, and S. F. Kianian. High-resolution

radiation hybrid map of wheat chromosome 1d. *Genetics*, 173:1089–1099, June 1990.

[29] V. Kalavacharla, K. Hossain, O. Riera-Lizarazu, Y. Gu, S. S. Maan, and S. F. Kianian. Radiation hybrid mapping in crop plants. *Advances in Agronomy*, 102:201–222, 2009.

[30] E. Keogh, X. Xi, L. Wei, and C. Ratanamahatana. The ucr time series classification and clustering homepage. *www.cs.ucr.edu/ẽamonn/time_series_data/*, 2006.

[31] Y. Kim, J. Sidney, C. Pinilla, A. Sette, and B. Peters. Derivation of an amino acid similarity matrix for peptide: Mhc binding and its application as a bayesian prior. *BMC Bioinformatics 10:394*, 2009.

[32] B. Kuhlman and D. Baker. Exploring folding free energy landscapes using computational protein design. *Current Opinion in Biotechnology*, 14:89–95, 2004.

[33] D. Leja. Talking glossary of genetic terms. *National Human Genome Research Institute, National Institutes Of Health.*

[34] N. Lesh, M. J. Zaki, and M. Ogihara. Mining features for sequence classification. In *Proceedings of the 5th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '99, pages 342–346, San Diego, California, USA, 1999. ACM.

[35] C. Leslie, E. Eskin, and W. Noble. The spectrum kernel: A string kernel for svm protein classification. *In Pacific Symposium on Biocomputing*, pages 566–575, 2002.

[36] C. Leslie and R. Kuang. Fast string kernels using inexact matching for protein sequences. *Journal of Machine Learning Research*, 5:1435–1455, December 2004.

[37] C. Li and J. Wang. Efficiently mining closed subsequences with gap constraints. In *Proceedings of the SIAM International Conference on Data Mining ,08*, pages 313–322, 2008.

[38] M. Magrane and The-UniProt-Consortium. Uniprot knowledgebase: a hub of integrated protein data. *Database, 2011: bar009 (2011)*, 2011.

[39] R. Murphy, A. Wallqvist, and M. Levy. Simplified amino acid alphabets for protein fold recognition and implications for folding. *Protein Engineering*, 13:149–152, 2000.

[40] L. Nanni and A. Lumini. A genetic approach for building different alphabets for peptide and protein classification. *BMC Bioinformatics 9:45*, 2008.

[41] K. Plaxco, D. Riddle, V. Grantcharova, and D. Baker. Simplified proteins: minimalist solutions to the protein folding problem. *Current Opinion in Structural Biology*, 8:80–85, 1998.

[42] H. Saigo, J.-P. Vert, N. Ueda, and T. Akutsu. Protein homology detection using string alignment kernels. *Bioinformatics*, 20:1682–1689, July 2004.

[43] S. Salzberg. On comparing classifiers: Pitfalls to avoid and a recommended approach. *Data Mining and Knowledge Discovery*, 1:317–327, 1997.

[44] A. K. Seewald. Towards a theoretical framework for ensemble classification. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence, IJCAI '03, Acapulco, Mexico*, pages 1443–1444, 2003.

[45] J. Shawe-Taylor and N. Cristianini. *Support Vector Machines and other kernel-based learning methods.* Cambridge University Press, 2000.

[46] R. She, F. Chen, K. Wang, M. Ester, J. Gardy, and F. Brinkman. Frequent-subsequence-based prediction of outer membrane proteins. In *In Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 436–445. ACM Press, 2003.

[47] R. Srikant and R. Agrawal. Mining sequential patterns: Generalizations and performance improvements. In *Proceedings of the 5th International Conference on Extending Database Technology: Advances in Database Technology*, EDBT '96, pages 3–17, London, UK, 1996. Springer-Verlag.

[48] P. Srivastava, D. Desai, S. Nandi, and A. Lynn. Hmm-mode-improved classification using profile hidden markov models by optimizing the discrimination threshold and modifying emission probabilities with negative training sequences. *BMC Bioinformatics 8:104*, 2007.

[49] Vapnik and N. Vladimir. *The nature of statistical learning theory.* Springer-Verlag New York, Inc., New York, New York, USA, 1995.

[50] J. Wang and G. Karypis. Harmony: Efficiently mining the best rules for classification. In *Proceedings of the SIAM International Conference on Data Mining, SDM 05', Newport Beach, California, USA*, pages 205–216, 2005.

[51] L. Wei and E. Keogh. Semi-supervised time series classification. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '06, pages 748–753, Philadelphia, Pennsylvania, USA, 2006. ACM.

[52] Z. Xing, J. Pei, and E. Keogh. A brief survey on sequence classification. *SIGKDD Explorations Newsletter*, 12:40–48, November 2010.

[53] O. Yakhnenko, A. Silvescu, and V. Honavar. Discriminatively trained markov model for sequence classification. In *Proceedings of the 5th IEEE International Conference on Data Mining*, ICDM '05, pages 498–505, Washington, DC, USA, 2005. IEEE Computer Society.

[54] Z. Yang, R. Nielsen, and M. Hasegawa. Models of amino acid substitution and applications to mitochondrial protein evolution. *Molecular Biology and Evolution*, 15:1600–1611, 1998.