COMPARATIVE ANALYSIS OF PROTEIN CLASSIFICATION METHODS

by

Pooja Khati

A MASTERS THESIS

Presented to the Faculty of

The Graduate College at the University of Nebraska - Lincoln

In Partial Fulfillment of Requirements

For the Degree of Master of Science

Major: Computer Science

Under the Supervision of Professors Stephen Scott and Etsuko Moriyama

Lincoln, Nebraska

December, 2004

COMPARATIVE ANALYSIS OF PROTEIN CLASSIFICATION METHODS

Pooja Khati, M.S.

University of Nebraska - Lincoln, 2004

Advisors: Stephen Scott and Etsuko Moriyama

A large number of new gene candidates are being accumulated in genomic databases day by day. It has become an important task for researchers to identify the functions of these new genes and proteins. Faster and more sensitive and accurate methods are required to classify these proteins into families and predict their functions. Many existing protein classification methods build hidden Markov models (HMMs) and other forms of profiles/motifs based on multiple alignments. These methods in general require a large amount of time for building models and also for predicting functions based on them. Furthermore, they can predict protein functions only if sequences are sufficiently conserved. When there is very little sequence similarity, these methods often fail, even if sequences share some structural similarities. One example of highly diverged protein families is G-protein coupled receptors (GPCRs). GPCRs are transmembrane proteins that play important roles in various signal transmission processes, many of which are directly associated with a variety of human diseases. Machine learning methods that have been studied specifically for a problem of GPCR family classification include HMM and support vector machine (SVM) methods. However, amino acid composition has not been studied well as a property for GPCR classification. In this thesis, SVMs with amino acid frequencies were used to classify GPCRs from non-GPCRs. The method was compared with several other methods as HMM-based and decision trees methods. Various sampling schemes were used to prepare training sets to examine if the sampling scheme affects the performance of the classification methods. The results showed that amino acid composition is a simple but very effective property for

identifying GPCRs. SVM with amino acid composition as input vectors appeared to be a promising method for protein classification even when sequence similarities are too low to generate reliable multiple alignments or when only short partial sequences are available.

ACKNOWLEDGEMENTS

I want to thank Dr. Etsuko Moriyama for her continuous guidance and support throughout this thesis. I would also like to thank Dr. Stephen Scott for advising me throughout this thesis and the M.S. program at CSE. Thanks to Dr. Jitender Deogun for serving on my committee.

I'd also like to thank Yavuz F. Yavuz for all the help he has given me in system support. Thanks also to my friends from the Moriyama lab for sharing ideas and creating a fun environment for research at the lab.

I also want to thank my family for their endless love and support. This thesis would not have been possible without the encouragement, support, and patience from my husband, Cory Strope. Cory also provided great help with LaTeX and gnuplot for this thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Day by day a large amount of new protein sequences are being accumulated in various databases. An important task for researchers in bioinformatics is to classify these proteins in families based on their structural and functional properties, thereby predicting the functions of these new protein sequences. Most frequently applied methods (e.g., Pfam, PRINTS, and PROSITE) use multiple alignments to create various forms of models (profile hidden Markov models, fingerprints, patterns, etc.). PROSITE [17], for example, is a database consisting of information on significant sites, patterns, and profiles that specify different protein families. PRINTS is a database of protein fingerprints [3]. Fingerprints are sets of short sequence motifs conserved among members of a protein family. Pfam is a database of alignments and profile-hidden Markov models (HMMs) of protein families [4]. All these three methods require multiple alignments of sequences to build their models.

However, generating reliable multiple alignments becomes problematic when dealing with extremely diverged protein sequences. One such example is the G-protein coupled receptor (GPCR). GPCR is a superfamily of cell membrane proteins that have seven transmembrane regions. Their classification and functional annotation is important in today's

medical and pharmaceutical research because GPCRs play key roles in many human diseases. However, identifying and classifying this membrane protein turns out to be a difficult task, due to the high level of sequence divergence found among the GPCR family members. For example, although all of the three classification methods described above (Pfam, PRINTS, and PROSITE) have successfully built multiple GPCR models each specific to a family or subfamily, none of their models covers the entire GPCR superfamily.

There have been several recent developments in the classification problem specific to the GPCR superfamily. Kim et al. [25] and Moriyama and Kim [31] developed classification methods based on discriminant function analyses using composition and physicochemical properties of amino acids. Karchin et al. [23] developed a system based on support vector machine built on profile HMMs. Liao et al.'s [30] method is similar to Karchin et al.'s [23], but uses pairwise similarity scores between protein sequences with a support vector machine. These newly developed methods learn from both positive (i.e., GPCRs) and negative (i.e., non-GPCRs) examples, giving them a better discrimination power, whereas profile HMMs and other motif-based methods mentioned earlier use only multiple alignments of positive samples to build their models. Lee [27] applied two forms of general multiple-instance learning methods, GMIL-2 and kernel-based GMIL on GPCR classification problem. They used structural properties of amino acids, like that of Kim et al. [25], to build the classifiers.

Despite the development of GPCR specific classification methods, there have been few comparative performance studies. This thesis compares methods that use multiple alignments with those that do not, and methods that use both negative and positive data for learning with those that only use positive data. Furthermore, there has been no study for using simply the amino acid composition for protein classification. Therefore, I examine

the use of amino acid frequencies with various pattern recognition methods and compare their classification performance for the GPCR superfamily.

In this thesis, nine different protein classification methods (classifiers) were included for the performance analysis. The nine methods used are the profile-HMM, support vector machines (SVMs) with four different kernel functions (linear, polynomial, sigmoid, and radial basis functions), SVM-pairwise developed by Liao et al. [29], SVM-Fisher developed by Karchin et al. [23], decision trees, and boosted decision trees. As mentioned above, HMMs make use of a multiple alignment of positive sample sequences to build a model. SVMs, on the other hand, use sequences belonging to a family (positive examples) as well as sequences that do not belong to the family (negative examples). It learns to discriminate between positive and negative samples based on sequence similarity scores or other attributes extracted from the sequences. The decision trees methods learn also from negative as well as positive sets of data, but try to build a tree with smallest number of nodes to best classify the two sets of data.

Three different sampling methods for positive example data were used to discover if such differences affected classification performance. The three basic methods of sampling were random sampling, taxonomical sampling, and phylogenetic sampling. Some sequences of bacteriorhodopsin proteins were added as a part of negative examples. Bacteriorhodopsin proteins are similar to GPCRs in that they have seven transmembrane regions, but are not actually GPCRs. The addition of such proteins in the example sets was expected to make the learning more specific. The methods for preparing training data (positive and negative example data used to train classifiers) are described in detail in Chapter 3.

The nine classifiers were tested on test data sets independently created from the training

sets. In one test, some classes of GPCRs that were not included in the training set were intentionally included. With this particular test, I was able to examine how well the nine classifiers can identify GPCRs even if their examples were not included in the dataset they were trained on.

The classifiers were also tested against a data set consisting of subsequences as short as 50 amino acids. This is based on Kim et al.'s [25] results showing that their discriminant function analysis method outperformed other methods (e.g., Pfam) for such short sequences. The expectation was that the classifiers using amino acid composition as input vectors might be able to identify short subsequences as well as the discriminant function analysis methods used by Kim et al. [25] and Moriyama and Kim [31].

Various statistics were used to analyze the classification performance: accuracy of each method, cross-validation test, minimum error point calculation, maximum and median rates of false positives, and receiver operating characteristics graphs. By using these various statistics, the performance of each classifier was examined in detail, not only on their accuracy rates, but also their sensitivities, specificities, and the relationships among these statistics. Chapter 3 describes the statistics used in this thesis.

The remainder of the thesis is organized as follows. Backgrounds on some representative methods used in protein classification as well as the protein families used in this study are described in Chapter 2. Chapter 3 explains data collection methods, different classification methods, and performance analysis used in this study. Results and discussions are given in Chapter 4. Finally, Chapter 5 concludes this thesis with the overall discussion and future works.

Chapter 2

Background

2.1 Protein families used in this study

2.1.1 G-protein coupled receptors (GPCRs)

G-protein coupled receptors (GPCRs) are a superfamily of cell membrane proteins. They are characterized by seven water-insoluble (hydrophobic) regions believed to represent those that pass through the cell membrane, or transmembrane regions, as shown in Figure 2.1. Each GPCR has an amino terminal (NH_2 or N-terminal) region outside of the cell (extracellular), followed by three sets of alternate intracellular (inside of the cell) and extracellular loops, which connect the seven transmembrane regions, and a final intracellular carboxyl terminal (COOH- or C-terminal) region [43].

GPCRs are involved in signal transmission from the outside to the interior of the cell through interaction with heterotrimeric[1] G-proteins, or proteins that bind to guanine (G) nucleotides. The receptor is activated when a ligand that carries an environmental signal

[1] A heterotrimeric protein is composed of three subunit proteins (-trimeric), where each subunit protein is different from the others (hetero-).

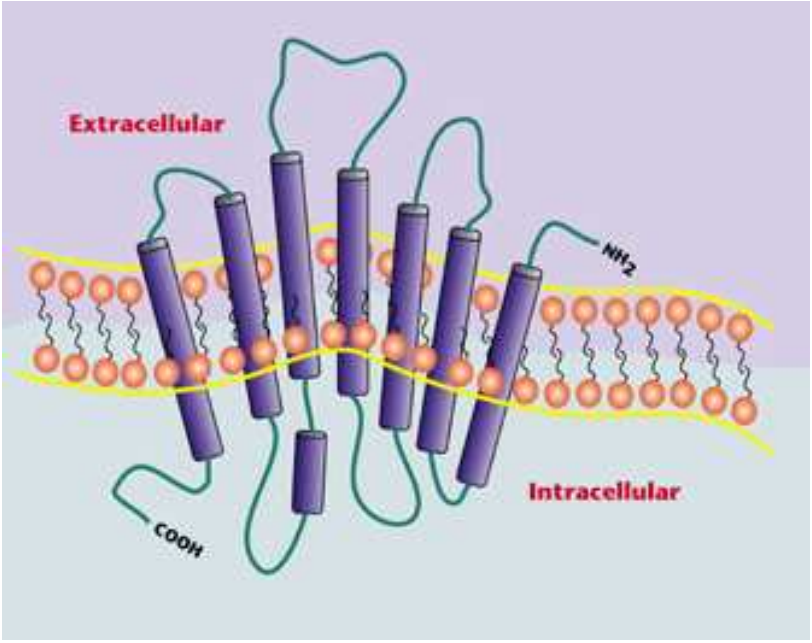

Figure 2.1: A model of G-protein coupled receptor showing seven transmembrane regions (courtesy [2]).

binds to a part of its cell surface component as shown in Figure 2.2. A wide range of molecules is used as the ligands including peptide hormones, neurotransmitters, pancrine mediators, etc., and they can be in many forms: e.g., ions, amino acids, lipid messengers, proteases.

The heterotrimeric G-proteins have three subunits, namely, α, β, and γ (Figure 2.2). The G-protein activity is regulated by the α subunit, which binds guanine (G) nucleotides. In an inactive state, α is bound to a GDP (guanine diphosphate), which together are bound to subunits β and γ (Figure 2.2A). A ligand binding at the extracellular domain of the receptor induces a conformational change in the receptor, which causes the G-proteins to bind to the intracellular domain of the receptor (Figure 2.2B). This stimulates the exchange of the GDP with a GTP (guanine triphosphate) in the binding site of the α subunit. The activated GTP-bound α subunit then dissociates from the β and γ subunits (Figure 2.2C).

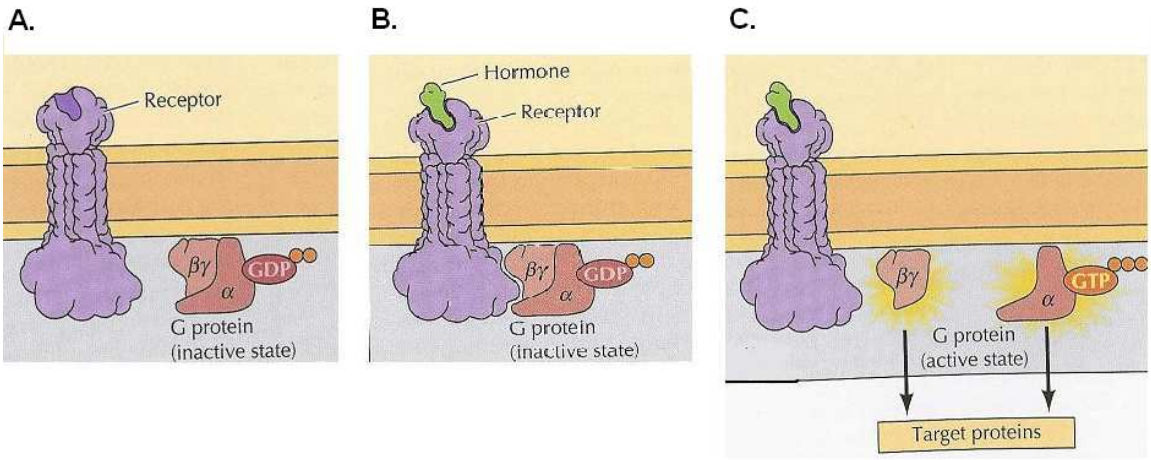

Figure 2.2: Signal transduction process by a GPCR and a G-protein with α, β, and γ subunits (courtesy [8]). This example shows a hormone peptide as a ligand.

The β and γ subunits remain bound to each other and function as the $\beta\gamma$ complex. The $\beta\gamma$ complex and the GTP-bound α subunit interact with their targets, for example, an enzyme or an ion channel, to transmit the signal. The bound GTP becomes a GDP due to hydrolysis after the transmission of the signal. The GDP-bound α subunit reassociates with the $\beta\gamma$ complex to form a heterotrimeric G-protein (Figure 2.2A), which is ready for another cycle of transmission of a signal through a GPCR [8].

GPCRs are involved, for example, as light sensing molecules in the eye (rhodopsins), odorant receptors in the olfactory system, and as taste receptors [12]. They are found in a wide range of eukaryotic organisms. The GPCRDB, a database system for GPCRs [15], divides the GPCR superfamily into five major classes based on the ligand types, functions, and sequence similarities, as shown in Table 2.1. The sequences of different GPCR classes are highly diverged from each other, except that they share one common structural feature, that is, they all have seven hydrophobic transmembrane regions. GPCRs within a class share common functions and more sequence similarities. Class A, the Rhodopsin-like class, is by far the most populated GPCR class with more than 3,500 members in the

Table 2.1: Major GPCR classification based on GPCRDB (as of November 2003).

Class	Examples	Number of entries
A: Rhodopsin like	Rhodopsin and adrenergic receptors	3,519
B: Secretin like	Calcitonin receptors	217
C: Metabotropic glutamate/pheromone	Metabotropic receptors	131
D: Fungal pheromone	pheremone receptors	24
E: cAMP receptors (Dictyostelium)	cAMP receptors	5

database. Each class is further divided into subclasses, subgroups, and so forth, depending upon the common agents they bind to and sequence similarities.

Identifying the function of GPCR sequences is important in biomedical and pharmaceutical research, because GPCRs play key roles in many biologically important functions and are related to many diseases (e.g., neurological cardiovascular diseases, depression, obesity, pain, and viral infections [1]). However, identifying and classifying this membrane protein family is a difficult task due to the high levels of divergence observed among the GPCR family members. Therefore, it becomes important that there be a way to accurately and efficiently identify any new GPCRs from genomic data. This would benefit the pharmaceutical research and give us a better understanding of GPCR functions. The methods developed in this thesis will also be applicable to other proteins. GPCRs are used in this study due to their scientific importance, and also as an example of highly diverged protein families.

2.1.2 Bacteriorhodopsin

Bacteriorhodopsin is a type of transmembrane protein found in bacteria. It is named as such because of its similarities to the rhodopsin (a GPCR) found in the outer segments of mammalian retina. Bacteriorhodopsin clusters in purple patches in the bacterium, *Halobacterium halobium*, and have seven transmembrane regions spanning the cell membrane. The

bacteria carry out a light-driven proton transport by means of bacteriorhodopsin. When there is not enough oxygen for the bacteria for oxydative metabolism, the bacteria use the energy from the sunlight to pump protons out of the cell. The proton gradient generated by such mechanism represents potential energy, which is later used by the cell to synthesize ATP (adenosine triphosphate) that powers the cell [13]. Although bacteriorhodopsins share the same seven transmembrane structure, they do not activate any G-proteins, hence they are not GPCRs.

The protein classification methods developed in this thesis learn from training sets composed of positive examples (GPCRs) and negative examples (non-GPCRs). Based on the information gained (models), they classify protein sequences in the test set. Bacteriorhodopsin sequences were included in some training as a part of negative examples to examine if they would improve the specificity of the algorithms to classify GPCRs. In other words, the algorithms were tested for their abilities to discriminate GPCRs from bacteriorhodopsins that also have seven transmembrane regions.

2.2 Protein Classification Methods

In this section, I explain the widely used hidden Markov model (HMM) method and some recently developed protein classification methods using support vector machines (SVMs) and discriminant function analysis.

2.2.1 Profile hidden Markov model (HMM)

One frequently used method for protein classification is a hidden Markov model. Hidden Markov models, which are extensions of Markov chains, have a finite set of states ($a_1, \ldots a_n$), including a begin state (where the sequence begins) and an end state (where

the sequence terminates). Each state has two probabilities associated with it:

- the *transition* probability T_{ij}, or the probability that a state a_i will transition to another state a_j, where $j = i + 1, \ldots, n$, and
- the *emission* probability $E(\mathrm{x}|j)$, or the probability that a state a_j will emit a particular symbol x. Emission probabilitites are properties of only HMMs and not Markov chains.

The difference between a Markov chain and a hidden Markov model is in the information known on each state. In a Markov chain, for any sequence, all state transitions are exactly known– i.e., there is a unique, known path through the model. In a hidden Markov model, the state information is *hidden* from the user [9].

For example, Figure 2.3A shows Markov chains for two models (loaded and fair dice). A Markov chain can determine the probability that a given sequence of rolls was generated from a loaded die or a fair die. It, however, cannot determine which ”segments” of the sequence of rolls were generated by which die. On the other hand, Figure 2.3B shows a hidden Markov model with two states (dice), the transition probabilities between them, and the emission probabilities of each of the 4 symbols at each state. An HMM can tell, for a given sequence of rolls, which segment was generated by a loaded die and which by a fair die. For example, for the sequence `12434132443444134`, the following state sequence can be predicted by an HMM.

```
Rolls:  12434132443444134
State:  FFFFFFFFLLLLLLLLL
```

where `F` is a fair die and `L` is a loaded die.

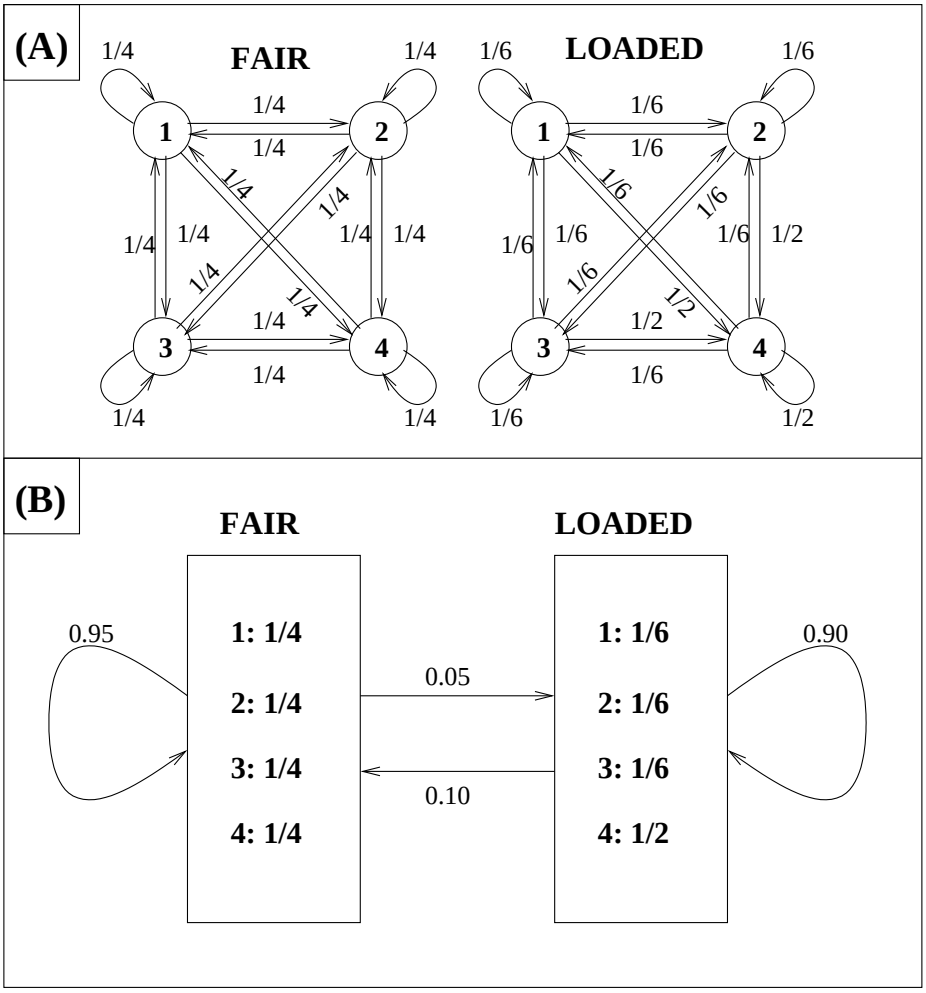

Figure 2.3: An example of a Markov chain and a hidden Markov model. If we consider two dice (loaded die and the fair die) as two models (A), then each of them has its own Markov chain with transition probabilities between different numbers in a die. If we consider the two dice as two states of a hidden Markov model (B), it has transition probabilities between the both states (dice), and emission probabilities within each state.

In biological sequence analysis, hidden Markov models are built based on a multiple alignment as shown in Figure 2.4. In general, the multiple alignments are generated from a training set consisting of positive examples of protein sequences that belong to a certain functional family sharing a level of sequence similarities.

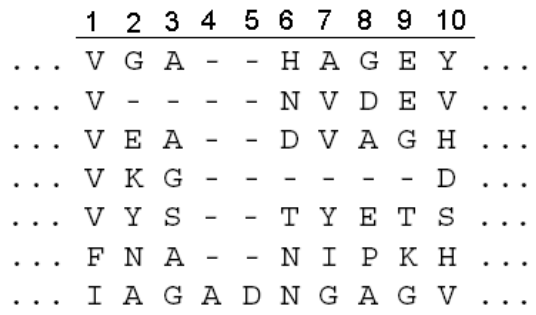

Figure 2.4: An example multiple alignment to create a hidden Markov model. A gap is represented by a '–'. Columns 1-3 and 6-10 are "match" columns, while the columns 4 and 5 are "insert" columns.

Given a multiple alignment of protein sequences, "match", "insert", and "delete" states are first identified. If a column of the multiple alignment has less than or equal to fifty percent gaps (i.e., a half or more of the sequences emit an amino acid), then it is classified as a "match column" (columns 1-3 and 6-10 in Figure 2.4). A non-gap entry in a match column is a "match state" in the HMM, while a gap in a match column is a "delete state". Delete states are presumed to be modifications that stem from an amino acid sequence losing one or more amino acids in an evolutionary event. The last type of state is the "insert" state. "Insert columns" (columns 4 and 5 in Figure 2.4) are similar to delete states, except that the evolutionary modification to the amino acid sequence is that of gaining amino acids. A non-gap in an insert column is an "insert state", while a gap in an insert column is ignored since it does not represent an event of evolutionary significance.

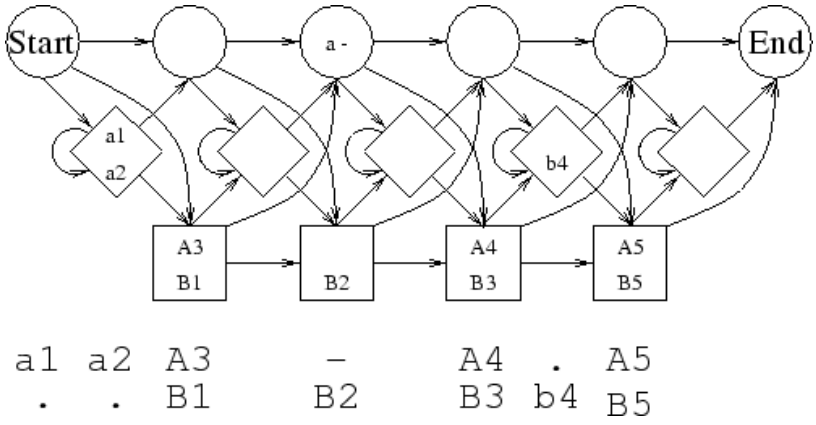

Figure 2.5: A hidden Markov model (courtesy [16]) with delete (circle), insert (diamond), and match (square) states. Transitions are allowed along each arrow. Delete and match states can only be visited once for each position along a path. Delete states do not emit any symbols. Insert states are allowed to insert multiple symbols. The alignment at the bottom is used to build the model in this example. The sequences begin in the start state. Amino acids a1 and a2 are inserted at the beginning of the sequence. A3 and B1 are the first matched symbols, followed by a deletion, where B2 is matched with a gap. A4 is then matched with B3, b4 is inserted, A5 is matched with B5, and finally the end state is reached.

As shown in Figure 2.5, a hidden Markov model, which can be visualized as a finite state machine, has a start and an end state in addition to the previously identified match, insert, and delete states. Each of these states has position-specific transition probabilities for transitioning into each of these states from the previous state (represented by arrows in Figure 2.5). Match states have position-specific emission probabilities for each of the 20 amino acids. Insert states also have position-specific emission probabilities for inserting each of the 20 amino acids at that state. When no residue is associated with a node, it is a delete state, and no emission probability is associated with it.

To obtain the probability that a new sequence belongs to the family of the model, the new sequence is compared to the HMM by aligning it to the model. The most probable path taken to generate the sequence similar to the new sequence gives the similarity score.

It is calculated by multiplying the emission and transition probabilities along the path. The most likely path through the model is computed with the *Viterbi* algorithm or the *forward* algorithm [9]. One could also generate the most probable sequence obtained from a particular HMM by summing over all possible paths and choosing the path with the maximum score. In both ways, the most probable path can be efficiently and optimally calculated.

2.2.2 Support vector machines (SVMs)

A support vector machine (SVM) is a learning machine that makes a binary classification based on a separating hyperplane on a remapped instance space [7]. The goal of the classification is to remap the input vectors onto a multi-dimensional space so that the instances are linearly separable.

SVMs learn from labeled examples from a training set including both positive and negative samples. Depending upon a set of attributes, SVMs find a hyperplane that classifies the positive and negative data in the training set. The hyperplane is optimized in such a way that the distance called the *margin*, between the hyperplane and the closest training example is maximized. The data points nearest to the margin on both sides are called *support vectors*. We assume that there is a mapping or target function between the data and their labels the machine will learn [22]. A kernel function, which is a dot product that is used in remapping input feature vectors, is used to find the hyperplane. Once the hyperplane is found, unlabeled examples from the test set can be classified as shown in Figure 2.6. Classification can be done solely based upon the support vectors found.

Let us represent each sequence by a feature vector (a collection of the attributes in a vector format). If the dimension of the feature vector is l (l attributes), a sequence $\mathbf{x}$ can

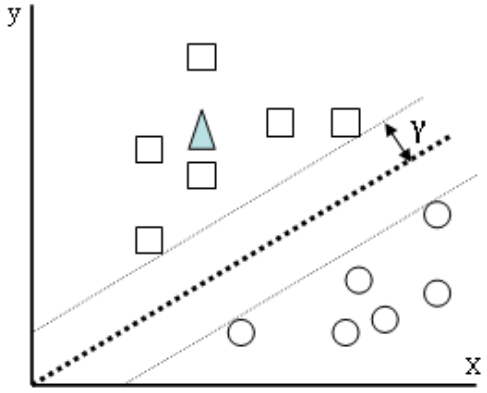

Figure 2.6: A hyperplane classifying two classes of data. A new sample of an unknown class can be classified based on the hyperplane. In this figure, the training data have two dimensions, represented by the x and y axes. Two classes of data are represented by squares and circles. The hyperplane that is calculated from these training examples is given by the bold dotted line, separated from the closest training vectors by the distance γ. The classification of an unknown sample (triangle) is done by determining which side of the hyperplane the new instance falls. In this example, the prediction for the unknown sample would be square.

be represented by $\mathbf{x} = [x_1, x_2, \ldots, x_l]$. In a two-class problem, the label of the sequence can be either 1 or -1. Let us represent the label of the sequence $\mathbf{x}$ with $y_x = \{1, -1\}$. A classifier is then built using the feature vectors of the training set. A weight vector of the same dimensions as the feature vector is represented by $\mathbf{w} = [w_1, w_2, \ldots w_l]$. The label of the sequence is then predicted as 1 if $\mathbf{w} \cdot \mathbf{x} > b$ (b is a threshold), else the label is -1. The equation of the margin γ_x is given by

$$\gamma_x = y_x(\mathbf{w} \cdot \mathbf{x} + b) \tag{2.1}$$

If γ is positive, then the sequence is correctly classified, otherwise, it is not correctly classified. Every time a sequence is incorrectly classified, the weight vector $\mathbf{w}$ and the threshold b are updated. A simple algorithm of an SVM is presented in Algorithm 1. In this al-

Algorithm 1: SVM algorithm

```
begin
    w_0 ← 0, b_0 ← 0, k ← 0;
    R ← Radius of the vector most distant from the origin of vector space;
    while mistakes are made on the training set do
        for i = 1 to N do
            if y_i(w_k · x_i + b_k) ≤ 0 then
                w_{k+1} ⟵ w_k + η y_i x_i;
                b_{k+1} ⟵ b_k + η y_i R^2;
                k ⟵ k + 1;
            end
        end
    end
end
```

gorithm, $\mathbf{w}$, the weight vector, is initialized to 0 at the beginning. The threshold b is also initialized to 0. k is the number of mistakes made, R is the radius of the hypersphere, and is initialized to the maximum distance of a training vector from the origin of the hypersphere (i.e., the hypersphere containing the data). N is the total number of training vectors. η is a learning rate. In this algorithm, the final predictor or the decision hyperplane is given by the equation

$$h(\mathbf{x}) = \text{sgn}(\mathbf{w}_k \cdot \mathbf{x} + b_k) \tag{2.2}$$

When the loop in Algorithm 1 exits, the final weight vector $\mathbf{w}$ is in the form

$$\mathbf{w} = \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i, \tag{2.3}$$

where α_i is the number of mistakes made on example. Now the equation for the decision hyperplane becomes

$$h(\mathbf{x}) = \text{sgn}(\sum_{i=1}^{N} \alpha_i y_i \langle \mathbf{x}_i \cdot \mathbf{x} \rangle + b). \tag{2.4}$$

The data is represented in dot products. The dot product allows us to use kernels which implicitly remap and compute dot products. An SVM algorithm using the kernel function,

Algorithm 2: SVM algorithm with a kernel function $K()$

```
begin
    while mistakes are made in for loop do
        for i = 1 to N do
            if y_i(Σ_{j=1}^{N} α_j y_j K(x_j · x) + b) ≤ 0 then
                α_i ⟵ α_i + 1;
                b ⟵ b + y_i R^2;
            end
        end
    end
end
```

$K(X, Y)$, for some vectors X and Y, is shown in Algorithm 2 [2]. Some commonly used kernel function includes: linear, polynomial, radial basis, and sigmoid functions. The equations for the respective kernel functions are listed in Table 2.2.

Table 2.2: The four types of kernel functions frequently used with SVM.

Linear Kernel	$K(x, y) = (x \cdot y + 1)$
Polynomial Kernel	$K(x, y) = (kx \cdot y + c)^p$
Sigmoid Kernel	$K(x, y) = \tanh(kx \cdot y + c)$
Radial Basis Kernel	$K(x, y) = e^{-\gamma\|\|x-y\|\|^2}$

The advantage of using the SVM in this study is the ability to classify protein sequences without depending on multiple alignments. There have been only a few studies using SVMs in the classification of protein sequences. Karchin [22] (and also Karchin et al. [23]) developed the SVM-Fisher method. Liao and Noble [29] on the other hand used the SVM-pairwise method. Wang et al. [42] and Zhang [45] also experimented with SVM on identifying Thioredoxin proteins, another example of protein family with low primary sequence similarity. Both SVM-Fisher and SVM-pairwise methods were used in this study, and they are described next.

[2] The algorithm presented here is a simple form of SVM that is not used by typical SVM packages. Regular SVMs (including SVM-light) formulate the learning problem as a convex quadratic optimization problem and then apply interior point methods to solve it.

SVM-Fisher

Jaakkola et al. [19] developed a method to derive kernel functions from generative probability methods. Using this method, protein sequences with variable lengths can be handled by extracting fixed length vectors. In addition, prior knowledge from the probability model on a sequence can be used by a kernel function. It requires an HMM built from a set of sequences of interest, most likely from a certain protein family. The forward-backward algorithm is used to obtain the likelihood score for a query sequence. The forward-backward algorithm also extracts "sufficient statistics" for each state. The sufficient statistics are the posterior frequencies of having taken a particular transition or having generated one of the residues of the query sequence X from a particular state [18]. Analogous to the sufficient statistics are the "Fisher scores" given by

$$U_X = \nabla_\theta \log P(X|H_1, \theta), \tag{2.5}$$

where each component of U_X is a derivative of the log likelihood score for the query sequence X with respect to each parameter given the model H_1. The magnitude of the components specifies the extent to which each parameter contributes to generating the query sequence.

Karchin's [22] experiments consisted of using the mixture priors to compute enhanced Fisher score vectors. These mixtures are the pre-calculated amino acid distributions estimated by studying large databases of protein sequences. The probability of amino acids in each state is decomposed into a number of components. In this case, there are nine components. The nine components are the probability distribution of an amino acid at a match state belonging to the nine subclasses as described in [22]. The final Fisher score vector for a sequence X, given an HMM model H_1, therefore has nine components for every

match state in H_1. Transition probabilities are not used as the parameter set, (θ in Equation 2.5) during the computation of Fisher scores, reducing the computational complexity of the calculation. Karchin also noted that using transition probabilities does not improve the performance of SVM classification [22].

After deriving the Fisher score vectors for training sample sequences, those vectors are used to train the SVM with a Gaussian radial basis kernel. This method is known as the *SVM-Fisher* method. Details of the derivation of the Fisher score can be found in Jaakkola et al. [18].

SVM-Fisher method was used to discriminate GPCRs in Karchin [22] and Karchin et al. [23]. Karchin's results showed that the HMM method, using the SAM software, was the best method to discriminate the GPCR superfamily. The HMMs were able to discriminate GPCRs including Classes A, B, and E from non-GPCRs perfectly. SAM outperformed other methods including BLAST, Smith-Waterman, and SVM-Fisher. On the other hand, GPCR subfamilies within the superfamily were best discriminated by the SVM-Fisher method.

SVM-pairwise

Liao and Noble [30] developed another vectorization method for protein sequences. The vectors are then used in a support vector learning. A protein sequence is compared to every protein in the dataset. Comparison is conducted by performing the Smith-Waterman local alignment algorithm on two sequences, and an E-value for the similarity is estimated [30]. Since the pairwise scores between every two sequences are used as an input vector for the SVM, this method is called *SVM-pairwise*. After comparing a sequence X with all the sequences in the data set, the feature vector corresponding to a protein X is in the form of

$F_X = [f_{x1}, f_{x2}, \ldots, f_{xn}]$ where n is the total number of proteins in the training set and f_{xi} is the E-value of the Smith-Waterman score[3] between sequences X and the ith training sequence. As in SVM-Fisher, a radial basis kernel function is used in the SVM.

Liao and Noble [30] tested the SVM-pairwise method to measure how well it could classify proteins into superfamilies based on the Structural Classification of Proteins (SCOP) database [32]. Sequences were selected from the SCOP database after removing very similar sequences. SVM-pairwise was compared with other classification methods: PSI-BLAST, SAM, SVM-Fisher, and Family Pairwise Search (FPS). Similar to SVM-pairwise, FPS scores a protein sequence against a family of sequences [14]. A query sequence is compared to a set of sequences and the pairwise scores are combined to obtain an overall score for the similarity of the query sequence to the family of sequences compared to. Their results showed that SVM-pairwise performed better than all of the other four methods.

2.2.3 Discriminant function analysis

Another interesting GPCR classification method was developed by Kim et al. [25]. The method was called Quasi-periodic Feature Classifier (QFC). A feature space was generated using statistical measures of physico-chemical properties of amino acids. Linear discriminant function analysis with non-parametric optimization was used to discriminate GPCRs from other proteins.

The general form of linear discriminant function is given by

$$DS = a_1X_1 + a_2X_2 + \ldots + a_nX_n \tag{2.6}$$

[3]The Smith-Waterman score is the maximum score for a local alignment between the two protein sequences being compared. The E-value or the expectation value of a score (and an alignment) is the number of different alignments with scores equivalent to or better than that particular score expected to occur in the database search by chance. The lower the E-value, the higher the confidence.

where DS is the discriminant score, X_i is the value for each descriptor representing the protein sequence, and a_i is the coefficient for that particular variable. The discriminat score is calculated for each sample and is used to determine the class to which the sample belongs. The algorithm tries to solve for a set of coefficients in such a way that it would give the highest accuracy on classification of the training set. In a parametric linear discriminant analysis, the algorithm tries to maximize the between-class to within-class variance [6].

In QFC, however, the authors used a non-parametric optimization method using 'runs' statistics in order to avoid assuming normal distributions of variables. After training, a linear hyperplane is determined that discriminates the two classes optimally. Test data is then classified to respective classes using this linear function.

The main idea of the method was to construct the most useful feature space. Fourteen physico-chemical properties were considered, and after step-wise deletion, four variables were selected. After training the algorithm with GPCRs, Kim et al. [25] tested the method comparing with three other methods: PROSITE, Pfam, and PRINTS. QFC performed better than or as good as other methods in the test set. More interestingly this method outperformed the other three methods when tested on randomly fragmented short sequences as short as 50 amino acids. They also observed that QFC performed well identifying GPCRs from other transmembrane proteins.

Moriyama and Kim [31] later used parametric discriminant analysis with linear, quadratic, and logistic functions. They also included one non-parametric method, K-nearest neighbors. They compared the performance of GPCR discrimination among these methods and QFC described above. They found that the performance of these discriminant function methods is comparable to QFC and similarly better against short sequences compared to

Pfam, PRINTS, and PROSITE.

Lee [27] applied a multi-instance version of Kim et al.'s method in predicting functional classes of GPCRs from Class A. The generalized multiple-instance learning methods (GMIL-2 and kernel-based GMIL) was used with 7 physico-chemical properties derived from Kim et al. to model the protein functional classes of Class A GPCRs. Then GMIL algorithms were used to learn and predict functional classes. They observed that kernel-based GMIL outperformed GMIL-2 by having better accuracy rates for 6 out of 11 train groups and same accuracy rates for 2 train groups.

Discriminant analysis methods were not included in this study. However, their performance is compared with the nine methods based on the results by Moriyama and Kim [31].

Chapter 3

Materials and Methods

In this chapter, data collection methods used in this study are described first. Next, protein classification methods used, and finally how the performance of various methods were analyzed is described.

3.1 Data Collection

Before implementing any learning algorithms, training and testing data sets need to be prepared. The learning algorithms used in this project: support vector machines, hidden Markov models, and decision trees, all were trained on several training sets, and then their classification performance was examined against test sets. I will now begin by explaining how the training and testing sets were prepared.

3.1.1 Data sources

The positive datasets, which consist of known GPCR sequences, were taken from GPCRDB [15]. This database maintains a repository of known sequences from the GPCR superfamily. The

May 2003 release[1] of this database was used for this study. It consists of about 5,300 GPCR sequences. The GPCR superfamily is divided into five major classes according to function and sequence similarities as listed in Table 2.1. Class A, Rhodopsin-like, is the largest class and consists of 3,519 sequences. Class B consists of only 217 sequences, and other classes include even fewer sequences. The number of sequences in Classes B, C, D, and E add up to only 377. Because Class A is the largest, this study focused on this class and various training sets were created from Class A.

GPCRDB also maintains a collection of Bacteriorhodopsin proteins. These proteins have seven transmembrane regions as GPCRs. However, as described in Chapter 2, these proteins do not couple with G-proteins, and therefore, they are not GPCRs. The May 2003 release of GPCRDB included 25 such proteins in its Bacteriorhodopsin class. As described below, these bacteriorhodopsin sequences were used as a part of negative dataset.

All of the negative datasets were taken from Swiss-Prot database [5]. Swiss-Prot maintains a collection of protein sequences with detailed function-related annotations. As of October, 2003, the size of Swiss-Prot database was 135,850 entries[2]. Negative datasets longer than 100 amino acids [3] were sampled randomly from this database. They were checked so that none of the negative data belonged to the GPCR superfamily.

[1]Two major updates (in February and June) has been done since May 2003. Because of the elaborate dataset sampling scheme used in this study, newer versions of the database could not be used due to time constraints.

[2]The newest version of Swiss-Prot from July, 2004 consists of 153,871 entries.

[3]The lower bound of 100 amino acids was set so that the negative sequences were not too much shorter than the GPCRs, whose minimum length is close to 300 amino acids.

3.1.2 Training data

A. Sampling methods for preparing positive training sets

In order to examine if any of the algorithms' performance vary with different sampling schemes of training sets, six different sets of positive GPCR data were sampled from Class A. These included one "random sampling" set, one "taxonomical sampling" set, and four "phylogeny-based sampling" sets. Each sampling method is described next.

i) Random sampling

The random sampling set consisted of 200 GPCRs randomly sampled from Class A. This random sampling set may have GPCRs that are very close to each other or very diverged from each other in sequence similarity. Random sampling may not represent all of the subclasses within Class A evenly, especially if some subclasses have only a small number of member proteins. On the other hand, this sampling scheme represents the distribution of GPCR members included in GPCRDB. This training set may not train well if such a bias affects the power of classification models.

ii) Taxonomical sampling

The taxonomical sampling is based on the families and subfamilies of Class A according to the GPCRDB classification scheme. As shown in Figure 3.1, Class A is divided into 16 subfamilies, which are again divided into smaller subfamilies, totaling 87 subdivisions. In this sampling scheme, two sequences were chosen from each of these subdivisions, with a total of 173 sequences (one subdivision had only one sequence). This is the only positive training set consisting of fewer than 200 sequences. The taxonomical sampling set represents all of the subclasses within Class A regardless of the size of each subclass. Therefore, this training set was expected to train better than the random sampling, especially when training and test datasets had different class/subclass distributions.

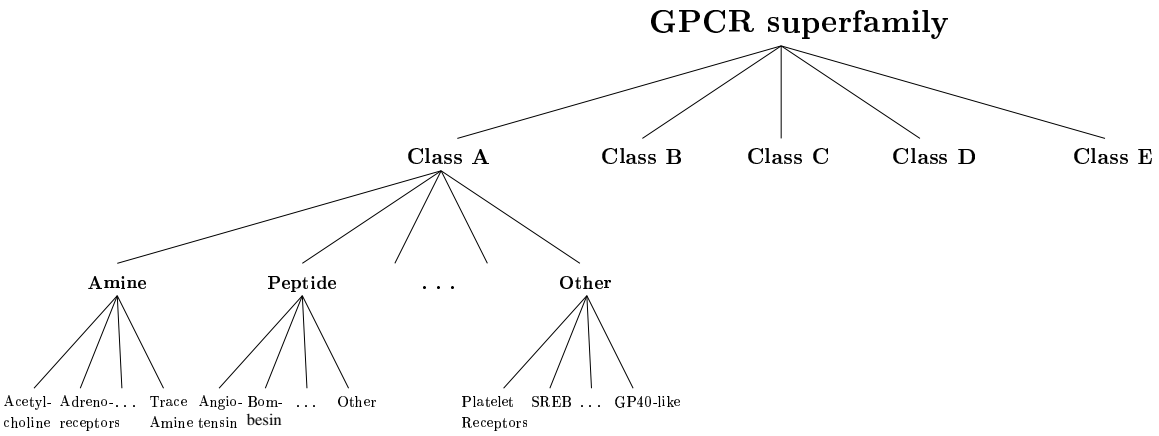

Figure 3.1: Classification scheme of the GPCR superfamily used in GPCRDB. Class A is divided into 16 divisions based on ligand types (e.g., Amine, Peptide, etc.). Each family is further divided into 87 subdivisions in total(e.g., Acetylcholine, Adrenoreceptors, etc. in the Amine family).

iii) Phylogenetic sampling

The phylogenetic sampling is a more complex procedure and needs more explanation. A phylogenetic tree of protein sequences illustrates the evolutionalry relationships among them [28]. It consists of nodes and branches, where the external (or terminal) nodes represent the protein sequences considered. The lengths of the branches are proportional to the numbers of amino acid substitutions estimated between the nodes.

GPCRs are extrememly diverged sequences, and even among the members of Class A their amino acid sequences have high divergence levels among each other. For example, pairwise distances estimated from GPCRs in Class A (based on JTT substitution model [21]) range from 0.00 to 32.27 amino acid substitutions per site with the average distance 2.77. Phylogenetic sampling is used to generate positive training sets that have various levels of divergence from this extremely diverged GPCRs. A phylogenetic tree of all Class A sequences was reconstructed[4] first. To build a phylogenetic tree, distances

[4]The original evolutionary events happen in nature, and phylogenetic methods try to "reconstruct" such evolutionary (divercifying) pathways based on sequence information. Note that any phylogenetic tree "reconstructed" is a hypothesis (or model) of the true evolutionary process.

between sequences were required. In order to avoid a problem generating a multiple alignment from extremely diverged sequences, only pairwise alignments were generated.

The ClustalW program [41] was used to obtain pairwise alignments between all 3,519 sequences of Class A GPCRs. ClustalW uses a progressive algorithm to generate a multiple alignment. However, for a pairwise alignment, it uses simply a dynamic programming. The default set of options: gap opening penalty of 10, gap extenstion penalty of 0.10, and the BLOSUM scoring matrices, was used. Pairwise distances were calculated using the $protdist$ program from the PHYLIP package [11]. The JTT substitution model [21] was chosen for estimating distances. All of the pairwise distances were assembled into one distance matrix. Finally, the $neighbor$ program from the PHYLIP package was used to reconstruct a phylogeny tree. Moriyama et al. (personal communication) compared the phylogenetic trees reconstructed based on pairwise distances and those based on regular distances using multiple alignments. They reported that the pairwise distance-based phylogeny is sufficiently accurate for illustrating major clustering patterns. Therefore, this method should be a good approximation for the purpose used in this study.

The UPGMA (Unweighted Pair Group Method with Arithmetic Mean) method [40], instead of the neighbor-joining (NJ) method [36], was used to reconstruct the phylogenetic relationships. The UPGMA reconstructs a rooted phylogenetic tree assuming a constant evolutionary rate. The NJ method, on the other hand, produces an unrooted tree without any assumption on the evolutionary rate. As described below, the phylogenetic sampling used in this study involves with choosing clusters at a given divergence level. This process becomes much simpler when UPGMA, instead of NJ, phylogenies are used due to the averaging clustering process of UPGMA. It should be noted that the NJ method in general reconstructs more accurate phylogenies than the UPGMA when the evolutionary rate can-

not be assumed to be constant. Although such an assumption is most likely invalid for the GPCR superfamily evolution, since the purpose of using phylogeny in this study is only to determine a rough clustering pattern, using UPGMA should not present a serious problem in this case.

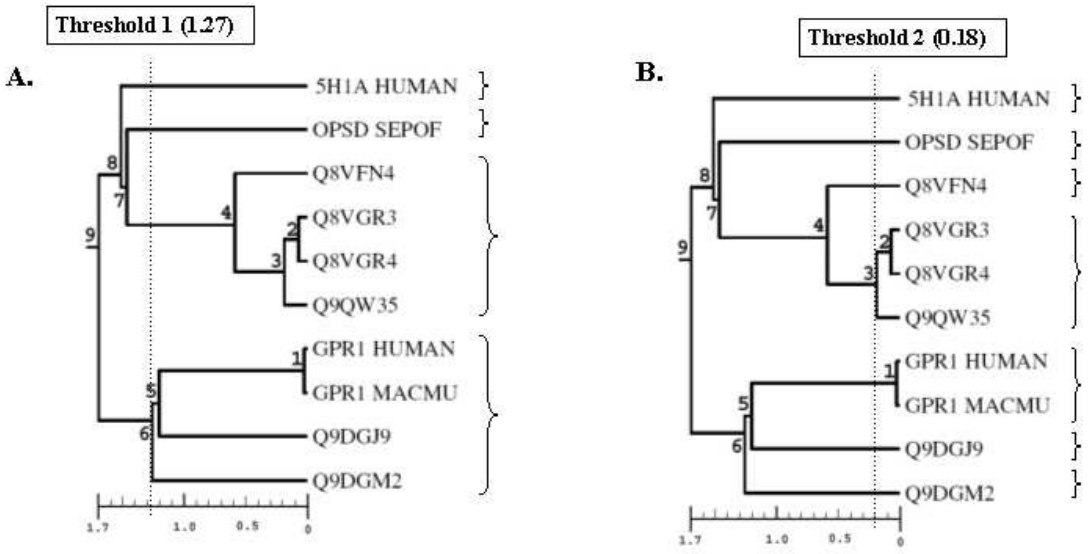

Figure 3.2: A phylogenetic tree of a subset of Class A GPCRs reconstructed by the UPGMA method. Threshold 1 at 1.27 branch length from the leaves gives four clusters of sequences(A). Using Threshold 2 at 0.18 branch length from the leaves, on the other hand, gives seven clusters of sequences as shown in B.

A sample UPGMA phylogenetic tree is shown in Figure 3.2. In the figure, ten sequences were used to reconstruct the phylogenetic tree. At Threshold 1, which is at the distance (branch length) of 1.27 amino acid substitutions per site from the leaves, four clusters can be formed (Figure 3.2A). The first two clusters each consists of a single sequence (5H1A_HUMAN and OPSD_SEPOF). The third cluster includes four sequences under the node 4, and the fourth cluster consists of the remaining four sequences under the node 6. At Threshold 2, which is at 0.18 distance (branch length) from the leaves, seven clusters can be formed (Figure 3.2B). Five clusters each consists of a single sequence, while the

Table 3.1: The numbers of clusters and three levels of thresholds used.

Number of Clusters	Distance threshold[1]
100	0.905
200	0.675
400	0.446

[1] Distance threshold is presented as the branch length from the leaves. To obtain 100 clusters, for example, the branch length of 0.905 amino acid substitutions per site from the leaf nodes was used as the cut-off line. This resulted in 100 clusters that were formed under this threshold.

other two clusters include three sequences under the node 3 and two sequences under the node 1.

After a phylogenetic tree including all the Class A GPCRs (3,519 sequences) was reconstructed, three different levels of distance thresholds were used to obtain 100, 200, and 400 clusters. Table 3.1 summarizes the three levels of distance thresholds. All the distance thresholds are the branch lengths from the leaves of the phylogenetic tree to the cut-off line.

For the first phylogeny-based sampling scheme, the distance threshold used was 0.675. At this threshold, there were 200 clusters as shown in Table 3.1. From each cluster, one sequence was randomly sampled. This resulted in 200 sequences. This method is called the “pure phylogeny-based sampling”. This sampling set is phylogenetically a good representation of Class A GPCRs. Thus, this set was expected to train well all the classification methods.

The second set, called “random phylogeny sampling”, was prepared by first choosing one sequence each from the 400 clusters at the distance threshold of 0.446 as shown in Table 3.1. Next, 200 sequences was randomly chosen from these 400 sequences. The average divergence level of this sampling set is expected to be lower than the pure phylogeny-based

sampling due to its lower threshold. This set incorporates also some random factors. This set may not train as well as the pure phylogeny-based sampling considering its less effective representation of the GPCR families. However, the inclusion of the random sampling could overcome this disadvantage.

For the third set, called “phylogeny and random sampling”, one sequence each was chosen from the 100 clusters with the threshold of 0.905 as shown in Table 3.1. Another set of 100 randomly sampled sequences without overlaps was added to this to make a total of 200 sequences. Phylogenetically, this sampling set represents Class A GPCRs better than random phylogeny sampling and it combines the possible benefit of random sampling.

The fourth set, “two from each cluster sampling”, was created by choosing two sequences each from the 100 clusters with branch length of 0.905. This resulted in 185 sequences since 15 clusters had only one sequence. Fifteen randomly sampled sequences were added to make a total of 200 sequences. The last three phylogenetic sampling schemes incorporated random samplings. Comparing them with the pure phylogeny-based method as well as the taxonomic sampling method, the effect of random sampling on the training could be evaluated.

B. Sampling methods for preparing negative training sets

Two sets of negative data containing non-GPCR sequences were prepared. For the first set, 200 non-GPCR sequences were randomly chosen from Swiss-Prot protein database. They consisted of any kind of protein sequences that were at least 100 amino acids long but were not GPCRs.

For the second set, a set of bacteriorhodopsin proteins were added to the first negative dataset. Bacteriorhodopsins are seven transmembrane proteins like GPCRs but they are not actually GPCRs, as described in Chapter 2. Addition of such proteins is expected to train the learning methods more specifically and with fewer false positives, resulting in better classification performance on the test sets. This is because the learning methods will learn to discriminate the protein sequences on properties other than the existence of seven transmembrane regions. There were a total of 25 bacteriorhodopsins in the May 2003 release of GPCRDB. Ten sequences were sampled out of 25 so that they were diverged as much as possible. A multiple alignment of these proteins were generated using ClustalW. Bootstrap analysis [5] with 1,000 replications was done by using $seqboot$, a program from the PHYLIP package. The PHYLIP program $protdist$ was used to estimate distances with the JTT method [21]. The PHYLIP program $neighbor$ was used to reconstruct neighbor-joining trees. The program $consense$ from PHYLIP package was used to calculate boostrap supporting scores. Strongly supported clusters, with bootstrap values of 95% or higher, were identified and ten sequences of bacteriorhodopsin were chosen to represent the entire diversity in the tree. These ten sequences were added to the previously created negative training set. The total number of sequences in the second negative training set was therefore, 210.

C. Twelve training sets

After putting together the six positive training sets with two negative training sets, the total number of training sets was twelve. They are summarized in Table 3.2.

[5] Bootstrap analysis is a method of generating multiple data sets that are resampled from the original input data set [11]. For phylogenetic analysis, bootstrap analysis is done on the multiple alignment, and bootstrap supporting values are calculated for each node in the phylogeny. Higher the values, the nodes (or the clusters) are supported more.

Table 3.2: The training sets used in this study.[1]

Training Set ID	Positive dataset	Negative dataset
1a (400)	Random (200)	Random (200)
2a (373)	Taxonomical (173)	Random (200)
3a (400)	Pure phylogeny (200)	Random (200)
4a (400)	Random phylogeny (200)	Random (200)
5a (400)	Phylogeny and Random (200)	Random (200)
6a (400)	Two from each cluster (200)	Random (200)
1b (410)	Random (200)	Random and Bacteriorhodopsin (210)
2b (383)	Taxonomical (173)	Random and Bacteriorhodopsin (210)
3b (410)	Pure phylogeny (200)	Random and Bacteriorhodopsin (210)
4b (410)	Random phylogeny (200)	Random and Bacteriorhodopsin (210)
5b (410)	Phylogeny and Random (200)	Random and Bacteriorhodopsin (210)
6b (410)	Two from each cluster (200)	Random and Bacteriorhodopsin (210)

[1] The numbers in the parantheses show the number of samples in each dataset.

3.1.3 Test data

The performance of each classification algorithm was tested against test datasets that included both positive and negative data. All the sequences used in the test data are exclusive from those in the training data.

A. Full-sequence test

Two positive test sets were created. The first dataset consisted of 200 sequences of randomly sampled Class A GPCRs from GPCRDB. The second set consisted of 200 sequences of randomly sampled GPCRs of Classes B, C, D, and E from GPCRDB. This second test set was created to examine how well the methods trained on Class A GPCRs would classify non-Class A GPCRs.

As described for the training sets, the negative samples for the test datasets were ob-

Table 3.3: The test sets used in this study.[1]

Test Set ID	Positive	Negative
t1 (410)	Random - Class A (200)	Random and Bacteriorhodopsin (210)
t2 (410)	Random - Class B, C, D, E (200)	Random and Bacteriorhodopsin (210)

[1] The numbers in parantheses show the number of samples in each dataset.

tained from the Swiss-Prot database. Only one set of negative test data was created. The set consisted of 200 randomly sampled non-GPCRs that were at least 100 amino acid long and a set of 10 bacteriorhodopsin protein sequences. The bacteriorhodopsin sequences were sampled in the same way as explained in Section 3.1.2 and they were not overlapped with those in the training set. The negative test set contained 210 sequences.

The two test datasets are summarized in Table 3.3.

B. Subsequence test

The test set containing 200 Class A GPCRs was used to create another sets of test data with amino acid sequences of different lengths. Six test sets were created, where each of the test sets consisted of a particular length of subsequences. The amino acid lengths chosen were: 50, 75, 100, 150, 200, and 300. These were random subsequences of the full GPCR sequences. These lengths were chosen because the average length of GPCRs in all the training sets was 397, ranging from 263 to 1,050 amino acids. These test sets were used to examine how well the classification methods perform with short subsequence of the real GPCRs, and how the performance changes with increasing subsequence lengths.

3.2 Protein Classification Methods

The purpose of this thesis is to compare and analyze different classification methods and their performance and the effect of various training datasets on these methods. The problem given to these methods is to classify two types of data, GPCR and non-GPCR. The methods used are profile hidden Markov models, support vector machines using several kernel functions and several type of feature vectors, and decision trees. In this section, these methods and the attributes used by these methods are described.

3.2.1 Profile-HMM

As a representative of the commonly used protein classification methods, a profile HMM method is included in this study, and the classification performance of other methods was examined against this method. Sequence Alignment and Modeling (SAM) software system [16, 24, 26] is an implementation of the profile HMM method for protein classification. SAM (version 3.4) was used for this study. Fasta format of unaligned sequences of the positive training sets including only GPCR sequences was used as input. Dirichlet mixture priors [38] were used to build the models so that they have better probability distributions. For building the models, the following command and options were used.

```
>buildmodel train_model -train trainset.fas
  -prior_library uprior.9comp -randseed 0
```

Here `trainset.fas` is the input file, `uprior.9comp` is the library of Dirichlet mixtures, and `train_model` is the name given to the model built by SAM and it is saved in the `train_model.mod` file. The `-randseed` parameter is for the selection for initial model length. The default value of `-randseed` is the process ID number. Here it is set to 0 so that the program run is reproducible. The test sequences in fasta format from each of the test sets were compared to each of the models using the following command.

```
>hmmscore outfile -i train_model -db testset.fas
  -sw 2 -calibrate 1
```

The output file (named `outfile.dist`) contains the e-values of the scores for each test sequence from the file `testset.fas` based on the model given in `train_model`. The option `-sw` is to specify the type of alignment. Setting `-sw` to 2 performs full-local alignment of sequences to the model. The `-calibrate` parameter with the value 1 is used for a better calibration of the e-values. The sequences in the output file are ranked according to the e-values. Classification was done using a certain e-value threshold. Sequences with the e-values lower than or equal to the threshold were classified as "positives" (GPCRs), while those with the e-values higher than the threshold were identified as "negatives" (non-GPCRs). For accuracy rate calculation (described later), the e-values threshold was found using the minimum error point described in Section 3.3.3.

3.2.2 SVM with amino acid frequencies

A support vector machine package SVM-light version 5.0, which is an implementation of support vector machines by Joachims [20] was used. Both positive and negative sequences (GPCRs and non-GPCRs) from the training sets were used to train the SVM.

For this study, simply nineteen amino acid frequencies of each protein sequence were used as the input vector for SVM-light. Four kernel functions (linear, polynomial, sigmoid, radial basis) were used to create a hyperplane for classification and the performance of each kernel function with SVM-light is compared. The four kernel functions are listed in Table 2.2.

The default kernel function used by SVM-light is linear kernel. For training the SVM with linear kernel function, the following command was used:

```
>svm_learn train.dat train_model
```

`train.dat` is the input file with the 19 amino acid frequencies of each training sequence in a vector format, and `train_model` is the model built by SVM based on the training data.

For training the SVM with the polynomial, sigmoidal, and radial basis kernels, the following commands were used:

```
>svm_learn -t 1 -s 1 -r 1.0 -d 100 train.dat train_model

>svm_learn -t 3 -s 10 -r 0.1 train.dat train_model

>svm_learn -t 2 -g 150 train.dat train_model
```

where the parameter `t` is the kernel function option where 0 is for linear (default), 1 is for polynomial, 2 is for radial basis, and 3 is for sigmoid kernel function. The other options are used to define the parameter values for the kernel functions. They were decided after various different values were tried. Table 3.4 summarizes the parameters used.

Table 3.4: The values used for parameters in SVM with polynomial, sigmoid, and radial basis kernel functions.

Kernels	Parameters[1]			
	k	c	p	γ
Polynomial	1	1.0	100	-
Sigmoid	10	0.1	-	-
Radial basis	-	-	-	150

[1] The parameters for each kernel are listed in Table 2.2.

For classifying the test set sequences, the following command was used for all the kernel functions.

```
>svm_classify test.dat train_model output_file > result_file
```

The `test.dat` is the test file with 19 amino acid frequencies for each test sequence in a vector format. The `train_model` is the model built from the training set and `output_file` consists of the predictions. The `result_file` contains the classification statistics including the accuracy of the SVM on the test set.

3.2.3 SVM-pairwise

SVM-light (version 5.0) package with the radial basis kernel function was used for the SVM-pairwise method. The input vectors for the SVM were the e-values of pairwise similarity scores between all sequences. E-values were derived using the $SSearch$ (version 3.4) program, which is an implementation of Smith-Waterman local alignment algorithm [39, 33]. The default options of $SSearch$ (open gap penalty of 12, gap extension penalty of 2, and the BLOSUM50 scoring matrix) were used.

From the training sets, each sequence was compared against each one included in the training sets. From each comparison, the e-values was calculated and put into a vector format. Each vector contained, for example, 400 e-values (200 from the comparison with positive samples and 200 from those with negative samples). Next, the input vectors were used to train a SVM using the SVM-light program, `svm_learn`. The radial basis function with γ = 0.0001 was used as the kernel function.

For the test sets, $SSearch$ program was used to obtain e-values for each test sequence against the training sets. The input vector file including these e-values was used with the `svm_classify` program to obtain the prediction results and the accuracy of the SVM on the test set.

Table 3.5: The profile HMMs and corresponding training and test sets used for SVM-Fisher method.[1]

SAM HMM	Fisher scores for training	Fisher scores for testing
M1	F_1a	F_t1_M1, F_t2_M1
	F_1b	F_t1_M1, F_t2_M1
M2	F_2a	F_t1_M2, F_t2_M2
	F_2b	F_t1_M2, F_t2_M2
M3	F_3a	F_t1_M3, F_t2_M3
	F_3b	F_t1_M3, F_t2_M3
M4	F_4a	F_t1_M4, F_t2_M4
	F_4b	F_t1_M4, F_t2_M4
M5	F_5a	F_t1_M5, F_t2_M5
	F_5b	F_t1_M5, F_t2_M5
M6	F_6a	F_t1_M6, F_t2_M6
	F_6b	F_t1_M6, F_t2_M6

[1] Six SAM models and 12 training sets were used to create the 12 sets of Fisher score vectors for training. For example, F_1a is the training set with Fisher scores obtained from the model M1 and the training set 1a. For testing, two test sets and the six SAM models were used to create 12 sets of Fisher score vectors. For example, F_t1_M1 is the test set with Fisher scores obtained from the model M1 and the test set t1.

3.2.4 SVM-Fisher

SAM (version 3.3.1) and SVM-light (version 5.0) were both used for this method. The profile HMMs built by SAM earlier as explained in Section 3.2.1 was used for this method.

Both positive and negative sequences from each training set was compared with its corresponding SAM model, and the Fisher scores [18] were extracted for each sequence. Table 3.5 summarizes the relationships among the several input data used for this method. For example, the HMM M1 was built using positive sequences of $1a$ training set, and both $1a$ and $1b$ training sets with positive and negative sample sequences were compared with the model M1 to obtain the Fisher score vectors (named F1_a and F1_b in the table). The following command from SAM was used to derive Fisher scores:

```
>get_fisher_scores unused -fisher_feature match_prior
  -i train_model -db trainset.fas -sw 2
```

where `unused` is the run-name (not used in this program) and `-fisher_feature` is the parameter that specifies which features are used to calculate the Fisher score vectors. The `match_prior` option given directs to use only the match states and Dirichlet mixture used to train the model. The SAM model file is specified with `-i` option, and `-db` specifies the dataset file whose Fisher score vectors need to be calculated.

This resulted in 12 training sets of Fisher score vectors (F_1a to F_6b in Table 3.5). These Fisher score vectors were then used to train the SVM with a radial basis kernel function (γ = 0.0001) using the `svm_learn` program in SVM-light.

Each sequence of the two test sets (t1 and t2 as shown in Table 3.3) was compared against each of the six SAM models, and the Fisher scores were obtained by using the `get_fisher_scores` program as described above. The Fisher score vectors obtained from each test set (F_t1_M1 to F_t2_M6 in Table 3.5) were classified using the `svm_classify` program in SVM-light, based on each trained SVM model. For example, as listed in Table 3.5, the SVM trained with the F_1a input vectors was tested against both of the F_t1_M1 and F_t2_M1 test vectors.

3.2.5 Decision trees

Decision trees method, another pattern recognition method, was also used in this study for comparison purposes. This method has rarely been used for protein classification problems. Decision trees work by discovering rules that best classify the given data. Each attribute of the data is evaluated first to see which one can classify the data best. That attribute is then chosen as the root node, and descendant nodes are created by sorting the examples to

appropriate branches. The entire process is repeated at each newly formed node. Once all the nodes have only one type of the classes associated with it, the process stops. The main idea is to choose tests at each node that maximally separates the data so that the final tree is small.

Since this method does not require multiple alignments, and works differently than the SVMs, it is interesting to see how this method works compared with others. For this method, I used Quinlan's c4.5 (release 8) program [35]. The 19 amino acid frequencies of both positive and negative training sequences were used as input to the program.

The following commands were used for training and testing.

```
>c4.5 -f filestem -u
```

where the parameter `-f` is to specify a filestem name. Three sets of files are used as inputs to c4.5. `filestem.data` consists of the 19 amino acid frequencies (attributes) of each sequence in the training set, followed by its label. `filestem.names` consists of the attribute names and the class names the data belongs to. `filestem.test` consists of the 19 amino acid frequencies of each sequence in the test set. The parameter `-u` is to specify c4.5 to classify the test data in the `filestem.text` file. The output given is a form of a confusion matrix (explained in the next section), which includes the numbers of true positives, false positives, true negatives, and false negatives.

Decision tree with boosting was also performed. In boosting, many decision tree classifiers are built from the same training set by changing the weights[6] of the misclassified vectors at each iteration [34]. The weights of all the vectors are normalized at every it-

[6]A weight for each instance is maintained in boosting. The higher the weight, the more the effect the instance has on the classifier [34].

eration. When the total weight of the misclassified vectors add up to 0 or is greater than 0.5, the program stops building trees. The test vectors are passed through each decision tree classifiers A decision on the class of a sequence is based on the majority vote by the classifiers.

The Weka implementation was used for the boosted decision trees [44]. In the Java interface, the amino acid frequencies of the training set sequences and the amino acid frequencies of the test set sequences were loaded. The decision trees program J48 [35] (equivalent to c4.5 [44]) and the boosting program AdaBoostM1 [37] were used with the following command:

```
>AdaBoostM1 -P 100 -S 1 -I 10 -W J48 -- -C 0.25 -M 2
```

where `-P` specifies the weight mass to be used, `-S` specifies the random number seed, `-I` specifies the number of iterations, and `-W` specifies the learning program (J48 is used here). The parameters for J48 include: `-C` specifies for the confidence threshold for pruning and -M specifies for the minimum number of instances in a leaf node of the decision tree. The output consisted of a confusion matrix with true predictions and false predictions by the classifier.

3.3 Performance Analysis

Classification performance of each method was analysed using various statistics. Their performance was examined against independently prepared test datasets described in Section 3.1.3. Cross-validation analysis was also performed. The accuracy rate is the simplest measure for the classification performance. More detailed analysis is done using the Receiver Operating Characteristic (ROC) curve. Performance analysis used by Karchin et

al. [22], which examines the minimum error point, maximum and median rate of false positives, was also performed.

3.3.1 Cross-validation

Cross-validation analysis was performed for all the experiments in this study. Cross-validation analysis is also called "leave-one-out" method. It is done as follows. One item from the training dataset is left out and the learning algorithm is trained on the rest of the items. The trained model is used to predict the label of the one left out earlier. For n sequences in the training set, this process is repeated n times leaving each of the n sequences out and creating a model from the remaining $n - 1$ sequences. The accuracy for each method is calculated as described next.

3.3.2 Confusion matrix and accuracy rate

A confusion matix is a 2×2 table showing the number of real sequences and the number of predicted sequences by a classifier. In a confusion matrix, as shown in Figure 3.3, there are four items:

- *True Positives (TP)*: Number of actual GPCRs that are predicted as GPCRs.
- *False Positives (FP)*: Number of actual non-GPCRs that are predicted as GPCRs.
- *True Negatives (TN)*: Number of actual non-GPCRs that are predicted as non-GPCRs.
- *False Negatives (FN)*: Number of actual GPCRs that are predicted as non-GPCRs.

The accuracy rate is defined as the proportion of correct predictions and is given by:

$$\frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}}.$$

		Predicted Label	
		+	−
Actual Label	+	True Positive (TP)	False Negative (FN)
	−	False Positive (FP)	True Negative (TN)

Figure 3.3: A confusion matrix with true positive, false positive, true negative, and false negative.

The minimum, average, and maximum accuracy rates were calculated for each method trained on different training sets. The accuracy rate was calculated from independently prepared test sets as well as from the cross-validation test.

3.3.3 Minimum error point

The minimum error point (MEP) is one of the performance measures used by Karchin [22]. Each classifier outputs a score for each prediction as an example shown in Table 3.6. The magnitude of these scores reflects the classifier's confidence in the prediction. The test sequences are ranked based on the scores. The threshold score where the minimum number of errors (FN + FP) occurs is the minimum error point (MEP) and the number of false positives and false negatives are assessed at this point. The minimum error point tells us the best case accuracy of a classifier. In Table 3.6, the MEP is obtained with the score 14, where the number of errors is only 3.

The minimum error point is calculated for all methods except the decision trees methods. These methods do not give out prediction scores for test sequences. Instead only a final result is given in a confusion matrix. Therefore, it was not possible to do this performance analysis on these methods.

Table 3.6: An example for calculating the minimum error point, maximum and median rates of false positive.

Rank	Seq. ID	Score	Actual Label	FP	FN	Error	FP rate
1	seq1	20	+	0	7	7	0
2	seq2	19	+	0	6	6	0
3	seq3	18	+	0	5	5	0
4	seq4	17	+	0	4	4	0
5	seq5	16	-	1	4	5	0.125
6	seq6	15	+	1	3	4	0.125
7	**seq7**	**14**	**+**	**1**	**2**	**3**	**0.125**
8	seq8	13	-	2	2	4	0.25
9	seq9	12	-	3	2	5	0.375
10	seq10	11	+	3	1	4	0.375
11	seq11	10	-	4	1	5	0.5
12	seq12	9	-	5	1	6	0.625
13	seq13	8	-	6	1	7	0.75
14	seq14	7	-	7	1	8	0.875
15	seq15	6	+	7	0	7	0.875
16	seq16	5	-	8	0	8	1

The sequences are ranked according to the prediction score. The numbers of false positives (FP) and false negatives (FN) are those obtained by setting the threshold at that score. The number of errors (Error) is given as FP+FN at each threshold score. The minimum error point (MEP) is the threshold score where the number of errors is minimum. In this example, the minimum error is 3 with the threshold score 14 and this is the MEP (as shown in boldfaces). The maximum rate of false positives (MaxRFP) is the false positive rate with a certain threshold score where all the positive sequences are identified. Here, MaxRFP is 0.875 with the threshold score 6. The median rate of false positives (MedRFP) is the false positive rate with a certain threshold score where only a half of the positive sequences are identified. Here, MedRFP is 0 with the threshold score 17.

3.3.4 Maximum and median rate of false positives

The maximum rate of false positives (MaxRFP) and median rate of false positives (MedRFP) were also used by Karchin [22]. The MaxRFP is the rate of false positive with a certain threshold score where all the positive sequences are identified. For example, in Table 3.6, to identify all the positive sequences, the threshold needs to be at the score 6. At this threshold, the false positive rate is 0.875, which is the MaxRFP for this example.

The MedRFP is the rate of false positive with a certain threshold score where only a half of the positive sequences are identified. For example, in Table 3.6, a half (four) of the positive examples are identified when the threshold score is 17. The false positive rate is 0 at this point and this is the MedRFP for this example. The lower the MaxRFP and MedRFP, the better the classifier.

MaxRFP and MedRFP is calculated for all methods except decision trees for the same reasons mentioned earlier.

3.3.5 Receiver operating characteristic

The Receiver Operating Characteristic (ROC) is a popular method used today for performance analysis of different machine learning methods [10]. It is a graph plotting false positive rates on the x-axis and true positive rates on the y-axis. It shows the trade-offs between benefits (true positives) and costs (false positives) of a certain classifier. It is generated by cutting off the decision scores at different thresholds and by calculating the true positive rate and false positive rate for each threshold as done for MEP, MaxRFP, and MedRFP calculation. ROC plots make it easy to visualize and compare the performance of different classifiers.

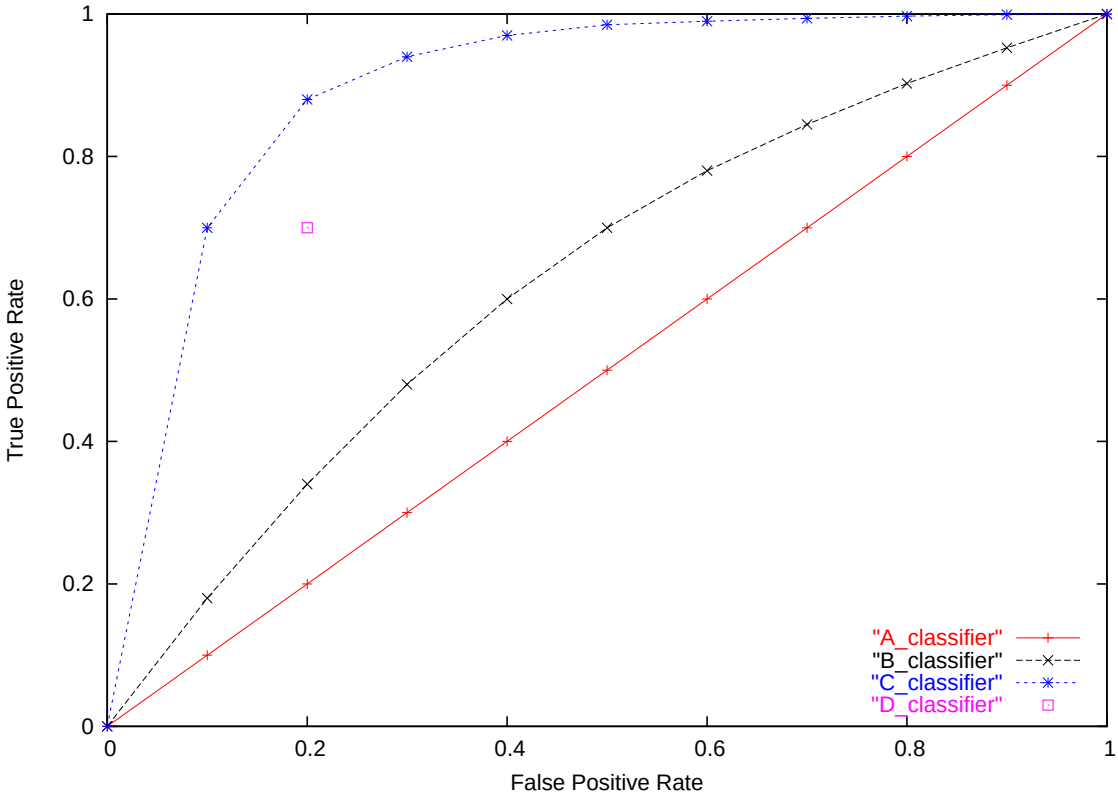

Figure 3.4: A ROC graph of four different classifiers. Classifier C is the best with the largest area under the curve, while Classifier A is the worst with the smallest area under the curve. A discrete Classifier D is drawn for a classifier that does not produce ranked scores.

An example of a ROC graph is shown in Figure 3.4. In the graph, the point (0,0) means that the classifier gives out no false positive error, but also gives out no true positives. At this point, everything is predicted as negative. At point (1,1) the classifier predicts everything as positive making both true positive and false positive rates as 1. At point (1,0) no true positive is produced. A perfect classification is represented by the point (0,1) where no false positive is produced and all positive sequences are identified correctly.

The area under the curve of a perfect classifier is 1. Therefore, we can look at the area covered by ROC graphs of different classifiers and analyse its performance.

In Figure 3.4, Classifier C is the best with the largest area under the curve, while Classifier A is the worst with the smallest area under the curve. A discrete classifier D is drawn for a classifier that does not produce ranked scores.

Decision trees are discrete classifiers. They only give a final classification result in a confusion matrix as described before. Therefore, instead of a curve, a point is plotted in the ROC graphs for these methods.

Chapter 4

Results and Discussion

As described in Chapter 3, the nine protein classification methods were trained on twelve different training sets. All of the methods were tested on three different types of test sets. In this chapter, first, the results from the test on identifying Class A GPCRs is described. In the second section, the test results to identify other GPCR classes are discussed. Finally, the methods are tested on the subsequence test sets and their results are discussed.

4.1 Identification of Class A GPCRs

To examine how each of the classification methods performs when trained on different training sets, first the test set (t1) containing Class A GPCRs (positive samples) and non-GPCRs (negative samples) were used to test the performance of the nine methods.

4.1.1 Accuracy rates

The accuracy rates of the nine classifiers trained on 12 different datasets are listed in Table A.1 in Appendix A. These results are summarized in Figure 4.1. The figure shows the maximum, minimum, and average accuracy rates of the nine methods. What we notice first

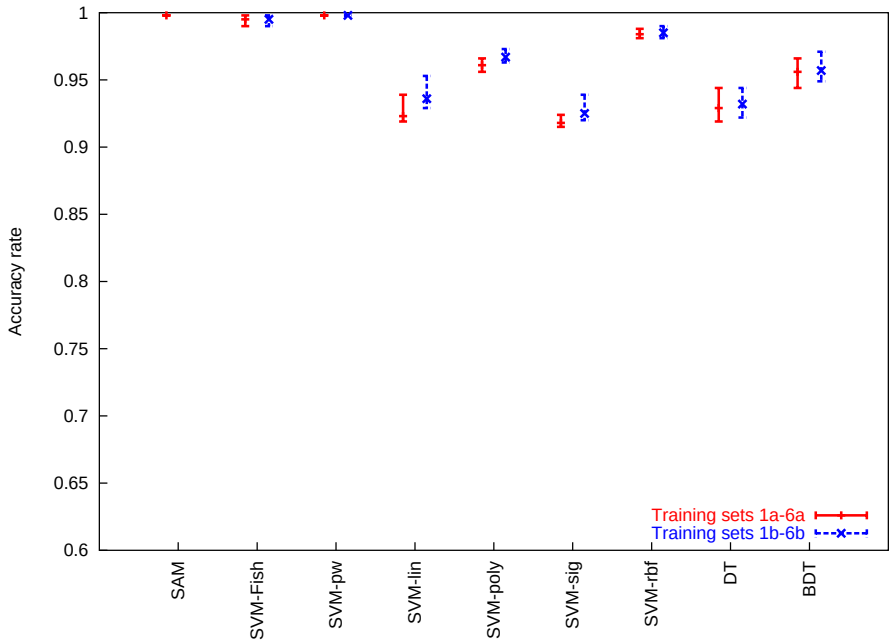

Figure 4.1: Accuracy rates of the nine classification methods trained on different datasets. Tests were done on the Class A GPCR datasets (t1). The average rates are plotted with bars showing the range from the minimum to the maximum rates obtained among different training sets. The results are summarized from Table A.1. All the accuracy rates except for SAM are produced by the programs used. The accuracy rate for SAM is the one at the minimum error point. For method name abbreviations, see the footnotes of Table A.1.

from these results is that all of the methods compared in this study have higher than 91% accuracy. Note that except for SAM, SVM-pairwise, and SVM-Fisher, these methods do not rely on multiple alignments. Attributes used are simple amino acid composition. However, all methods managed to discriminate Class A GPCRs correctly at a high accuracy.

The four best methods for classifying Class A GPCRs from non-GPCRs were SAM, SVM-pairwise, SVM-Fisher, and SVM using the radial basis kernel functions with amino acid frequencies (SVM-rbf). All of these three best SVM methods use radial basis kernel function. Also, both SVM-Fisher and SAM are the classifiers based on profile-HMMs. Among the different kernels used, the sigmoid function turned out to be the worst. It was also observed that the boosted decision trees performed better than the regular decision trees, although neither of them performed as well as the four best methods.

In order to examine the effect of different training sets on the classification methods, accuracy rates were compared among those based on different training sets. Details are shown in Table A.1. The training sets made no difference in classification by SAM and SVM-pairwise; regardless of the training sets used both maintained 99.8% accuracy. The best classifiers, SAM, SVM-Fisher, SVM-pairwise, and SVM-rbf showed very small effects of training sets on their performance. On the other hand, the methods wtih lower accuracy rates showed more varied performance depending on the training sets.

Using the training sets prepared by phylogenetic samplings (3a/b-6a/b) was expected to be better than that trained with random samplings (1a/b) from GPCRDB. However, the results obtained did not show any particular advantage of using such sampling method. Alternatively, random sampling was sufficiently good or frequently better than the other sampling methods. Note also that taxonomical sampling following the GPCRDB classifi-

cation system (2a/b) seemed to show a slightly better performance for the two better SVM methods using amino acid composition: SVM-rbf and SVM-poly, although as described before their performance varied only slightly depending on the training sets.

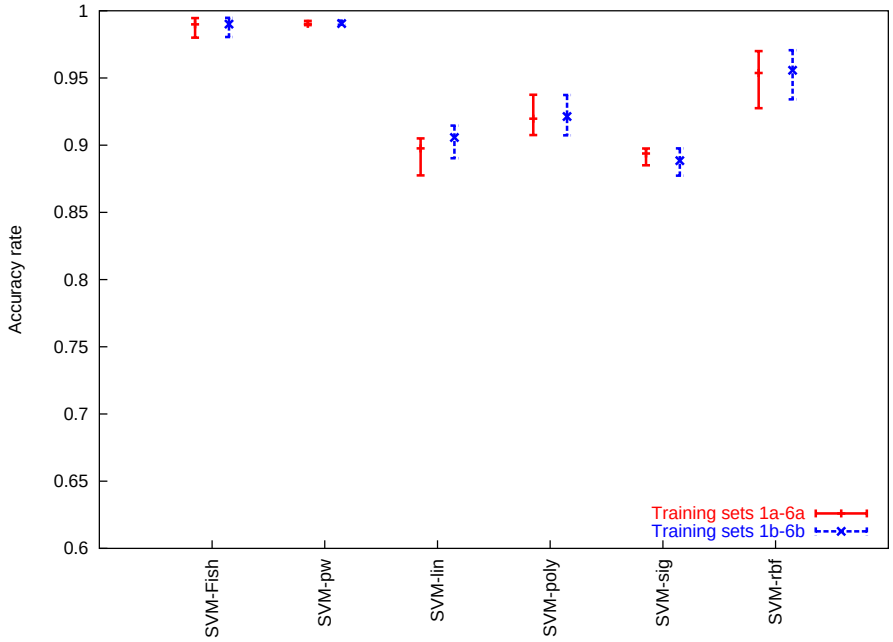

Figure 4.2: Accuracy rates of cross-validation tests of the six SVM-based classification methods trained on different datasets. The average rates are plotted with bars showing the range from the minimum to the maximum rates obtained among different training sets. The information is summarized from Table A.2.

Cross-validation tests were done also for the six SVM based methods. The results are summarized in Figure 4.2. It shows consistent results as discussed earlier, and SVM-Fisher, SVM-pairwise, and SVM-rbf performed very well.

A slight difference in performance of the classification methods was observed when bacteriorhodopsin sequences were added to the negative training sets as shown in Figures 4.1 and 4.2 (green lines for the training sets 1b-6b). The average, maximum, and minimum accuracy rates seemed to increase slightly or remained the same when the clas-

Table 4.1: The minimum error point, maximum and minimum rates of false positives for the Class A GPCR test set classification.

Method	Training Set	MEP[1]					MaxRFP[1]	MedRFP[1]
		Errs[2]	TP	FP	TN	FN		
SAM	1a	1	200	1	209	0	0.005	0.005
SVM-Fisher	1a	1	200	1	209	0	0.005	0
SVM-pairwise	1a	1	200	1	209	0	0.005	0
SVM-linear	1a	25	192	17	193	8	0.300	0.019
SVM-polynomial	1a	15	191	6	204	9	0.176	0.005
SVM-sigmoid	1a	31	189	20	190	11	0.324	0.019
SVM-radial basis	1a	7	199	6	204	1	0.067	0.005

[1] Refer to Chapter 3 for the description of MEP, MaxRFP, and MedRFP.
[2] Errs = FP+FN.

sification methods were trained on datasets 1b-6b.

4.1.2 Minimum error point and false positive rate analysis

Table 4.1 summarizes the minimum error point (MEP) analysis. The table shows the results only from training set 1a (randomly sampled data), since using other training sets did not show any significant difference. SAM, SVM-pairwise, and SVM-Fisher performed extremely well in discriminating Class A GPCRs from non-GPCRs as mentioned earlier. The number of errors made by these methods at the MEP was only one. SVM-rbf was the second best with seven errors, and almost all these errors were from the false positives (6 out of 7); it means that very few actual GPCRs are missed by this classifier. SVM-sigmoid was the worst classifier of all at the MEP, and one third (11/31) of the errors were from false negatives.

Based on the MaxRFP and MedRFP analysis, where lower numbers mean better classifiers, the same performance pattern is clearly seen. The lowest MaxRFP was for SAM,

Table 4.2: The minimum error point, the maximum and minimum rates of false positive when the classification methods were trained on datasets including bacteriorhodopsin sequences (1b).

Method	Training Set	MEP[1]					MaxRFP[1]	MedRFP[1]
		Errs[2]	TP	FP	TN	FN		
SAM	1a	1	200	1	209	0	0.005	0.005
SVM-Fisher	1b	1	200	1	209	0	0.005	0
SVM-pairwise	1b	1	200	1	209	0	0.005	0
SVM-linear	1b	19	193	12	198	7	0.310	0.014
SVM-polynomial	1b	14	194	8	202	6	0.148	0.005
SVM-sigmoid	1b	25	193	18	192	7	0.371	0.019
SVM-radial basis	1b	7	199	6	204	1	0.067	0.005

[1] Refer to Chapter 3 for the description of MEP, MaxRFP, and MedRFP.
[2] Errs = FP+FN.

SVM-pairwise, and SVM-Fisher. SVM-sigmoid had the worst MaxRFP. MedRFP did not show a large difference among different methods. This implies that 50% of the GPCR samples were easily identified by any method. However, considering the larger difference found in MaxRFP, better classifiers could identify more difficult GPCRs much more correctly.

Table 4.2 summarizes the results when the training sets include bacteriorhodopsin as the negative samples. Note that SAM does not use negative samples to build HMMs. Thus SAM in Table 4.2 lists the same results as in Table 4.1. The performance did not change for the three classifiers: SVM-pairwise, SVM-Fisher, and SVM-rbf. The number of errors decreased only slightly in other SVM methods. Overall, we did not see a large positive effect of including bacteriorhodopsin in the training sets for the classifiers compared.

Figure 4.3 shows the Receiver Operating Charateristic (ROC) analysis. It is again clear that SAM, SVM-Fisher, SVM-pairwise, and SVM-rbf performed the best. It is also evident

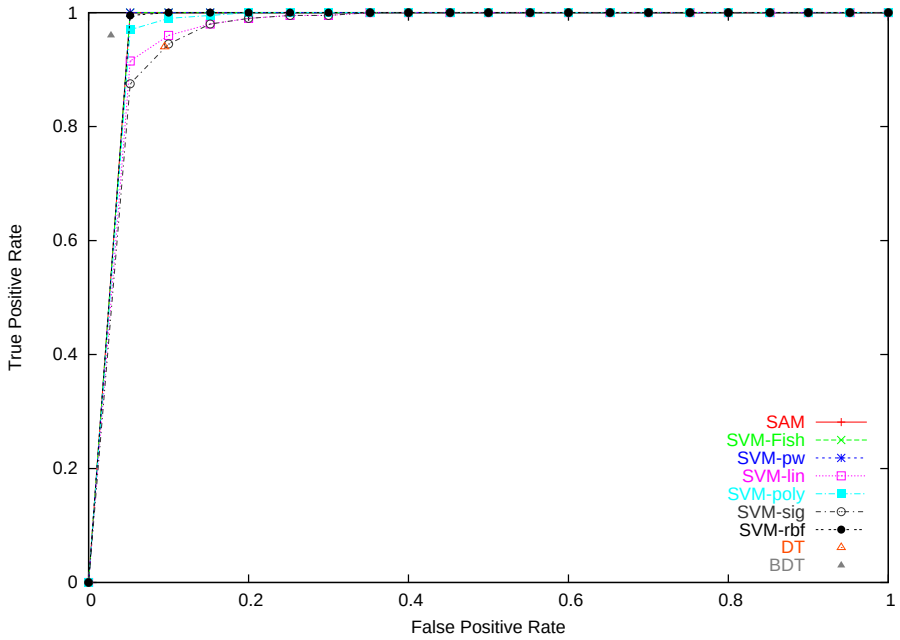

Figure 4.3: The ROC curves of the nine methods for the Class A GPCR test set (t1). The methods were trained on training dataset 1a. Similar ROC curves were obtained when other training sets were used.

from the graph that all the methods compared were competitive and worked very well since the ROC graph of all the methods are close to having an area of 1. Classifiers trained on training sets with bacteriorhodopsin (1b) did not make much difference in the ROC curves (data not shown).

4.2 Identification of the Other Classes of GPCRs

Since all the positive training data were taken from Class A GPCRs, it was interesting to see how well the methods trained on one particular class of GPCRs would identify those from the other classes. The performance from this test could indicate how well each method can predict noble GPCRs based on the existing data. This section describes the performance of the nine methods tested on the test set including GPCR sequences other than Class A (t2).

4.2.1 Accuracy rates

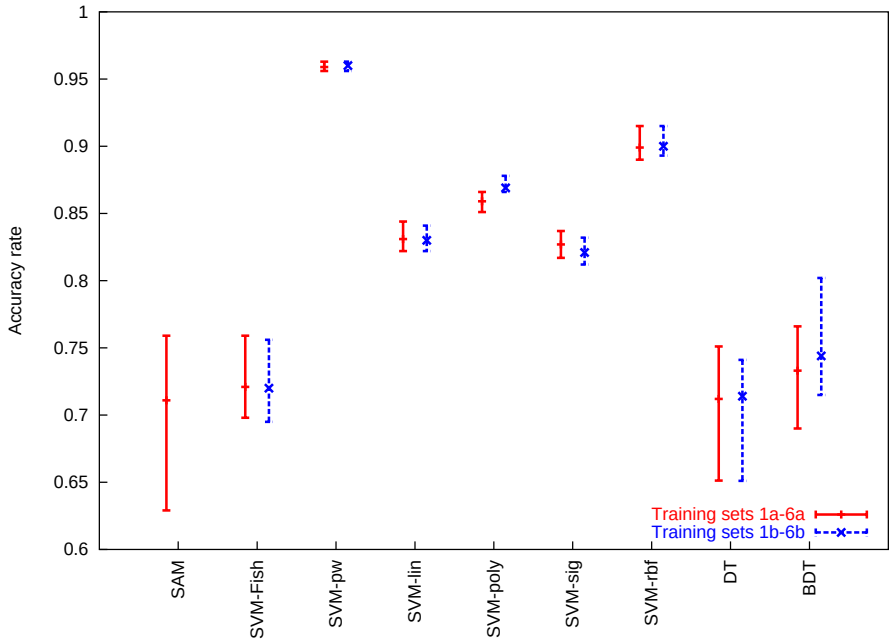

Figure 4.4: Accuracy rates of the nine classification methods trained on different datasets. Tests were done on the non-Class A GPCR datasets (t2). The average rates are plotted with each bar showing the range from the minimum to the maximum rates obtained among different training sets. The results are summarized from Table A.3. For the method name abbreviations see the footnotes of Table A.3

The accuracy rates of the nine methods trained on 12 different datasets are listed in Table A.3 in Appendix A. Figure 4.4 summarizes these results. The best methods were SVM-pairwise and SVM-rbf. Unlike the Class A GPCR identification, SAM and SVM-Fisher performed poorly in this case. This is because both SAM and SVM-Fisher use HMMs built from the Class A GPCRs, and these models may be specific to that class. The higher specificity of HMMs contributed to very low errors, for the Class A GPCR classification especially low false positive rates of SAM and SVM-Fisher compared to other non-alignment based methods (e.g., SVM-rbf) as described in the previous section. However, GPCRs from different classes share low sequence similarities and the lengths also vary among the classes; The average lengths of Classes A and B, for example, are 397 and

746 amino acids respectively. This could explain why GPCRs from other classes were not identified well by the HMM based methods. All of the SVM methods using amino acid frequencies and different kernel functions performed better than SAM, SVM-Fisher, as well as the decision trees. Their performance was not affected by the use of different training sets.

The inclusion of bacteriorhodopsin sequences in the training sets did not show much difference in performance. The addition of these sequences was not helpful in increasing the performance of the methods when identifying non-Class A GPCRs.

4.2.2 Minimum error point and false positive rate analysis

Table 4.3: The minimum error point, maximum and minimum rates of false positives for the non-Class A GPCR test set (t2).

Method	Training Set	MEP[1]					MaxRFP[1]	MedRFP[1]
		Errs[2]	TP	FP	TN	FN		
SAM	1a	139	63	2	208	137	1	0.224
SVM-Fisher	1a	122	110	32	178	90	0.995	0.138
SVM-pairwise	1a	17	197	14	196	3	0.133	0.005
SVM-linear	1a	68	186	54	156	14	0.748	0.143
SVM-polynomial	1a	55	179	34	176	21	0.652	0.052
SVM-sigmoid	1a	70	187	57	153	13	0.771	0.148
SVM-radial basis	1a	35	179	14	196	21	0.919	0.019

[1] Refer to Chapter 3 for the description of MEP, MaxRFP, and MedRFP.
[2] Errs = FP+FN.

As before, Table 4.3 shows only the results from the training data set 1a (random sampled data), since using the other training sets did not show any significant difference. The best method was SVM-pairwise with only 17 errors. The lowest MaxRFP and MedRFP were also found for SVM-pairwise. The second best method, SVM-rbf, had 35 errors,

Table 4.4: The minimum error point, maximum and minimum rates of false positives when the classification methods were trained on the dataset including bacteriorhodopsin sequences (1b).

Method	Training Set	MEP[1]					MaxRFP[1]	MedRFP[1]
		Errs[2]	TP	FP	TN	FN		
SAM	1a	139	63	2	208	137	1	0.224
SVM-Fisher	1b	122	110	32	178	90	0.995	0.138
SVM-pairwise	1b	16	198	14	196	2	0.129	0.005
SVM-linear	1b	65	186	51	159	14	0.662	0.110
SVM-polynomial	1b	53	174	27	183	26	0.643	0.048
SVM-sigmoid	1b	69	186	55	155	14	0.690	0.148
SVM-radial basis	1b	35	178	13	197	22	0.914	0.019

[1] Refer to Chapter 3 for the description of MEP, MaxRFP, and MedRFP.
[2] Errs = FP+FN.

twice as many as SVM-pairwise. Note that the higher error rate of SVM-rbf was solely due to its higher false negative rate. This means SVM-rbf missed more GPCR candidates than SVM-pairwise. This pattern is clearly shown in the very high MaxRFP rate (0.9) of SVM-rbf, whereas SVM-pairwise maintained a low MaxRFP rate (0.1). On the other hand, SVM-rbf did not misidentify non-GPCRs more than SVM-pairwise (indicated by the same number of false positives). The worst methods were SAM and SVM-Fisher with 139 and 122 errors, respectively. Their false negative rates were very high, and their MaxRFP rates were 1.0.

When bacteriorhodopsins were added to the training set, there was a decrease in the number of errors by only a small number as shown in Table 4.4).

Figure 4.5 shows the ROC curves compared among different methods. It shows that some methods are extremely bad while others are good. SAM and SVM-Fisher were the worst methods with the least area under their curves. Other SVMs did not do too bad. The best classifier to correctly identify GPCRs and non-GPCRs even from non-Class A

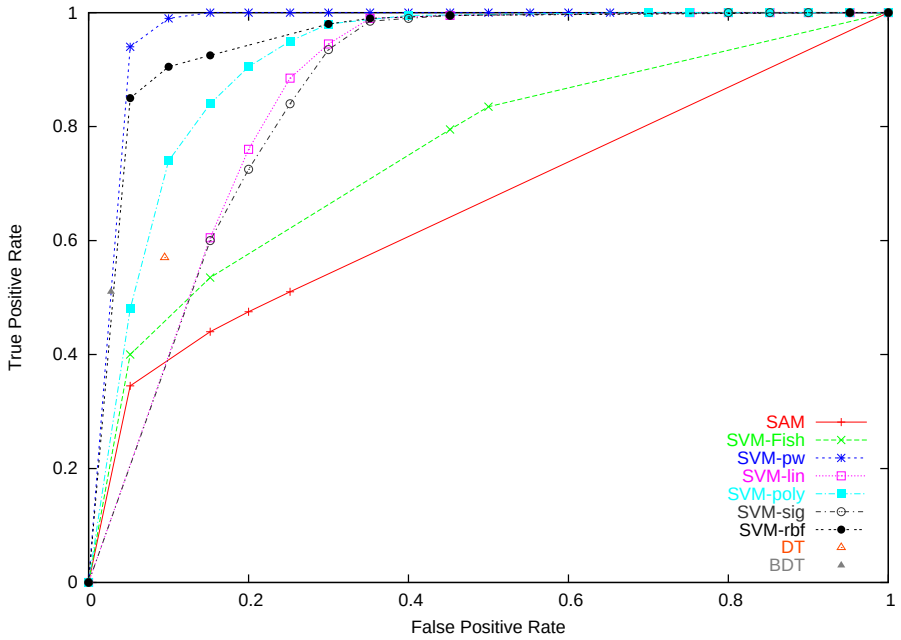

Figure 4.5: The ROC curves of the nine methods for the non-Class A GPCR test set (t2). The methods were trained on the training dataset 1a. Similar curves were obtained when other training sets were used.

GPCRs was SVM-pairwise. The second best method, SVM-rbf, after the true positive rate reached 0.85, showed a very slow increase in the true positive rate while the false positives accumulate quickly. This illustrates why SVM-rbf suffered from a very high MaxRFP rate as shown in Table 4.3.

4.3 Identification of Subsequences

Kim et al. [25] and Moriyama and Kim [31] had performed experiments to identify short subsequences of GPCRs with several different methods including their discriminant analysis methods. Their discriminant analysis methods based on amino acid properties outperformed HMM-based Pfam and other methods even when the sequences were 50 or 75 amino acids long. In order to examine how SVM-based methods perform for short subsequences, similar subsequence analysis was conducted in this study.

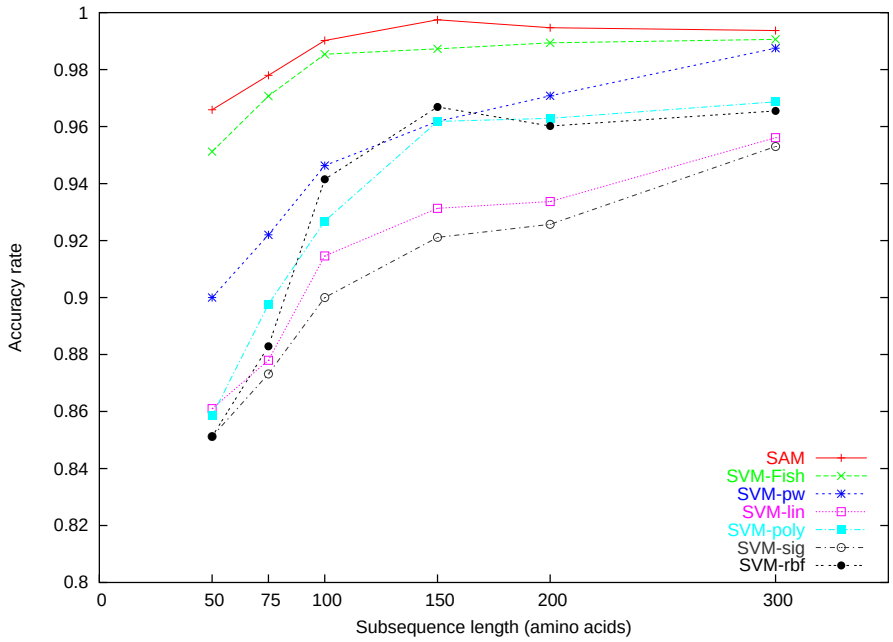

Figure 4.6: Accuracy rates of different methods tested on short subsequences.

The seven methods including SAM and six SVM-based methods trained on Class A GPCR datasets (1a-6a) were tested against the six subsequence test sets whose lengths range from 50 to 300 amino acids and prepared from the Class A GPCR test set (t1) as described in Chapter 3. Figure 4.6 summarizes the performance of the methods trained on the random sampling dataset (1a). No significant difference in performance was observed among methods trained on different training sets. The two best methods for identifying short subsequences were HMM-based SAM and SVM-Fisher. These two methods performed consistently better than the other methods. Even against very short 50 or 75 amino acid subsequences, they maintained higher than 95% accuracy. Other methods including SVM-pairwise dropped their accuracy to 85-90%. All of the methods examined recovered their performance quickly once the sequence length became 100 amino acids or longer.

The results in Figure 4.6 seem to be surprisingly good when compared to those reported

in Moriyama and Kim [31] (and Kim et al. [25]). Figure 4.7 is taken from Moriyama and Kim [31]. It shows higher performance of their discriminant function analysis methods (non-parametric LDA, LDA, and KNN) compared to an HMM (Pfam) and other methods. For 50 or 75 amino acid subsequences, discriminant analysis methods performed at 70-85% accuracy, whereas even HMM-based Pfam showed only 50-70% accuracy.

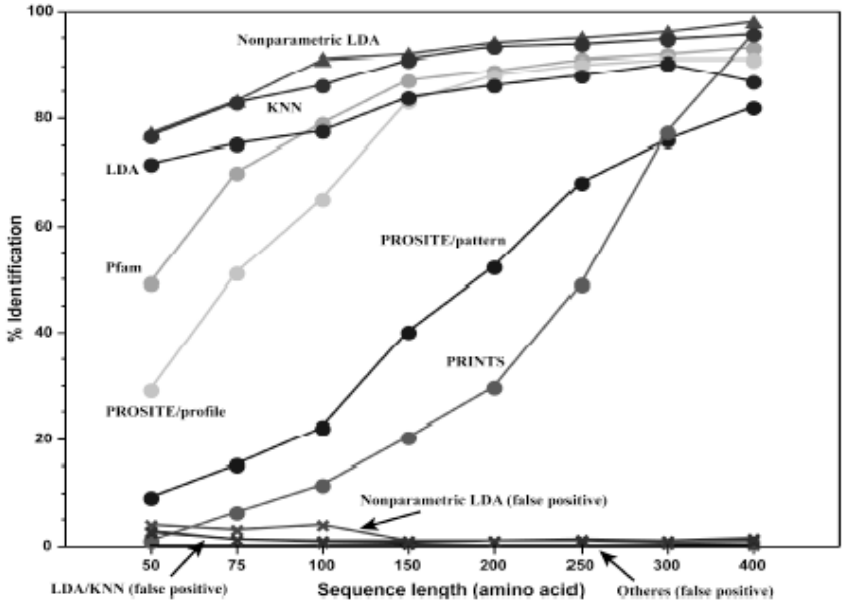

Figure 4.7: Identification rates of discriminant analysis methods compared with other methods (taken from Moriyama and Kim [31]). Their "% identification" is the same as the accuracy rate in this study.

The difference between these two studies can be explained by the way classifiers were trained and tested. In this study, only Class A GPCRs were used to train and test the classifiers. This made it easier to identify subsequences of Class A GPCRs for the classifiers because information specific to the Class A GPCR sequences were learned. It is also remarkable that in this study multiple alignment and HMM-based methods, SAM and SVM-Fisher, outperformed other methods even for the very short sequences. Class A GPCRs are relatively consistent in length and relatively easier to obtain better multiple alignments.

Such conservative sequence nature may have given an advantage for HMM-based methods to identify correctly even short subsequences.

On the other hand, Moriyama and Kim's [31] data sets (both training and test) were randomly sampled across the entire GPCR classes. For Pfam and other existing methods, no new training was done, since multiple alignments were impossible from such datasets. Instead, multiple HMMs, patterns, and fingerprints covering the entire GPCR classes were collected from HMM/motif databases and used against the test sets they prepared. Their results (Figure 4.7) illustrate the disadvantage these HMM/motif based methods face similar to what described in the previous section. Therefore, it is interesting to perform another subsequence test using non-Class A GPCR sequences. In such tests, the real advantage of using SVM-rbf over SAM, SVM-Fisher, or SVM-pairwise may be revealed.

Chapter 5

Conclusion and Future Work

From the experiments performed in this study, it can be concluded that SVM-pairwise is the best method in classifying Class A GPCRs as well as non-Class A GPCRs if full sequences are available. SVM-pairwise does not depend on an HMM nor multiple alignments. Both of the HMM-based methods, SAM and SVM-Fisher, worked extremely well in identifying Class A GPCRs, but performed poorly when trained on Class A GPCRs and tested for classifying non-Class A GPCRs.

Using amino acid frequencies with SVMs, especially with the radial basis kernel function, was very effective and an easy way to represent a protein sequence. This was evident from identifying Class A GPCRs with at least 91% accuracy and non-Class A GPCRs with at least 81% accuracy. Decision trees methods (boosted or not) did not perform as well as SVMs, although the boosted decision trees method was better than the regular (non-boosted) decision trees.

The addition of Bacteriorhodopsins in the negative training samples or the use of different sampling schemes of the positive training samples made very little difference in

the performance of the algorithms studied. Especially the classifiers that performed better (SVM-pairwise, SVM-Fisher, SAM, and SVM-rbf) did not show any effect of using different sampling schemes for training data. Overall, the simple random sampling for training data seemed to be good enough for preparing training data for the methods studied.

The two HMM-based methods, SAM and SVM-Fisher, worked surprisingly well for identifying short subsequences of Class A GPCRs. SVM-pairwise, on the other hand, did not perform well compared to the HMM-based methods. The SVMs based on amino acid composition could identify short subsequences as short as 50 amino acids at 85% or higher accuracy, although their performance was not as good as the alignment-based methods, SAM, SVM-Fisher, and SVM-pairwise.

It would be interesting to generate short subsequences form non-Class A GPCRs sequences (e.g., t2) and compare the performance of the nine methods used in this study as well as the discriminant analysis methods developed by Kim et al. [25] and Moriyama and Kim [31]. In such tests, the real advantage of using amino acid composition with SVMs should become evident. It is also interesting to see how these SVM methods perform compared to the discriminant analysis methods. Since they used some amino acid properties (hydrophobicity etc.) as well as amino acid composition as attributes, their methods may perform slightly better than the SVM-rbf used in this study. On the other hand, especially for short partial sequences, there may be no advantage for using other amino acid properties other than simple amino acid composition.

For comparison purposes, it would be interesting to try Lee's [27] approach with the kernel-based GMIL method with pairwise alignments and using amino acid properties, amino acid composition, and pairwise scores between sequences to build the GMIL classi-

fiers.

The SVM methods described in this thesis will be tested for identifying other types of proteins in the future. If amino acid composition is enough for other protein classification, and if other amino acid properties need to be included for more general protein classification, need to be examined further.

Bibliography

[1] Predix pharmaceuticals, Jan. 2000. http://www.predixpharm.com.

[2] Upstate cell signalling solutions, Mar. 2004. http://www.upstate.com/img/ features/gpcr.gif.

[3] T. Attwood, P. Bradley, D. Flower, A. Gaulton, N. Maudling, A. Mitchell, G. Moulton, A. Nordle, K. Paine, P. Taylor, A. Uddin, and C. Zygouri. PRINTS and its automatic supplement, prePRINTS. *Nucleic Acids Research*, 31(1):400–402, 2003.

[4] A. Bateman, L. Coin, R. Durbin, R. D. Finn, V. Hollich, S. Griffiths-Jones, A. Khanna, M. Marshall, S. Moxon, E. L. L. Sonnhammer, D. J. Studholme, C. Yeats, and S. R. Eddy. The Pfam protein families database. *Nucleic Acids Research*, 32(1):D138–D141, 2003. http://www.sanger.ac.uk/Software/Pfam/.

[5] B. Boeckmann, A. Bairoch, R. Apweiler, M.-C. Blatter, A. Estreicher, E. Gasteiger, M. Martin, K. Michoud, C. O'Donovan, I. Phan, S. Pilbout, and M. Schneider. The SWISS-PROT protein knowledgebase and its supplement TrEMBL in 2003. *Nucleic Acids Research*, 31:365–370, 2003. http://us.expasy.org/sprot/.

[6] D. R. Burgard, J. T. Kuznicki, and L. M. Peterson. *Chemometrics: Chemical and Sensory Data*. CRC Press, 1990.

[7] N. Christianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines*. Cambridge University Press, 1st edition, 2000.

[8] G. M. Cooper. *The Cell: A Molecular Approach*. ASM Press, 2nd edition, 2000.

[9] R. Durbin, S. Edy, A. Krogh, and G. Mitchison. *Biological sequence analysis: Probabilistic models of proteins and nucleic acids*. Cambridge University Press, 1998.

[10] T. Fawcett. ROC Graphs: Notes and Practical Considerations for Data Mining Researchers, 2003. Copyright Hewlett-Packard Company 2003.

[11] J. Felsenstein. PHYLIP (Phylogeny Inference Package) Version 3.6, 1993. "http://evolution.genetics.washington.edu/phylip.html Distributed by the author. Department of Genetics, University of Washington, Seattle.

[12] J. C. Foreman and T. Johansen. *Textbook of receptor pharmacology*. CRC Press, 2nd edition, 2003.

[13] R. H. Garret and C. M. Grisham. *Biochemistry*. Saunders College Publishing, 2nd edition, 1998.

[14] W. N. Grundy. Homology Detection via Family Pairwise Search. *Journal of Computational Biology*, 5(3):479–492, 1998.

[15] F. Horn, J. Weare, M. W. Beukers, S. Hörsch, A. Bairoch, W. Chen, Ø. Edvardsen, F. Campagne, and G. Vriend. GPCRDB: An Information system for G protein-coupled receptors. *Nucleic Acids Research*, 26(1):275–279, 1998. http://www.gpcr.org/7tm/.

[16] R. Hughey and A. Krogh. Hidden Markov models for sequence analysis: Extension and analysis of the basic method. *CABIOS*, 12(1):95–107, 1996.

[17] N. Hulo, C. Sigrist, V. L. Saux, P. Langendijk-Genevaux, L. Bordoli, A. Gattiker, E. D. Castro, P. Bucher, and A. Bairoch. Recent improvements to the PROSITE database . *Nucleic Acids Research*, 32:134–137, 2004. http://au.expasy.org/prosite/.

[18] T. Jaakkola, M. Diekhans, and D. Haussler. A discriminative framework for detecting remote protein homologies. *Journal of Computational Biology*, 7(1-2):95–114, 2000.

[19] T. S. Jaakkola and D. Haussler. Exploiting generative models in discriminative classifiers. *Unpublished*, 1998.

[20] T. Joachims. *Advances in Kernel Methods - Support Vector Learning* , chapter 11: Making large-Scale SVM Learning Practical. MIT-Press, 1999. http://svmlight.joachims.org/.

[21] D. Jones, W. Taylor, and J. Thornton. Amino acid substitution matrices from protein blocks. *CABIOS*, 8:275–282, 1992.

[22] R. Karchin. Classifying G-protein Coupled Receptors with Support Vector Machines. Master's thesis, University of California, Santa Cruz, June 2000.

[23] R. Karchin, K. Karplus, and D. Haussler. Classifying G-protein coupled receptors with support vector machines. *Bioinformatics*, 18(1):147–159, 2002.

[24] K. Karplus, R. Karchin, C. Barrett, S. Tu, M. Cline, M. Diekhans, L. Grate, J. Casper, and R. Hughey. What Is the Value Added by Human Intervention in Protein Structure Prediction? *PROTEINS*, 5(1):86–91, 2001.

[25] J. Kim, E. N. Moriyama, C. G. Warr, P. J. Clyne, and J. R. Carlson. Identification of novel multi-transmembrane proteins from genomic databases using quasi-periodic structural properties. *Bioinformatics*, 16(9):767–775, 2000.

[26] A. Krogh, M. Brown, I. S. Mian, K. Sjölander, and D. Haussler. Hidden Markov models in computational biology: Applications to Protein Modeling. *Journal of Molecular Biology*, 235(5):1501–31, 1994.

[27] S.-M. J. Lee. Protein Function Classification with Generalized Multiple Instance Learning. Master's thesis, University of Nebraska, Lincoln, August 2004.

[28] W.-H. Li. *Molecular Evolution*. Sinauer Associates, Inc., Publishers, 1997.

[29] L. Liao and W. S. Noble. Combining pairwise sequence similarity and support vector machines for remote protein homology detection. *Unpublished*, 2002.

[30] L. Liao and W. S. Noble. Combining Pairwise Sequence Similarity and Support Vector Machines for Detecting Remote Protein Evolutionary and Structural Relationships. *Journal of computational biology*, 10(6):857–868, 2003.

[31] E. N. Moriyama and J. Kim. Protein family classification using discriminant analysis. In J. Gustafson, editor, *Data Mining the Genomes: 23rd Stadler Genetics Symposium*, New York, In press. Kluwer Academic/Plenum Publishers.

[32] A. G. Murzin, S. E. Brenner, T. Hubbard, and C. Chothia. SCOP: a structural classification of proteins database for the investigation of sequences and structures. *Journal of Molecular Biology*, 247:536–540, 1995. http://scop.berkeley.edu/.

[33] W. R. Pearson. Searching protein sequence libraries: comparison of the sensitivity and selectivity of the Smith-Waterman and FASTA algorithms. *Genomics*, 11:635–650, 1991.

[34] J. Quinlan. Bagging, boosting, and c4.5.

[35] J. R. Quinlan. *C4.5: Programs for machine learning*. Morgan Kauffman Publishers, 1993.

[36] N. Saitou and M. Nei. The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution*, 4:406–425, 1987.

[37] R. Schapire. Theoritical views of boosting and applications. *Tenth International Conference on Algorithmic Learning Theory*, pages 13–25, 1999.

[38] K. Sjolander, K. Karplus, M. Brown, R. Hughey, A. Krogh, I. Mian, and D. Haussler. Dirichlet Mixtures: A Method for Improving Detection of Weak but Significant Protein Sequence Homology. *CABIOS*, 12(4):327–345, 1996.

[39] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195–197, 1981.

[40] Sneath and Snokal. Numerical Taxonomy, 1973.

[41] J. D. Thompson, D. G. Higgins, and T. J. Gibson. CLUSTALW: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Research*, 22(22):4673–4680, 1994.

[42] C. Wang, S. D. Scott, J. Zhang, Q. Tao, D. E. Fomenko, and V. N. Gladyshev. A study in modeling low-conservation protein superfamilies, 2004. Technical Report TR-UNL-CSE-2004-0003.

[43] S. Watson and S. Arkinstall. *The G-Protein linked receptor FactsBook*. Academic Press, 1994.

[44] I. H. Witten and E. Frank. *Data Mining: Practical machine learning tools with Java implementations*. Morgan Kaufmann, San Francisco, 2000.

[45] J. Zhang. A fast algorithm for GMIL and its applications to protein superfamily identification. Master's thesis, University of Nebraska, Lincoln, December 2003.

Appendix A

Accuracy Rate Tables

Table A.1: Accuracy rates of the nine methods for classifying the Class A GPCRs.

Training set[1]	Classification methods[2]								
	SAM	SVM-lin	SVM-poly	SVM-sig	SVM-rbf	SVM-pw	SVM-Fish	DT	BDT
1a	0.998	0.939	0.963	0.924	0.982	0.998	0.997	0.922	0.965
2a	0.998	0.922	0.965	0.919	0.987	0.998	0.995	0.943	0.963
3a	0.998	0.919	0.958	0.914	0.980	0.998	0.992	0.919	0.951
4a	0.998	0.919	0.956	0.917	0.982	0.998	0.997	0.922	0.943
5a	0.998	0.922	0.961	0.914	0.985	0.998	0.990	0.924	0.953
6a	0.998	0.919	0.963	0.917	0.982	0.998	0.995	0.941	0.958
1b	0.998	0.953	0.965	0.939	0.982	0.998	0.997	0.922	0.961
2b	0.998	0.934	0.973	0.924	0.990	0.998	0.995	0.943	0.958
3b	0.998	0.929	0.963	0.919	0.980	0.998	0.992	0.943	0.951
4b	0.998	0.931	0.963	0.919	0.982	0.998	0.997	0.922	0.948
5b	0.998	0.931	0.963	0.924	0.985	0.998	0.992	0.924	0.951
6b	0.998	0.936	0.970	0.924	0.985	0.998	0.995	0.939	0.970

[1] For each training set, refer to Table 3.2.

[2] The method name abbreviations are as follows. SVM-lin: SVM with linear kernel function, SVM-poly: SVM with polynomial kernel function, SVM-sig: SVM with sigmoid kernel function, SVM-rbf: SVM with radial basis kernel function, SVM-pw: SVM-pariwise, SVM-Fish: SVM-Fisher, DT: Decision trees, and BDT: Boosted decision trees.

Table A.2: Accuracy rates of cross-validation test for classifying the Class A GPCRs.[1]

Training set	Classification methods					
	SVM-lin	SVM-poly	SVM-sig	SVM-rbf	SVM-pw	SVM-Fish
1a	0.905	0.937	0.897	0.970	0.990	0.992
2a	0.895	0.935	0.895	0.962	0.989	0.994
3a	0.877	0.910	0.885	0.927	0.990	0.990
4a	0.902	0.907	0.892	0.947	0.990	0.980
5a	0.897	0.910	0.897	0.957	0.990	0.990
6a	0.902	0.917	0.895	0.957	0.992	0.992
1b	0.914	0.936	0.890	0.970	0.990	0.992
2b	0.890	0.937	0.877	0.963	0.989	0.994
3b	0.902	0.907	0.882	0.934	0.990	0.990
4b	0.907	0.909	0.897	0.946	0.990	0.980
5b	0.914	0.914	0.892	0.958	0.990	0.990
6b	0.904	0.922	0.890	0.961	0.992	0.992

[1] See the footnotes of Table A.1

Table A.3: Accuracy rates of nine methods for classifying the non-Class A GPCRs.[1]

Training	Classification methods								
set	SAM	SVM-lin	SVM-poly	SVM-sig	SVM-rbf	SVM-pw	SVM-Fish	DT	BDT
1a	0.661	0.834	0.865	0.829	0.914	0.958	0.702	0.741	0.746
2a	0.629	0.843	0.865	0.836	0.890	0.963	0.700	0.722	0.739
3a	0.726	0.829	0.861	0.822	0.900	0.956	0.724	0.700	0.726
4a	0.751	0.822	0.853	0.817	0.892	0.956	0.697	0.704	0.690
5a	0.739	0.831	0.851	0.829	0.890	0.963	0.741	0.651	0.719
6a	0.758	0.829	0.853	0.829	0.904	0.958	0.758	0.751	0.775
1b	0.661	0.841	0.870	0.831	0.914	0.961	0.702	0.741	0.756
2b	0.629	0.841	0.878	0.829	0.892	0.963	0.702	0.722	0.734
3b	0.726	0.829	0.868	0.819	0.900	0.956	0.724	0.734	0.726
4b	0.751	0.824	0.865	0.817	0.897	0.956	0.695	0.704	0.729
5b	0.739	0.824	0.865	0.817	0.892	0.963	0.741	0.651	0.714
6b	0.758	0.822	0.865	0.812	0.904	0.958	0.756	0.729	0.802

[1] See the footnotes of Table A.1