Classifying proteins using gapped markov feature pairs

Xiaonan Ji[a], James Bailey[*,a], Kotagiri Ramamohanarao[a]

[a]*NICTA Victoria Laboratory, Department of Computer Science and Software Engineering, The University of Melbourne, Australia*

Abstract

Classifying protein sequences has important applications in areas such as disease diagnosis, treatment development and drug design. In this paper we present a highly accurate classifier called the g-MARS (gapped Markov Chain with Support Vector Machine) protein classifier. It models the structure of a protein sequence by measuring the transition probabilities between pairs of amino acids. This results in a Markov chain style model for each protein sequence. Then, to capture the similarity among non-exactly matching protein sequences, we show that this model can be generalized to incorporate gaps in the Markov chain. Theoretical justification for the power of our gapped feature space model is provided through its connections to analysis methods for nonlinear dynamical systems. We perform an experimental study and compare g-MARS to several other state-of-the-art protein classifiers. Overall, we demonstrate that g-MARS has high accuracy and operates efficiently on a diverse range of protein families.

Key words: Protein classification, Gapped markov chains, Support vector machine, Delay space

1. Introduction

With the development of genome sequencing techniques, biologists have accumulated huge numbers of protein sequences and new ones are being discovered daily. Predicting the class or the main function of a new protein sequence can assist experts in understanding its nature. It is a difficult problem, however, and it is not easy to advance the state of the art. Successful protein classifiers must be able to compare sequences efficiently, detect important features and also show good predictive capability.

A number of algorithms have been developed for classifying proteins into families or into clusters of functions or localizations. The basic assumption mostly used is the first fact of biological sequence analysis: "In biomolecular

[*]Corresponding author.
Email addresses: xji@csse.unimelb.edu.au (Xiaonan Ji), jbailey@csse.unimelb.edu.au (James Bailey), rao@csse.unimelb.edu.au (Kotagiri Ramamohanarao)

sequences (DNA, RNA or amino acid sequences), high sequence similarity usually implies significant functional or structural similarity."[11]. So, to create highly-accurate classifiers, we need a way to compare the similarity of a large number of diverse sequences precisely and efficiently.

Our contribution. In this paper, we describe a new protein classifier called the g-MARS (gapped Markov Chain with Support Vector Machine) classifier. The g-MARS approach has two main stages. Firstly, each protein sequence is individually modeled using what we call a "gapped markov chain", to capture its statistically important features. Next, a new dataset is derived from the collection of all gapped markov chains and it is passed to a support vector machine for decision making. The advantages of g-MARS are its simplicity and good accuracy compared to several existing protein classification methods. This is a claim validated in our experimental study, which considers a diverse range of protein families with different characteristics. The technique also scales well for large datasets. We first begin with a review of related work in the area.

2. Related work

Previous work on protein classification mainly falls into six categories: Amino acid composition-based algorithms, support vector machine-based kernel algorithms, markov model-based algorithms, sequence alignment-based algorithms, markov chain-based algorithms and support vector machine-based hybrid algorithms.

Amino acid composition-based algorithms measure the similarity of proteins from the compositions of their amino acids. For each protein in the training dataset, the algorithm[10] calculates the frequency of each of its amino acids. For a new protein to be classified, its amino acid frequency histogram is calculated and compared with the compositions of the proteins in each class of training data. The protein is then classified to the class containing the protein with the smallest composition difference. The shortcomings of this approach are the loss of the ordering relationship among amino acids and the simplistic comparison in the composition difference. These compositions may be biased for small training datasets.

Amino acid composition with gaps[12] is an improvement of the pure amino acid composition algorithm[10]. The first improvement is that it considers pairs of the amino acids rather than individual ones. The second improvement is that it uses a support vector machine to make decisions. This is useful for situations where the number of features relatively large in comparison to the number of instances. Many other studies[12, 13, 17, 18, 22, 28] also show that support vector machines, if used properly, perform well in protein classification problems. The limitation is that the measurement is still based on the percentages of the pairs of amino acids among the whole protein sequence. When two proteins have different lengths, although they share some similar sections, certain amino acid pairs may have composition differences.

Support vector machine-based kernel algorithms concentrate on developing new kernels that are capable of handling sequence similarity measurement directly. The spectrum kernel [18] is a support vector machine algorithm that calculates the similarity of two sequences by their common k-mers. A user-defined parameter k is provided and the number of occurrences of each k-mer (contiguous amino acids with fixed length of k) is calculated. Next, the support vector machine is used to compute the support vectors using the k-spectrum kernel. A testing sequence is mapped to the k-spectrum in the same way and is compared with the support vectors to decide which class it belongs to. The underlying principle is, the more common k-mers being shared between two proteins, the more structurally similar they are. In practice, the spectrum kernel works quite well [18]. However, there are limitations: it is far more computationally expensive than the amino acid composition algorithm. Secondly the choice of k in practice must be small, since the number of k-mers increase exponentially with k (so $k = 3$ is generally used). Thirdly, since k-mers must be contiguous, there can be less tolerance when proteins contain errors or mutations. In the mismatch kernel[17], the sharing of the similar k-mers, along with the identical ones, is used to measure the similarity.

Previous work by Wang et al [26] presents an interesting, but very general framework called GMM (Generalizations of Markov Model), for using markov models to classify proteins using amino acid feature combinations which may include gaps. It uses generalizations of Markov models to characterize biological sequences. It computes the log-odds scores (i.e. the natural logarithm of the ratio of the probability that an event happens to the probability that the event does not happen) of the transition probabilities from the prior (the amino acids upon which the probability is conditioned) to the posterior (the amino acids whose probability one wishes to compute). In GMM, the prior is O number of L_1-grams, each separated by a gap g_1 from one other. The posterior is L_2 amino acids, each separated by a gap g_2 from one other. There can also be a third gap G, which is the distance between the prior and the posterior. If there are O number of L_1-grams in the prior, then the prior-posterior pair is called an O-th order pair. Note that the standard n-order Markov model captures the transition probability from an n-gram to the next single item, which is equivalent to the prior-posterior pair used in GMM when $L_1 = 1, O = n, g_1 = 0, G = 0, g_2 = 0, L_2 = 1$. In the training procedure, GMM scans the training data set once to compute the log-odds scores of the transition probabilities for all prior-posterior pairs, for which the prior is of sufficiently high frequency. It also retains higher order pairs in preference to lower order pairs. In the testing procedure, GMM moves a sliding window along the testing instance. Given a class, for each window position the log-odds score of the best matching prior-posterior pair is added to the score for that class. GMM predicts the testing instance to belong to the class with the highest score.

Our g-MARS algorithm can roughly fit into this framework, but with a number of key differences: i) GMM requires the configuration of between six and ten different parameters and does not provide any general strategy for choosing them, a difficult challenge for a user . Thus it is better described as a large space of possible algorithms, rather than a single algorithm (and so it is very difficult to try to experimentally benchmark against). For g-MARS, there is a single parameter g (the gap) that needs to be chosen to construct the feature space. This feature space is then passed to a support vector machine for classification. Choices of kernel and kernel parameters are needed for using the support vector machine. ii) Different combinations of features are used. Only the prior and posterior pair with the highest order is used for classifying a protein by GMM. In g-MARS, however, we consider variable gaps and use all resulting prior-posterior pairs for the classification decision, iii) The GMM classification/decision model is essentially a set of prior-posterior pairs which work as rules and classification relies on aggregating scores of these rules. In contrast, g-MARS learns a classification model based on training a support vector machine. Also, g-MARS and GMM use somewhat different representations. g-MARS constructs a vector of feature values for each protein, describing its markov characteristics. GMM, though, constructs a vector of feature values *for each entire class of proteins*, aggregating the statistics of each feature across *all* proteins in the class.

Among the classifiers built on sequence alignment, Hidden Markov Models (HMM) are widely used. In its most naive form, a single hidden markov model is built to summarise a set of sequences for a given data class. The model has five types of hidden states and their total number can be varied. State transition and emission (observation symbol) probabilities are calculated from the training sequences by using the Baum-Welch algorithm[5]. Given a testing sequence, the probability for this sequence to be generated by an HMM can be calculated by using the forward-backward procedure[5]. Markov chains can be treated as a special case of hidden markov models and there are methods for directly applying markov chains to classify proteins. One can calculate the log-odds ratio between these markov chains and use this to classify unknown proteins. The difficulty is that there is usually not sufficient training data to obtain good maximum likelihood estimators for the transition probabilities of the markov chain. This can seriously impair classification accuracy.

One difficulty of using Support vector machines (SVMs) for protein classification is that classic kernels cannot handle sequence data. Instead, an extra step is needed to "transform" each sequence into a fixed-dimensional vector, so that these kernels can then be applied. Such methods come under the category of support vector machine-based hybrid algorithms.

The Fisher kernel[13] combines the support vector machine and the hidden markov model. It first builds a hidden markov model using the set of training data. Given a testing sequence, a sufficient statistic representation of it is computed from the model. The representation is a fixed-length vector with a value for each parameter in the model as the posterior frequencies of having taken a particular transition or having generated one of the residues of the testing sequence from a particular state. Next, the "distances" between pairs of the vectors of the sufficient statistics are computed using a specific kernel function. Finally, the "distance" between the testing sequence's vector of sufficient statistics and the vector of sufficient statistics of each training sequence from each class is summed up. The testing sequence is classified to the class with the smallest summation. Our g-MARS approach is different from the Fisher kernel. Firstly, we do not use a hidden markov model generated from the whole training dataset. Instead, we use the gapped markov chain generated from each individual training protein. Secondly, the "distance" between two proteins in the SVM is not calculated directly by the kernel function[13]. It is instead calculated by a classic relational kernel such as the RBF kernel. We note that there also exist several other sequence classifiers that transform sequences into attribute-value pairs and build support vector machines for classification, e.g. Oligomer distance kernels [20], Pairwise kernels [19], profile kernels [16] and cluster kernels [27].

Work[22, 28] has been done on building a series of classifiers which make use of frequent substring patterns and the support vector machine. The algorithms firstly mine the frequent substrings from the training proteins that are frequent and discriminative for their own class (each pattern is mined with high confidence). Then they reform each sequence (training and testing sequences) by verifying which patterns are contained in it. An SVM is used for decision making on the reformatted dataset. In terms of the formatting, each sequence is translated into a vector of 0s and 1s with fixed length (the dimension of each such vector is the total number of frequent substrings found). If a pattern is found to be contained in the sequence, the corresponding dimension has a value 1. For other patterns that are not contained, the dimensions have a value 0. The decision is made by the support vector machine built upon these vectors. The shortcomings of these algorithms are: 1.The dimension of the vectors for the protein sequences depends on the number of frequent substrings found, which is difficult to control. In order to keep the dimension within a reasonable range, some pruning strategies are needed. 2.There are several parameters such as the frequency threshold, the confidence threshold and the minimum and maximum lengths of the patterns, as well as the desirable number of patterns, that need to be settled in order to obtain good performance. There is no general strategy for choosing them. 3.The pattern mining procedure can require considerable computational resources (time and space).

Finally, we note that a preliminary conference version of this current paper appeared in [14].

3. Preliminaries.

A sequence $p = a_1a_2a_3...a_n$ is a length n sequence. Each character a_k in p is chosen from an alphabet set $\mathbb{A}$ and referred to as $p(k)$. Throughout this paper, we consider protein primary structure (amino acid sequences), but our technique is easily adapted to classification of other types of sequences as well.

In protein classification problems, a training dataset *TrDB* contains proteins whose classes are known to the classifier. The class label for each protein p is denoted as *p.c*. A testing dataset *TeDB* contains proteins whose classes are unknown to the classifier. The task is to predict the class label of each unknown protein sequence according to the training dataset. The predicted class label for each such protein p is denoted as *p.pc*. Given a testing protein p, if the predicted class label is the same as its real class label, that is, $p.pc = p.c$, we say it is correctly classified by the classifier, otherwise it is misclassified.

If the dataset only contains proteins from two classes, it is a binary-class classification problem. For the multi-class classification problem, where the testing dataset contains proteins belonging to more than two classes, we choose proteins from one class and merge the rest of the proteins into another class. In this way the multi-class classification problem can be reduced to a binary-class classification problem. The task is then to predict whether a testing protein belongs to the chosen class or not. The chosen class is called the positive class (or the target class) and can be denoted as T. The merged set of instances (named the negative class) containing all other proteins is denoted as $\neg T$. $TrDB_T = \{p \in TrDB \mid p.c = T\}$ is called the training positive set and $TrDB_{\neg T} = \{p \in TrDB \mid p.c \neq T\}$ is called the training negative set. Corresponding definitions exist for sets of testing instances $TeDB_T$ and $TeDB_{\neg T}$.

4. g-MARS Methodology

Training the g-MARS classifier has two main phases. Firstly, g-MARS builds for each $p \in TrDB$, a gapped markov chain. Secondly, g-MARS passes the vectorial expressions of the gapped markov chains to a support vector machine (SVM) for decision making.

Markov chains are a well known method for modeling sequences. The system consists of a set of states, where each is labelled by a character $a \in \mathbb{A}$ and a set of transitions which are associated with some probabilities. From one position to the next one of the sequence, the system undergoes a change of state (possibly a self-loop to the same state), according to the transition probability between the states. An important special case is the first order markov chain, where the transition probability depends only on the current and the predecessor position, i.e., $Pr[p(i) = a_k \mid p(i-1) = a_j, p(i-2) = a_m, ...] = Pr[p(i) = a_k \mid p(i-1) = a_j]$.

Furthermore, the markov chains we will consider are independent of the sequence positions. In other words, the probabilities of a transition from item a_m to a_n do not depend on the position in the sequence where transition occurs.

A markov chain modeling a sequence p consists of two kinds of components. One is the set of the states $\{S_i\}$ representing each character from $\mathbb{A}$ and the other is the set of transition probabilities $\{t_{ij}\}$ between states. The formal definition of transition probability t_{ij} leading from state S_i to S_j is: $t_{ij} = Pr[p(k) = a_j \mid p(k-1) = a_i]$

In order to build a markov chain of the sequence p, we have to decide the probability of each pair of the states. A maximum likelihood estimation procedure is applied to calculate these probabilities: $t_{ij} = \frac{c_{ij}}{\sum_k c_{ik}}$, where c_{ij} is the number of times amino acid j follows amino acid i in p and $\sum_k c_{ik}$ is the number of times the amino acid i is followed by any amino acid.

Example 1. *Consider the sequence $p = ABACCAB$. The markov chain for p has three states and we have $t_{AA} = 0$, $t_{AB} = \frac{2}{3}$, $t_{AC} = \frac{1}{3}$, $t_{BA} = 1$, $t_{BB} = 0$, $t_{BC} = 0$, $t_{CA} = \frac{1}{2}$, $t_{CB} = 0$ and $t_{CC} = \frac{1}{2}$.*

Example 2. *Given a markov chain m and a sequence s, it is straightforward to compute the probability of the m to produce s. e.g. If $s = ABABC$, the probability for m to produce s is $p(s|m) = t_{AB} \times t_{BA} \times t_{AB} \times t_{BC}$.*

The purpose of building the markov chain for each protein is that similar global or local structures of two proteins can be captured by their markov chains. E.g., the probability for amino acid X followed by amino acid Y can be discriminative for proteins from two different classes. This is true if the proteins from the same class share a lot of common sections and those common sections are different between different classes. One issue is that it is rare for many proteins from the same class to share long common sections. The common parts may be similar, but not exactly the same. An example to further illustrate is:

Example 3. *Consider two sequences $p_1 = ABC$ and $p_2 = ADC$. The first order markov chains of them are quite different. For p_1, the non-zero probability transitions are $t_{AB} = 1$ and $t_{BC} = 1$. For p_2, the non-zero probability transitions are $t_{AD} = 1$ and $t_{DC} = 1$. There is no common non-zero transition probability between the markov chains of p_1 and p_2. However p_1 and p_2 share two out of three characters, which may indicate some similarity.*

4.1. Introduction to Gapped Markov Chains

To overcome the limitation of traditional markov chains which only model successive state transitions, we modify the traditional markov chain in two ways. The first is to model the ending of the sequence and the second is to add the concept of gaps.

Modelling the ending of the sequence. In Example 1, the transition probability t_{BA} is 1, meaning that in sequence p, if B is followed by any amino acid, it must be A. This does not consider the last character $p(7)$, which has no character following. A more complete model should illustrate that in p, the probability for B to be followed by A is 0.5 and the probability for B to be followed by nothing is 0.5. This can be reflected by changing the transition probability definition to $t_{ij} = \frac{c_{ij}}{c_i}$, where c_{ij} is the number of times amino acid j follows amino acid i in p and c_i is the number of times the amino acid i appears in p.

Although we consider the ending of the sequence in the probability calculation, when we translate our markov chain into a feature representation (described shortly), we do not use the null (end of sequence) state and state transitions from other states to the null state (null transitions). There are two reasons: Firstly, when the transitions from one state to another non-null state are determined, its null transitions are also implicitly determined. Including the null transition is redundant. Secondly, by removing the null transitions, we reduce the model size, which has benefits for the classification process used later. In practice, the exclusion of these transitions does not impair classification accuracy.

The concept of gaps. In a g-gapped markov chain, we determine the probabilities of amino acid transitions, where there may be gaps between the amino acid pairs being considered. In particular, we allow contiguous (with no gap), jumping of one amino acid (with the gap as 1), jumping of two amino acids (with the gap as 2) and so on up to the g-th gap. The state transition probabilities are redefined as $t_{ij}^{k} = \frac{c_{ij}^{k}}{c_i}, \ \ 0 \leq k \leq g$, where t_{ij}^{k} is the probability of a transition from amino acid i to amino acid j with gap as k in p; c_{ij}^{k} is the number of times amino acid i has gap k to amino to amino acid j in p. c_i is the number of times amino acid i appears in p.

Suppose we allowed a character $\emptyset$ called "The-Character-Don't-Care". Our gapped markov chain can be used to directly model sequences containing $\emptyset$. An example is given in Example 4.

Example 4. *Given a sequence $p = AB\emptyset BC$, the probability for it to be produced by a gapped markov chain can be calculated as $Pr(p) = t_{AB}^{0} * t_{BB}^{1} * t_{BC}^{0}$. The probability of p can be directly reflected by the gapped markov chain. Note that the probability of p could also be calculated by the traditional markov chain indirectly: $Pr(p) = t_{AB} * (\sum_i (t_{Bi} * t_{iB})) * t_{BC}$.*

The purpose of being able to model sequences containing $\emptyset$ is to capture the approximate similarity between protein sequences.

Example 5. *Consider two sequences $p_1 = ABC$ and $p_2 = ADC$. Comparing contiguous amino acid pairs gives no similarity between their transition probabilities (c.f. Example 3). If we ignore their second characters, the sequences become $p_1' = A\emptyset C$ and $p_2' = A\emptyset C$, which are the same. This commonality is reflected when we compare p_1 and p_2 allowing gaps in the markov chain: for gap equal to 1, we have the non-zero transition probabilities of p_1 as $t_{AB}^{0} = 1$, $t_{BC}^{0} = 1$ and $t_{AC}^{1} = 1$. The non-zero transition probabilities of p_2 are $t_{AD}^{0} = 1$, $t_{DC}^{0} = 1$ and $t_{AC}^{1} = 1$. We can see p_1 and p_2 now share one common transition probability.*

Given that we can generate a g-gapped markov chain for a sequence, how do we compare two markov chains to obtain the similarity between two sequences? A direct way would be, for each pair of states, compare their transition probabilities and count the number which are identical to get a score of the similarity of the two sequences. E.g., considering $p_1 = ABC$ and $p_2 = ADC$ from the previous example, the number of transitions having the same non-zero probability under a 0-gapped markov chain model is 0, so the similarity of p_1 and p_2 under gap 0 would be 0. The similarity score for a 1-gapped markov chain model would be 1, because they share exactly one common transition, namely t_{AC}^{1}.

In order to discriminate between classes of proteins, we will take the approach of translating markov chains into vectors and then using a support vector machine to derive a decision boundary between the classes. We now provide some brief background on support vector machines.

Support Vector Machines: The Support vector machine (SVM) [25] is a classification technique that has its roots from statistical learning theory.

SVMs require all data to be presented in vectorial format in some feature space. Suppose each instance is denoted by a tuple $<\mathbf{x_i}, y_i>$, where $\mathbf{x_i} = (x_{i1}, x_{i2}, \cdots, x_{id})^T$ corresponds to the d-dimension vector of the i-th instance and $y_i \in \{-1, 1\}$ denotes the class label of the instance. A hyperplane of a linear classifier can be written as:

$$\mathbf{w} \cdot \mathbf{x} + b = 0, \tag{1}$$

where $\mathbf{w}$ and b are parameters determining the hyperplane and $\mathbf{x}$ is the data points locating on the hyperplane. If the data point $\mathbf{x}$ is on either side of the hyperplane, Equation 1 should be changed to either:

$$\mathbf{w} \cdot \mathbf{x} + b > 0, \tag{2}$$

or

$$\mathbf{w} \cdot \mathbf{x} + b < 0. \tag{3}$$

From the classification point of view, a good hyperplane should be able to separate data points from different classes. SVMs try to find out the single best one that maximizes the margins between different classes. This leads to the following constrained optimization problem:

$$\arg\min_{\mathbf{w}}(\tfrac{\|\mathbf{w}\|^2}{2}),$$

$$\text{subject to } y_i(\mathbf{w} \cdot \mathbf{x_i} + b) \geq 1, \text{ for all } <\mathbf{x_i}, y_i>.$$

The constraints can be integrated and the new objective function to minimize becomes the following Lagrangian:

$$L_P = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{N} \lambda_i [y_i(\mathbf{w} \cdot \mathbf{x_i} + b) - 1].$$

During the training procedure, SVMs try to find out the parameters λ_i, $\mathbf{w}$ and b to identify the best hyperplane that can maximally correctly separate training data points from different classes with largest margins.

In the testing procedure, a testing instance $\mathbf{z}$ is classified as follows:

$$f(\mathbf{z}) = sign(\mathbf{w} \cdot \mathbf{z} + b),$$

where $f(\mathbf{z})$ indicates the locating side of the data point $\mathbf{z}$ according to the hyperplane.

For more complex data point distributions where no linear hyperplane exists to separate the training data points, the data can be mapped to another space and a linear separating hyperplane may exist in that new space. A kernel function allows one to compute similarity of two vectors in the new space, by using their original attribute-value pairs. This is usually called the kernel trick.

Some of the popular kernel functions are the linear kernel and Gaussian (RBF) kernel.

Support vector machines using classic kernel functions require the input format to be vectors. We must therefore be able to represent gapped markov chains as vectors. This is straightforward: simply form a vector where each dimension corresponds to a transition and the value for that dimension is the probability of the transition. Transitions are annotated with gaps, so t^0_{AA} is considered to be a different dimension to t^1_{AA}. The ordering of the transitions does not matter as long as it is consistent for all the sequences in *TrDB* as well as *TeDB*. An example is:

Example 6. *Consider the sequence $p = ABACCAB$. The vector for this 1-gapped markov chain is ($t^0_{AA} = 0$, $t^0_{AB} = 0.66$, $t^0_{AC} = 0.33$, $t^1_{AA} = 0.33$, $t^1_{AB} = 0$, $t^1_{AC} = 0.33$, $t^0_{BA} = 0.5$, $t^0_{BB} = 0$, $t^0_{BC} = 0$, $t^1_{BA} = 0$, $t^1_{BB} = 0$, $t^1_{BC} = 0.33$, $t^0_{CA} = 0.5$, $t^0_{CB} = 0$, $t^0_{CC} = 0.5$, $t^1_{CA} = 0.5$, $t^1_{CB} = 0.5$, $t^1_{CC} = 0$).*

Difference between gapped markov chains and traditional markov chains. A main difference between our gapped markov chain and traditional markov chains is that traditional markov chains describe successive states of a system. Our gapped markov chain can do this because any gapped markov chain contains the 0-th transition matrix, which is the traditional markov chain. But it can also model sequences containing $\emptyset$. As we discussed earlier, these two changes enhance their suitability for protein classification.

Differences between sequence modeling by gapped markov chains and by amino acid compositions. Recall the amino acid pair composition technique[12] we discussed earlier. There are several differences between that technique and our gapped markov chain technique. In the former case, the discriminative information is measured by the frequencies of the amino acid pairs. If the amino acid pairs are treated as patterns, these algorithms model each protein by their length-2 pattern frequencies. The model can be interpreted as: given an ordered pair of amino acids, how likely is it that this pair occurs in the protein? In our case, the discriminative information is measured by the probabilities of the amino acid transitions. The gapped markov chain models each protein by these pairwise amino acid transition probabilities. The model can be interpreted as: given a specific amino acid m, how likely is it that another amino acid n follows it? An important advantage is that the probability model is much less likely to be affected by the protein lengths.

Example 7. *Given two sequences $p = ABBCDACDD$ and $p' = ABBD$. With gap as 0, for pair BB, its compositions in p and p' are very different, because the total lengths of p and p' are different. From a probabilistic point of view, the chance for B to follow B is the same in p as in p'. So the similarity modeled by the transition probabilities is much higher than the similarity modeled by the frequency composition. Visually checking these two sequences, we can see that p' is indeed very similar to the length-4 prefix of p.*

Algorithm 1 g-MARS_Training(*TrDB*, *g*): Train g-MARS with training dataset *TrDB* and the gap parameter *g*

```
Require: $TrDB$: training dataset. $g$: gap parameter.
Ensure: g-MARS is trained with the inputs.
1: for each class $T$ in $TrDB$ do
2:   for all $p \in Tr$ do
3:     t = MarkovChainBuilder($p$, $g$) {Build the $g$-gapped markov chain.}
4:     add$t$ to the vector set $VS$
5:   end for
6:   Train an SVM with $VS_T$ and $VS_{\neg T}$
7: end for
```

4.2. The g-MARS Algorithm

g-MARS takes a set of training data *TrDB* and a gap parameter g as the input. For each protein in *TrDB*, g-MARS builds a g-gap markov chain. The associated vector for this chain has $(g+1)*20*20$ dimensions (unlike the k^{20} dimensions for the Spectrum kernel[18]). This is because there are $20*20$ possible amino acid pairs and we need to consider transitions with gap up to g for each pair. As we will later see, we can easily set g to values as large as 10 and still attain good classification performance and reasonable running time. The markov chains for proteins from $TrDB_T$ and $TrDB_{\neg T}$ are passed as input to an SVM and it builds a classification model using these inputs. Any kernels available for the traditional SVM can be used, such as linear and RBF kernels. Building a g-gap markov chain for a set of n proteins requires $O(n*g*l)$ time, where l is the average length of the n proteins. The training time for g-MARS is the markov chain building time plus the SVM training time. Given a testing protein, the same g-gap markov chain is computed and passed to the SVM and it makes the classification decision is made by the SVM. The testing time for a length l protein in g-MARS is the markov chain building time($O(g*l)$) plus the SVM prediction time.

Although the discussion above is for the binary-class classification problem, g-MARS can be easily generalized to handle the multi-class classification problem. We turn the m-class classification problem into m reduced binary-class classification problems. Each time we pick one class out from the m classes as T and merge all the rest of the proteins as $\neg T$. In this way, way we build m SVMs, one for each target class. Given a testing protein, if there is an SVM classifying it to its target class, we classify it to that class. If more than one SVM classifies it to their target class, we classify it to the class with the highest score.

The training and testing algorithms are listed in Algorithms 1, 2 and 3.

5. Experimental Results

Datasets. In order to test the general performance of g-MARS, we use four different benchmark datasets, which were chosen since they covered a diverse

Algorithm 2 MarkovChainBuilder(p, g): Build the g-gapped markov chain for the protein p.

```
Require: p: a sequence to build from. g: gap parameter.
Ensure: Return the matrix t representing the g-gapped markov chain of p.
 1: for all $1 \leqslant h \leq g + 1$ do
 2:   for all positions $j$ in $p$ do
 3:     $t_h[p(j)][p(j + h)] + +$ {$t_h[i][j]$ stores the count of the transition from
        amino acid $i$ to $j$ w.r.t gap $h$.}
 4:     $c[p(j)] + +$ {$c[i]$ stores the total times the amino acid $i$ appearing in $p$.}
 5:   end for
 6:   for all $0 \leqslant k < |t_h|$ do
 7:     $t_h[k] = t[k]/c[k]$
 8:   end for
 9:   merge $t_h$ to $t$ {Append the matrix with gap $h$ to the overall markov chain
      matrix.}
10: end for
11: return t
```

Algorithm 3 g-MARS_Testing(p, g): Predict the class label of p.

```
Require: p: the protein with unknown class, g: the gap parameter.
Ensure: p.pc is set to the predicted class.
 1: t = MarkovChainBuilder($p$, $g$)
 2: $p.pc$ =SVM_Predict($t$) {Predict the class label of $t$ by SVM.}
```

range of characteristics and have also been used in previous work on protein classification. The first two of these datasets relate to different localizations of proteins. Being able to predict the localization of a protein is important, since it allows inferences to be made about its function, thus offering the potential for development of candidate lists of proteins to be screened for drug targets. The third dataset relates to discriminating between proteins which have different structural motifs. This is important since it offers the potential to make predictions about secondary and tertiary structure, based on primary sequence information. The fourth dataset is a multi class one and relates to discriminating between multiple subfamilies of G protein-coupled receptors, which are popular drug targets. Classification of such proteins is again important for understanding about a protein function and its potential for use as a drug target. We now describe each dataset in more detail.

The first set of data is chosen from PSORTb[8]. It contains proteins from different localizations of the bacteria. We pick out the proteins from the outer membrane of the Gram negative as the positive class and merge the proteins from the inner membrane, cytoplasmic and extra-cellular of the Gram negative as the negative class. Part of this data was previously used to evaluate the classifiers built on frequent substring patterns[22, 28]. The positive class contains 352 proteins and the negative class contains 1013 proteins.

Table 1: G Protein-Coupled Receptor dataset list.

Subfamily	#protein	% of Dataset
Class A Rhodopsin like	**1884**	**69.4%**
Amine		
Acetylcholine	66	15%
Adrenoceptors	120	27.3%
Dopamine	94	21.4%
Serotonin	159	36.2%
Class B Secretin like	**309**	**11.4%**
Class C Metabotropic glutamate/ pheromone	**206**	**7.6%**
Class D Fungal pheromone	**65**	**2.4%**
Class E cAMP receptors	**10**	**0.4%**
Class F Frizzled/Smoothened family	**130**	**4.8%**
Class Z Archaeal/bacterial/ fungal opsins(Non-GPCR)	**110**	**4.1%**
Total	**2714**	**100%**

The second set of data is proteins from different subcellular localizations from the Proteome Analyst Project[21]. We choose the proteins from the extracellular localization (127 proteins) as the positive class and the proteins from the intracellular localization as the negative class (3166 proteins). The third set of data is the outer membrane proteins versus the globular proteins which was used to evaluate the classifier built on amino acid compositions[10]. It contains 377 proteins from bacterial outer membrane and 674 Globular proteins.

The fourth set of data uses the G Protein-Coupled Receptor (GPCR)[4], the biggest known protein family. The GPCR database contains five level-0 GPCR classes (level-0 subfamilies). The largest subfamily is the Class A Rhodopsin like subfamily. It can be further divided into 16 level 1 subfamilies and more level 2 subfamilies. Classifiers have been developed to classify GPCR proteins from non-GPCR ones, the GPCR proteins from level 1 subfamilies, as well as the GPCR proteins from level 2 subfamilies[4]. We perform two experiments on this data. For the first experiment, we try to classify proteins from the level-0 subfamilies. Besides the five GPCR level-0 subfamilies, we add a non-GPCR family in order to test the ability for g-MARS to separate the GPCR proteins from non-GPCR ones. All six families can be obtained from *http://www.gpcr.org*[9]. For the second experiment, we try to classify proteins from the level 2 subfamilies. We select 4 level-2 subfamilies belonging to the Amine subfamily under level-0 subfamily Class A Rhodopsin like, namely, acetylcholine, adrenoceptors, dopamine and serotonin. These two experiments are multi-class classification problems. The specification of all the five experiments is listed in Tables 1 and 2.

Algorithms. We compare the accuracy of g-MARS against several algo-

Table 2: Binary-class classification dataset list.

Dataset Description		# Protein		% of Dataset	
D	$\neg D$	D	$\neg D$	D	$\neg D$
Outer Membrane Proteins (OMP)*	Inner Membrane, Extracellular, Cytoplasm	352	1013	25.8%	74.2%
Extracellular proteins	Intracellular proteins	127	3166	3.9%	96.1%
Outer Membrane Proteins (OMP)*	Globular proteins	377	674	35.9%	64.1%

rithms: i) the spectrum kernel[18](Spectrum for short), which has been claimed to be better than Fisher kernel[18], ii) an amino acid composition classifier[10](AAC for short), iii) an amino acid pair composition with gap constraints classifier[12](AAPC for short), iv) simple markov chain classifier[5](MC for short), v) Frequent Substring Pattern based SVM[28](FS for short), vi) Generalised markov model (GMM[26]). The reasons for choosing these algorithms are: 1.g-MARS, AAPC and FS are all SVM-based hybrid algorithms. The difference between them is the way they "translate" sequences into vectors. 2.Spectrum is a famous protein classifier which makes use of the SVM and self-defined kernel function. 3.The AAC, GMM and MC methods are not based on support vector machines. They simply sum up the scores computed in each way up to make decisions. They are simple, well-known methods. All the experiments were conducted on a Mac Os X system with a 2.8GHz CPU and 4GB memory. We used the LIBSVM[3] package. MC required no parameter settings. For the Spectrum kernel, we used $k = 3$. For g-MARS, AAC and AAPC, for each dataset we used the gap that gave the best average performance (according to f-measure, see below), using 5-fold cross validation with a verification dataset (a subset of the training data whose class labels are known to the classifiers, but which is not used in training). For the FS algorithm we mined the frequent substring patterns from the target class having minimum length as 3, minimum support as either 0.1% or 3 (whichever is greater) and minimum confidence of 90%[28]. For g-MARS, FS and AAPC, we used the RBF kernel. The gamma and cost parameters for this kernel were chosen using the Gridsearch tool in the LIBSVM package[3], again using a verification dataset (a subset of the training data in each fold used for parameter optimisation). For g-MARS, a gamma value of 0.0078125 and cost value of 32.0 or 2048.0 gives generally good performance. For GMM, we tested the three configurations that were evaluated by the authors in their paper [26]. The first of these, a 6th order single nucleotide Markov model, produced the best results in all our datasets and we list its performance in the tables.

Evaluation. In order to give a comprehensive analysis of how good the classifiers are, we use 4 metrics: accuracy (a), precision (p), recall (r) and f-measure (f) as:

$a = \frac{|\{t \subseteq TeDB|t.pc=t.c\}|}{|TeDB|}$; $p = \frac{|\{t \subseteq TeDB_T|t.pc=T\}|}{|\{t \subseteq TeDB|t.pc=T\}|}$;

Table 3: Results (%) of the three binary-class experiments.

	OMP vs. Inn+Ext+Cyt				Extra vs. Intra				OMP vs. Globular			
Algorithm	**A***	**P**	**R**	*F*	**A**	**P**	**R**	*F*	**A**	**P**	**R**	*F*
gMARS	95.16	**94.97**	85.8	90.15	**98.66**	93.68	70.08	**80.18**	**96.76**	**95.98**	94.96	**95.47**
Spectrum	94.36	92.83	84.66	88.56	98.15	92.31	56.69	70.24	95.62	95.59	92.04	93.78
FS	90.04	79.35	82.95	81.11	98	**95.52**	50.39	65.98	91.53	82.88	**96.29**	89.08
AAC	78.38	51.49	78.69	62.25	88.59	18.64	58.27	28.24	80.02	67.88	84.08	75.12
AAPC	**95.6**	94.24	88.35	**91.2**	98.45	93.18	64.57	76.28	92.86	85.78	96.02	90.61
MC	82.49	61.06	**88.64**	72.31	94.02	36.74	**76.38**	49.62	86.77	76.44	91.25	83.19
GMM	34.1	42.74	**100**	59.89	97.4	65.2	39.4	36.25	90.0	88.55	77.34	82.27

∗ *A: accuracy, P: precision, R: recall, F: f-measure.*

$r = \frac{|\{t \subseteq TeDB_T | t.pc=T\}|}{|\{t \subseteq TeDB_T\}|}$; $\qquad f = \frac{2*p*r}{(p+r)}$.

For multi-class classification, a different overall accuracy measurement is used: $a = \sum_{T_i} \frac{|\{t \subseteq TeDB_{T_i} | t.pc=t.c\}|}{|TeDB|}$. The accuracy for each target class T_i is calculated as: $a_i = \frac{|\{t \subseteq TeDB_{T_i} | t.pc=t.c\}|}{|TeDB_{T_i}|}$. The accuracy measurement tells how many proteins are classified correctly overall. For the rare-class classification case, precision and recall are more meaningful. When comparing algorithms, the f_measure is a standard way of combining precision and recall to get a single measure. We used stratified 5-fold cross validation for testing.

Performance on binary-class data. The accuracies of the five algorithms on the 3 binary-class classification problems are listed in Table 3. g-MARS performs strongly for the first set of data, the outer membrane proteins vs. the inner membrane, extracellular and the cytoplasm proteins. In this set the amino acid pair composition algorithm works quite well. g-MARS gives generally good performances on all the datasets, because the discriminative information of markov chains in g-MARS does not rely on any particular property of proteins being in specific domains. The MC classifier given in the second last row uses the log odd ratio score to classify the proteins[5]. The performance tells that simply adding up the score ratios from different classes does not give good answers. This partly shows the power of using the SVM to make the decisions.

Performance on GPCR subfamilies. The classification results for the GPCR level-2 and level-0 subfamilies are given in Tables 4 and 5, respectively. The diversities of subfamilies are greater for level-0 proteins than for level-2 proteins. That is the reason why generally we gain better results for level-2 proteins. There are certain subfamilies in level-0 that are easily separated from other subfamilies such as Class E cAMP receptors. Most of the classifiers do not make mistakes for proteins from this family. By looking at this family we know that the structures of the proteins within this family are quite different from proteins of other families. Some proteins contain long contiguous asparagine and histidine. The performances for most classifiers are quite good for identifying which protein belongs to Class A Rhodopsin like subfamily (high percentages in the first row of Table 5). This is due to the abundant proteins of this family. So as an observation about Table 5, we can say that for the classifiers tested here, having more testing data means a more accurate the model can be built.

Table 4: The accuracy (%) of the GPCR level-2 subfamilies prediction.

Level-2 Subfamily	gMARS	Spectrum	FS	AAC	AAPC	MC	GMM
Acetylcholine	**100**	95.45	95.45	87.88	93.93	95.45	85.52
Adrenoceptors	**100**	**100**	**100**	62.5	**100**	95.83	83.67
Dopamine	**98.93**	95.74	94.68	76.6	85.11	85.11	80.21
Serotonin	98.11	**100**	97.48	77.99	94.97	94.34	75.48

Table 5: The accuracy (%) of the GPCR level-0 subfamilies prediction.

Level-0 Subfamily	gMARS	Spectrum	FS	AAC	AAPC	MC	GMM
Class A Rhodopsin like	**99.84**	99.52	98.57	78.66	99.73	91.77	77.93
Class B Secretin like	**99.38**	98.06	95.47	60.2	95.46	96.12	94.0
Class C Metabotropic glutamate/pheromone	**98.06**	95.15	91.26	76.21	97.09	93.69	82.7
Class D Fungal pheromone	89.23	86.15	81.54	89.23	83.07	**95.38**	76.2
Class E cAMP receptors	**100**	**100**	**100**	90	90	**100**	83.0
Class F Frizzled/Smoothened family	**98.46**	97.69	96.15	93.85	92.31	90.77	85.53
Class Z Archaeal/bacterial /fungal opsins (non-GPCR)	**96.36**	94.55	86.36	92.73	92.73	95.45	90.12

The more distinctive the data is, the easier it is for the model to make correct decision. This is generally true for most feature-based classifiers.

5.1. Statistical Evaluation

For each pair of classifiers on a given dataset, we computed a p-value assessing whether their difference in performance was significant. Given a classifier and a dataset, there are five performance values, one for each fold of the cross validation. We compare the two sets of five performance results of two classifiers, using a paired t-test. If the p-value was less than 0.05, then we categorised that classifier comparison as a win/loss. Otherwise it was classified as a draw.

For the three binary classification problems, we used the f-measure value as the performance metric. For the GPCR level-0 and level-1 families, as a performance metric we used the overall accuracy from each cross validation of all the subfamilies in the same level as:

$$\text{overall accuracy} = \sum_i \frac{\#\text{True Positive in } \ D_i}{|D_i|}.$$

So for a given classifier, it was compared against the other 6 classifiers over the 5 classification tasks ($6 \times 5 = 30$ comparisons in total). The wins, draws and losses for the classifier were recorded. Table 6 shows the results for g-MARS against the other classifiers. It won **21** times, drew **9** times and lost **0** times. Of the other classifiers, the Spectrum ranked the second best with **16** wins, **13** draws and **1** loss. The third best algorithm was the AAPC and its performance

Table 6: Statistical comparison of g-MARS against other classifiers using paired t-test

dataset	Spectrum	FS	AAC	AAPC	MC	GMM
OMP vs. Inner+Extra+Cyt.	D	W	W	D	W	W
Extra vs. Intra	D	W	W	D	W	W
OMP vs. Globular	D	W	W	W	W	W
GPCR level 0 subfamilies	W	W	W	W	D	W
GPCR level 2 subfamilies	D	D	W	D	W	W

D (draw) for g-MARS if p-value >= 0.05 else W (win) for g-MARS.

was **13** wins, **14** draws and **3** losses. Comparing directly against the Spectrum, g-MARS won on 1 dataset and drew on the other 4.

5.2. How to choose the proper gap.

The gap can be chosen by performing cross validation on the training dataset. Set aside a portion of training data as test data, try different gaps and choose the one which yields best accuracy. A cost of this strategy is that it increases the overall time required for training. We can also make some general remarks about gap behaviour. Figure 1 shows the change in accuracy for g-MARS with various gaps. For gap 0, which is the case in the traditional markov chain, the performance is poor. With the increase of the gap, overall performance becomes stable. The f-measure achieves its peak value when the gap is set to 11. So instead of using cross validation, as a heuristic, one could also begin by using the gap as 7 and then increase and decrease the gap from this value, finishing when the result remains stable (changes are smaller than a certain θ). Of course such a strategy doesn't guarantee an optimal gap, but can help to reduce the training time.

5.3. Running time

For classifiers working on large volumes of data, time efficiency is a relevant issue. We discussed the time complexity of g-MARS in Section 4.2. We also measured the running time for g-MARS for various gaps on both 1) OMP vs. Inner, Extra and Cytoplasm dataset and 2) Extracellular vs. Intracellular Dataset. The behavior is shown in Figure 2, which provides curves for both the training time and the testing time, across 5 folds of cross validation. We see that both training and testing time increase roughly linearly with the increment of the gap, as expected from our discussion in Section 4.2.

5.4. Robustness experiment.

In our final experiment, we train the classifiers with original training proteins. In the testing phase, we select a random position of each protein, chop off a certain length of it starting from that position and concatenate the remaining

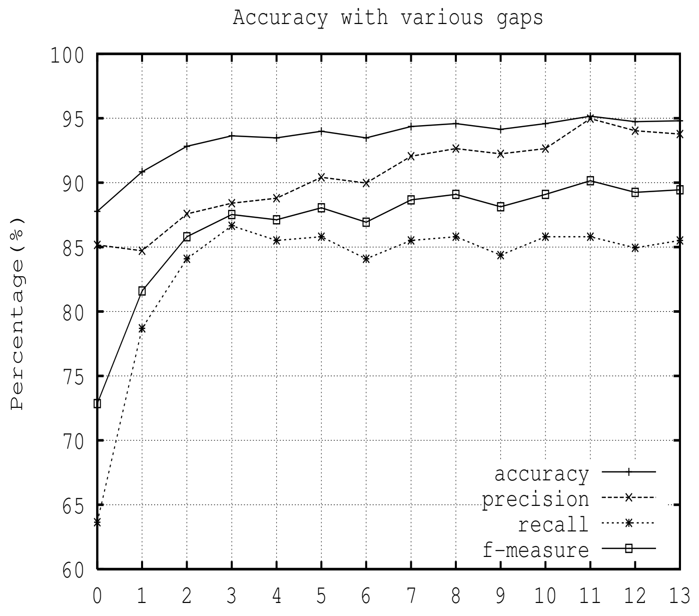
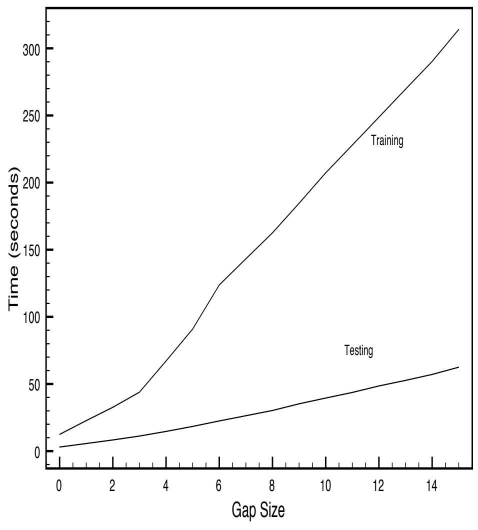
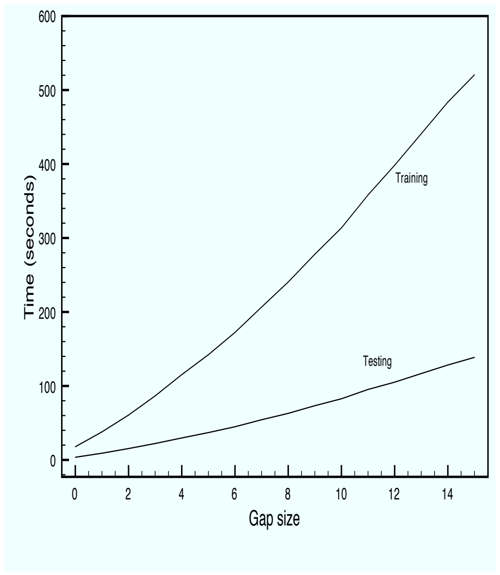


Figure 1: g-MARS performance for varying gap on OMP vs. Inner, Extra, Cytoplasm dataset

parts together to form a new sequence. The practical intuition is that sometimes we may want to classify a certain part of an unknown protein sequence, rather than use of all of it, while in other situations we might know there are some mistakes in the sequence (possibly due to contamination) of the data and we wish to remove them before attempting classification. Specifically we chop off from each sequence, a random snippet whose length is a certain percentage of the total sequence length. We increase this percentage to allow longer and longer snippets to be chopped. From a machine learning point of view, this allows us to test the robustness of the classification techniques. The results of this experiment is shown in Figure 3. We see that all the classifiers suffer deterioration in performance due to the chop-offs. The f-measures are at their peak values when the chop-off is 0%. As chop-off lengths increase, the general performances of all the classifiers decrease. We judge the robustness of each algorithm by calculating the standard deviation of the 11 f-measures with various chop-off percentages. The standard deviation formula is $\sigma = \sqrt{\frac{\sum_i^N (x_i - \mu)^2}{N}}$, where $N = 11$, x_i is the i-th f-measure and μ is the average value of the f-measures. The standard deviations for g-MARS, Spectrum, FS, AAC, AAPC and MC are 3.32, 11.36, 3.45, 3.57, 3.66 and 4.22, respectively. From these results we can see that g-MARS maintains the best performance for this dataset. FS is quite robust in this experiment, because the algorithm only relies on the substring patterns. If the chop-off snippet does not contain any pattern or does not cut



(a) OMP vs. Inner, Extra and Cytoplasm Dataset



(b) Extracellular vs. Intracellular Dataset

Figure 2: g-MARS running times when varying gap size

up any pattern, the decision will not be affected. The Spectrum has the biggest changes in performance. The problem is that it relies on using all k-mers, so it is very likely that some common k-mers get chopped off. If pattern pruning strategies were applied to select the most discriminative k-mers rather than all of them, then its performance might be better.

6. Further Analysis: A Dynamical Systems Perspective

In light of the experimental results we have presented, we now further explore the question of why the performance of g-MARS is so strong, despite the relative simplicity of its approach. Clearly, the accuracy of g-MARS is related to the quality of the feature space it constructs, using ordered pairs of amino acids. As we have seen, such pairs can be interpreted as “gapped markov chain” features. In this section, we present another framework for interpreting these ordered pair features.

At a high level, we can explain the naturalness of our feature space by showing its connection to representations that are used in nonlinear time series analysis and dynamical systems theory [2]. In particular, the g-MARS feature space can be derived by performing an appropriate delay co-ordinate embedding of the protein sequence. We begin our description by first providing some background about dynamical systems.

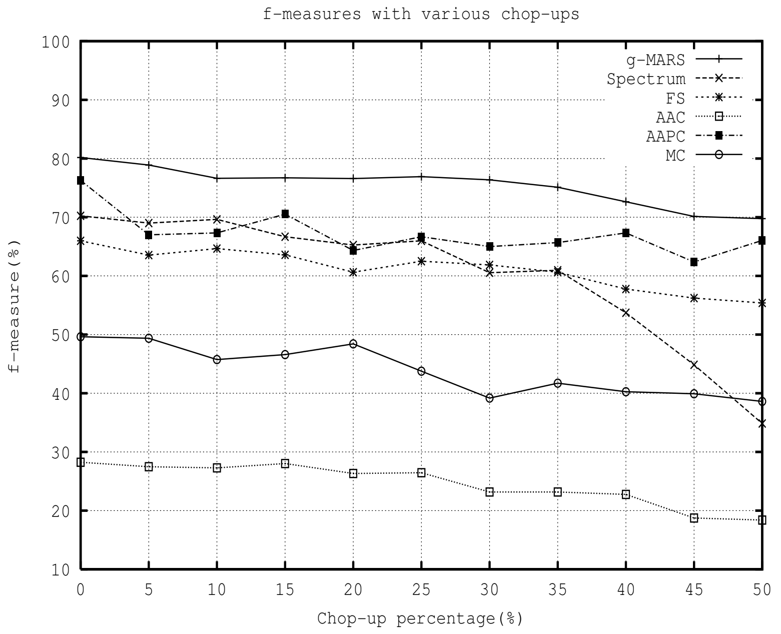


Figure 3: The performances change with various chop-off lengths on Extracellular vs. Intracellular dataset.

6.1. Dynamical Systems

A dynamical system is a system that evolves over time. Classical examples include a mass on a spring, a collection of planets or an electrical circuit. At each point in time, the system can be described by a set of state variables, known as the *dimension* of the system. In the case of a swinging pendulum, the state variables might correspond to the angular position and velocity of the pendulum.

A dynamical system's behaviour can be represented using a phase space (also known as state space) portrait, which is a multidimensional space, whose axes correspond to the different state variables and in which the system 'flows' along a trajectory between successive state configurations. Analysis of such trajectories may potentially reveal characteristics such as fixed points, periodic behavior, or chaotic behaviour.

In general though, the state space of a dynamical system can have many dimensions (possibly even an infinite number), and sometimes dimensions may be measurable or even known. So to understand the properties of the system, it is necessary to use partial information.

A common method of measuring the behaviour of a dynamical system is to measure the values of a single state variable, at equally spaced intervals over time. This yields a time series of N observations (a sequence of scalar measurements):

$$x(t) = \{x_1, x_2, x_3, \dots x_N\} \tag{4}$$

In non linear dynamics, such a time series may be transformed by using the technique of delay co-ordinate embedding. One constructs a set of vectors $z_1, z_2, z_3 \dots$ by sampling from this time series. Each vector has m components, sampled at intervals of τ, as follows:

$$\begin{aligned}
z_1 &= x_1, x_{1+\tau}, x_{1+2\tau}, \dots, x_{1+(m-1)\tau} && (5)\\
z_2 &= x_2, x_{2+\tau}, x_{2+2\tau}, \dots, x_{2+(m-1)\tau} && (6)\\
z_3 &= x_3, x_{3+\tau}, x_{3+2\tau}, \dots, x_{3+(m-1)\tau} && (7)\\
z_4 &= \dots && (8)
\end{aligned}$$

where τ is a parameter known as the *delay time* or *lag* and m is known as the embedding dimension.

Taking these vectors and plotting them in m dimensional space yields a *delay space* portrait of the system. The key idea here is that constructing a delay co-ordinate embedding can provide insight into the dynamics of the system. Indeed a central theorem in the area due to Takens [24], states that given a noise free time series and a sufficiently large value for m ($m = 2d + 1$, where d is the true dimension of the system), the trajectory of the system in the delay co-ordinate embedding (delay space) is topologically equivalent (diffeomorphic) to the trajectory of the system in the full, unobserved phase space. This implies that conclusions one makes about the dynamics in delay space can in a formal sense, also be valid with regard to the dynamics of the system in the full phase space.

6.2. A Protein as a Dynamical System

In our context, the dynamical system of interest is a protein. Although a protein is not a time series, we can still process it in an analogous way. For a protein, the amino acid order plays the role of successive time intervals.

For example, given a protein $ACCE$ of length $n = 4$, we view it as the time series

$$\begin{aligned}
p(t) &= p_1, p_2, \dots, p_n\\
&= A, C, C, E
\end{aligned}$$

Thus, by starting at the beginning of the protein sequence and traversing it to the end, one can trace out a trajectory of different states at different times, where the state variable corresponds to the type of amino acid.

One issue is that the states of the protein here are discrete symbols, rather than values measured on a continuous scale, as is usual in a dynamical system. Hence, to remain close to the dynamical system analogy, it is useful to convert the symbols in the protein time series into a continuous scale. There are a variety of possibilities that can be employed here [23], based on the physiochemical properties of the amino acids, such as charge density, or volume.

For illustration purposes, we choose to characterise the amino acids in a protein by their hydrophobicity values (tendency to repel water). Hydrophobicity values are known to relate to the three dimensional structure of a protein [23]. Each amino acid can be replaced with a number on a continuous scale, representing its hydrophobicity. We use the Engelman hydrophobicity scale (GES-scale) [6], which uses the mapping:

$A = 1.6$	$C = 2.0$	$D = -9.2$	$E = -8.2$
$F = 3.7$	$G = 1.0$	$H = -3.0$	$I = 3.1$
$K = -8.8$	$L = 2.8$	$M = 3.4$	$N = -4.8$
$P = -0.2$	$Q = -4.1$	$R = -12.3$	$S = 0.6$
$T = 1.2$	$V = 2.6$	$W = 1.9$	$Y = -0.7$

So the protein $ACCE$ would become the sequence $\{1.6, 2.0, 2.0, -8.2\}$.

After translating the protein into a numeric "time series', we can now plot the trajectory of the protein in delay space. In doing this, we must choose values for both m (the embedding dimension) and τ (the time delay/lag). An example is given next:

Example 8. *Consider the protein* $LKCHKLVPPVWKTCPEGKNLCYKMFMVSTSTVPVKRGCIDVCPKNSALVKYVCCSTDKCN$. *Let* $m = 2$ *and* $\tau = 3$. *The delay space vectors are then:*

$(p(t)$	,	$p(t+3))$
(*2.8*	,	-3.0)
(*-8.8*	,	-8.8)
(*2.0*	,	2.8)
(*-3.0*	,	2.6)
(*-8.8*	,	-0.2)
...		

In figure 4, we show the delay space trajectory for this protein. The trajectory begins at the point labelled 'START' and finishes at the point labelled 'END'. Some clustering in the paths can be observed, with a higher preference for i) horizontal paths between approximately $(-8.8, 2)$ *and* $(2, 2)$*, and ii) vertical paths between approximately* $(2, -9)$ *and* $(2, 2)$.

6.3. Connections to g-MARS

It may be apparent that the feature representation used by g-MARS for an individual protein can be derived from the protein's delay space trajectories constructed using $\tau = 1, 2, \ldots, g + 1$ and $m = 2$.

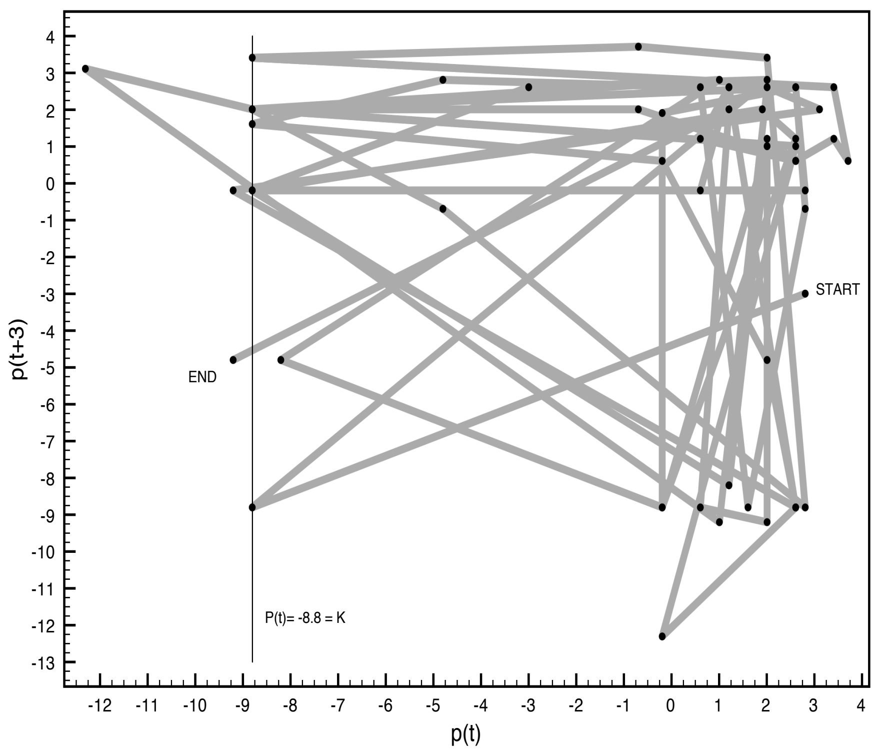


Figure 4: Delay space portrait when $m = 2$ and $\tau = 3$ using the hydrophobicity profile of the protein LKCHKLVPPVWKTCPEGKNLCYKMFMVSTSTVPVKRGCIDVCPKNSALVKYVCCSTDKCN A Poincare section at $p(t) = -8.8$ is also shown.

Note that the g-MARS representation of a protein is one dimensional, whereas the delay space representation is two dimensional. Hence it is necessary to use dimension reduction when computing the g-MARS features. This is accomplished by determining intersections of the protein's trajectory with a cutting plane, known as a *Poincare section*. This is a well known tool for visualising and revealing structures and patterns in delay space.

Looking again at figure 4, a Poincare section is drawn as the vertical line $p(t) = -8.8$ (i.e. $p(t) = K$, since $K = -8.8$). This line $p(t) = -8.8$ has eight delay vectors lying directly on it, corresponding to the values of $p(t+3) = \{K = -8.8, P = -0.2 \text{ (twice)}, A = 1.6, C = 2.0\text{(three times)}, M = 3.4\}$. Thus, we can make inferences about the distribution of amino acids that follow the amino acid K, at a delay of $\tau = 3$. So $Pr(K|K) = 1/8, Pr(P|K) = 2/8, Pr(A|K) = 1/8, Pr(C|K) = 3/8.Pr(M|K) = 1/8$. These correspond exactly to the values of the g-MARS features $\{KK, KP, KA, KC, KM\}$ with a gap size of $g = \tau - 1 = 2$. To compute the values of feature pairs that do not begin with K, one would compute the statistics from appropriate Poincare sections: $p(t) = A = 1.6, p(t) = C = 2.0, p(t) = D = -9.2, \ldots$.

So in essence, the g-MARS feature values (for a given gap size g) correspond to statistics computed from particular projections of the protein's trajectory in delay space. Of course since the g-MARS algorithm uses features corresponding to pairs of amino acids whose gap is $\leq g$, this means that its features must be in fact derived from *multiple* delay space portraits, each corresponding to a particular value of g. From each delay space profile, one computes the probabilities of projections of amino acid pairs as above.

This dynamical system interpretation of the features used by g-MARS is attractive for two key reasons:

- It suggests that the features used by g-MARS correspond to certain views of the protein's trajectory in delay space. Furthermore, Taken's theorem, mentioned above, states that this delay space trajectory is *topologically equivalent* to some unobserved trajectory of the protein in its full phase space. This trajectory of a protein in its full unobserved phase space could be related to possibly hundreds of state variables, some of which might, for example, describe its 3-dimensional structure. *In other words, we can argue that the features used by g-MARS are in some sense faithful and descriptive of the higher order behavior of the protein.*

- It offers insights into how the g-MARS feature space might be generalized. In particular, the features for g-MARS use an embedding dimension of $m = 2$. However, the delay space perspective suggests that the use of higher embedding dimensions is indeed possible and might be worthwhile.

  There is considerable discussion in the dynamical systems literature about how to choose an appropriate embedding dimension and many methods have been proposed, such as false nearest neighbors [15]. On a theoretical level, Taken's theorem only holds if $m \geq 2k + 1$, where k is the true dimension of the phase space of the original system. However k itself is unlikely to be known. More pragmatically though, it is known that useful information can still be recovered from an embedding, even if $m < 2K + 1$ [1, 2]. So in the case of g-MARS, even though the choice of $m = 2$ may not be optimal, it is not unexpected that the performance of the classifier is nevertheless very strong. For future work, it would be interesting to investigate adapting g-MARS for use with higher values of m.

  Similar to the choice of embedding dimension m, the dynamical systems literature also contains a variety of methods which consider how to choose an appropriate value of the delay τ. One well known approach from [7], is based on computing the mutual information between $x(t)$ and $x(t + \tau)$ and then choosing the value of τ which yields the first minimum of mutual information. In the case of g-MARS, we have chosen to take a pragmatic approach to the choice of τ (i.e. the gap g). The best value for g is determined experimentally by measuring the classifier's performance on a set of verification data. Moreover, the features used by g-MARS do not just correspond to a single value of g, but are instead computed using all the gap values $0, 1, \ldots, g$.

7. Conclusion

In this paper we have extended the traditional markov chain to the gapped markov chain. We proposed the g-MARS classifier, which uses gapped markov chains and support vector machines to classify proteins. We have described theoretical justification for the gapped features used by g-MARS by relating them to techniques used in dynamical systems theory. g-MARS has several merits: It is a simple, yet intuitive approach and the technique is robust to data with high diversity (i.e. data with various lengths). The growth of the gap length increases the dimension of the vectors linearly rather than exponentially like the Spectrum kernel, so it is realistic to use large gaps. Experimental results show it has generally high accuracy for a range of protein datasets with diverse characteristics. Overall, we believe g-MARS is a very practical algorithm to handle protein classification.

Acknowledgements: This work was partially supported by NICTA. NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program. We are grateful to Limsoon Wong for comments on a draft of this paper.

References

[1] H. D. I. Abarbanel. *Analysis of Observed Chaotic Data.* Springer, New York, 1996.

[2] Elizabeth Bradely. Time series analysis. In D. Hand and M. Berthold, editors, *Intelligent Data Analysis: An Introduction*, pages 199–227. Springer-Verlag, 2003.

[3] C. Chang and C. Lin. *LIBSVM: a library for support vector machines*, 2001.

[4] B. Cheng, J. Carbonell, and J. Klein-Seetharaman. Protein classification based on text document classification technique. *PROTEINS: Structures, Function and Bioinformatics.*, 58:955–970, 2005.

[5] R. Durbin, S. Eddy, A.Krogh, and Graeme Mitchison. *Biological sequence analysis—-Probabilistic models of proteins and nucleic acids.* Cambridge University Press, 1998.

[6] D. M. Engelman, T. A. Steitz, and A. Goldman. Identifying nonpolar transbilayer helices in amino acid sequences of membrane proteins. *Annu Rev Biophys Biophys Chem*, 15:321–353, 1986.

[7] A. M. Fraser and H. L. Swinney. Independent co-ordinates for strange attractors from mutual information. *Phy. Rev. A.*, 33(2):1134–1140, 1986.

[8] J. L. Gardy, M. R. Laird, F. Chen, S. Rey, C. J. Walsh, M. Ester, and F. S. L. Brinkman. PSORTb v.2.0: Expanded prediction of bacterial protein subcellular localization and insights gained from comparative proteome analysis. *Bioinformatics*, 21(5):617–623, 2005.

[9] GPCRDB. http://www.gpcr.org.

[10] M. Michael Gromiha and Makiko Suwa. A simple statistical method for discriminating outer membrane proteins with better accuracy. *Bioinformatics*, 21(7):961–968, 2005.

[11] Dan Gusfield. *Algorithms on Strings, Trees, and Sequences - Computer Science and Computational Biology.* Cambridge University Press, 1997.

[12] S. Huang, R. Liu, C. Chen, Y. Chao, and S. Chen. Prediction of outer membrane proteins by support vector machines using combinations of gapped amino acid pair compositions. In *Proceedings of the Fifth IEEE Symposium on Bioinformatics and Bioengineering*, pages 113–120, 2005.

[13] Tommi Jaakkola, Mark Diekhans, and David Haussler. Using the fisher kernel method to detect remote protein homologies. In *Proceedings of the Seventh International Conference on Intelligent Systems for Molecular Biology*, pages 149–158, 1999.

[14] Xiaonan Ji, James Bailey, and Kotagiri Ramamohanarao. g-mars:protein classification using gapped markov chains and support vector machines. In *Proceedings of the Third IAPR International Conference on Pattern Recognition in Bioinformatics (PRIB)*, pages 165–177, October 15-17, 2008.

[15] H. Kantz and T. Schreiber. *Nonlinear time series analysis, 2nd Edition.* Cambridge University Press, 2004.

[16] Rui Kuang, Eugene Ie, Ke Wang, Kai Wang, Mahira Siddiqi, Yoav Freund, and Christina S. Leslie. Profile-based string kernels for remote homology detection and motif extraction. In *Proceedings of the Third IEEE Computational Systems Bioinformatics Conference (CSB)*, pages 152–160, Stanford, CA, USA, August 16-19, 2004.

[17] C. S. Leslie, E. Eskin, A. Cohen, J. Weston, and W. S. Noble. Mismatch string kernels for discriminative protein classification. *Bioinformatics*, 20(4), 2004.

[18] C. S. Leslie, E. Eskin, and W. S. Noble. The spectrum kernel: A string kernel for SVM protein classification. In *Proceedings of the Pacific Symposium on Biocomputing*, pages 566–575, 2002.

[19] Li Liao and William Stafford 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.

[20] Thomas Lingner and Peter Meinicke. Remote homology detection based on oligomer distances. *Bioinformatics*, 22(18):2224–2231, 2006.

[21] Z. Liu. Predicting protein subcellular localization from homologs using machine learning algorithms. Master's thesis, Dept of Computer Science, University of Alberta, 2002.

[22] R. She, F. Chen, K. Wang, M. Ester, J. L. Gardy, and F. S. L. Brinkman. Frequent-subsequence-based prediction of outer membrane proteins. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 436–445, 2003.

[23] S. Sun and Parthasarathy. Protein sequence and structure relationship arma spectral analysis: Application to membrane proteins. *Biophysical Journal*, 66:2092, 1994.

[24] F. Takens. Detecting strange attractors in turbulence. In *Dynamical Systems and Turbulence, Lecture Notes in Mathematics, Vol. 898*, pages 366–381. Springer, 1981.

[25] Vladimir Vapnik. *The Nature of Statistical Learning Theory.* Springer-Verlag, 1995.

[26] Junwen Wang and Sridhar Hannenhalli. Generalizations of markov model to characterize biological sequences. *BMC Bioinformatics*, 6:219, 2005.

[27] Jason Weston, Christina S. Leslie, Eugene Ie, Dengyong Zhou, André Elisseeff, and William Stafford Noble. Semi-supervised protein classification using cluster kernels. *Bioinformatics*, 21(15):3241–3247, 2005.

[28] S. Zhou and K. Wang. Localization site prediction for membrane proteins by integrating rule and svm classification. *IEEE Trans. Knowl. Data Eng.*, 17(12):1694–1705, 2005.