g-MARS:Protein Classification using Gapped Markov Chains and Support Vector Machines

Xiaonan Ji[1], James Bailey[1], and Kotagiri Ramamohanarao[1]

NICTA Victoria Laboratory
Department of Computer Science and Software Engineering
University of Melbourne, Australia
{xji, jbailey, rao}@csse.unimelb.edu.au

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. We perform a thorough experimental study and compare g-MARS to several other state-of-the-art protein classifiers. Overall, we demonstrate that g-MARS has superior accuracy and operates efficiently on a diverse range of protein families.

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 sequences (DNA, RNA or amino acid sequences), high sequence similarity usually implies significant functional or structural similarity."[7]. 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 prime advantage of g-MARS is its superior 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.

Related work: 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[6] 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[8] is an improvement of the pure amino acid composition algorithm[6]. 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, which is useful to alleviate the potential bias introduced by the limited information from the training datasets. 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.

The spectrum kernel[11] is a support vector machine algorithm that calculates the similarity of two sequences by their common k-mers. In practice, the spectrum kernel works quite well[11]. 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[10], the sharing of the similar k-mers, along with the identical ones, is used to measure the similarity.

Previous work by Wang et al [14] presents an interesting, but very general framework (GMM) for using markov models to classify proteins using amino acid feature combinations which may include gaps. 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 not feasible to try to experimentally benchmark against), 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.

The Fisher kernel[9] combines the support vector machine and the hidden markov model. 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[9]. It is instead calculated by a classic relational kernel such as the RBF kernel.

Work[13, 15] 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.

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 $\neg T$. $\mathit{TrDB}_T = \{p \in \mathit{TrDB} \mid p.c = T\}$ is called the training positive set and $\mathit{TrDB}_{\neg T} = \{p \in \mathit{TrDB} \mid p.c \neq T\}$ is called the training negative set. Corresponding definitions exist for sets of testing instances TeDB_T and $\mathit{TeDB}_{\neg T}$.

2 g-MARS Methodology

Training the g-MARS classifier has two main phases. Firstly, g-MARS builds for each $p \in \mathit{TrDB}$, 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 other 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 $t_{CC} = \frac{1}{2}$.*

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 2 *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.*

2.1 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, our markov chain won't contain 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 the null transition is redundant. Secondly, by removing the null transitions, we reduce the model size, which benefits for the classification process used later. In practice, the exclusion of these transitions does not impair classification accuracy.

Since we remove the null state and the null transitions, the sum of all the out-going transition probabilities in our markov chain model won't necessarily be 1. This is different from the markov chain introduced in the last section. From another point of view, the "rest" of the probability of a state goes to the null state which is "hidden".

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 3.

Example 3 *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 4 *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 2). 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^1_{AC}.

In practice, we should not expect two similar proteins to share many such common transition probabilities. Instead, we would expect transition probabilities of proteins from the same class to have smaller variance and transition probabilities of proteins from different classes to have larger variance. SVMs are good at detecting such differences and so we use them for deriving a decision hyperplane that can separate gapped markov chain features of proteins from different classes.

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*.

Differences between gapped markov chains and traditional markov chains. As we can see, there are two main differences between our gapped markov chain and traditional markov chains. First of all, the summation of all out-going transition probabilities of a state is not necessarily to be 1 in our gapped markov chain, but it is a property of traditional markov chains. Secondly, the traditional markov chains describe successive states of a system. Our gapped markov chain can do that 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[8] 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.

2.2 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[11]). This is because there are $20*20$ possible amino acid pairs and we need

to consider transitions with gap up to g for for each pair. In practice, we can easily set g to be as large as 10 and not incur dimensionality overload in classification. 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.

3 Experimental Results

Datasets. In order to test the general performance of g-MARS, we choose several different benchmark datasets, which cover a diverse range of characteristics. The first set of data is chosen from PSORTb[4]. 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 used to evaluate the classifiers built on frequent substring patterns[13, 15]. The positive class contains **352** proteins and the negative class contains **1013** proteins. The second set of data is proteins from different subcellular localizations from the Proteome Analyst Project[12]. 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[6]. It contains **377** proteins from bacterial outer membrane and **674** Globular proteins.

The fourth set of data uses the G Protein-Coupled Receptor (GPCR)[2], 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[2]. 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

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%**

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%

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*[5]. 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 algorithms: i) the spectrum kernel[11](Spectrum for short), which has been claimed to be better than Fisher kernel[11], ii) an amino acid composition classifier[6](AAC for short), iii) an amino acid pair composition with gap constraints classifier[8](AAPC for short), iv) simple markov chain classifier[3](MC for short), v) Frequent Substring Pattern based SVM[15](FS for short), vi) Generalised markov model (GMM[14]). 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. We implemented all algorithms in Java using JDK version 1.4. All the experiments were conducted on a UNIX system with a 3.0GHz CPU and 1.5GB memory. We used the LIBSVM[1] Java 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%[15]. For g-MARS, FS and AAPC, we used the RBF kernel. The gamma and cost parameters for this kernel were chosen using the tool in the LIBSVM package[1]. For g-MARS, one can use gamma as **0.0078125** and cost as **32.0** or **2048.0** to expect generally good performance. For GMM, we tested the three configurations provided by the authors. The first of these (standard single item 6th order Markov model) produced the best results in all 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:

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

$r = \frac{|\{t \subseteq \mathit{TeDB}_T | t.pc = T\}|}{|\{t \subseteq \mathit{TeDB}_T\}|}$; $\quad f = \frac{2 * p * r}{(p+r)}$.

For multi-class classification, a different overall accuracy measurement is used: $a = \sum_{T_i} \frac{|\{t \subseteq \mathit{TeDB}_{T_i} | t.pc = t.c\}|}{|\mathit{TeDB}|}$. The accuracy for each target class T_i is calculated as: $a_i = \frac{|\{t \subseteq \mathit{TeDB}_{T_i} | t.pc = t.c\}|}{|\mathit{TeDB}_{T_i}|}$. 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 last row uses the log odd ratio score to classify the proteins[3]. The performance tells that simply adding up the score ratios from different classes does not give good answers. This partly shows the superiority of using the SVM to make the decisions.

Table 3. Results of the three binary-class experiments.

	OMP vs. Inn+Ext+Cyt				Extra vs. Intra				OMP vs. Globular			
Alg.	A%*	P%	R%	F%	A%	P%	R%	F%	A%	P%	R%	F%
g-MARS	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	34.10	42.74	**100**	59.89	97.40	65.20	39.40	36.25	90.00	88.55	77.34	82.27

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

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

Level-2 Subfamily	g-MARS	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

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 (Hight 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. 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. From both the tables we can also see that g-MARS performs generally better than all the other classifiers.

T-test. We conducted t-tests with a 95% confidence on the results. g-MARS wins **16** times, draws **9** times and loses **0** times. The Spectrum ranks the second best with **11** wins, **14** draws and **1** loses. The third best algorithm is the AAPC and the performance is **8** times winning, **14** times drawing and **3** times losing. From a statistical point of view, g-MARS wins the most which means it performs generally the best. Comparing directly against the Spectrum, g-MARS wins on 1 dataset and on the rest draws. It's running time is generally at least 10% faster than the Spectrum, even for high gaps.

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

Level-0 Subfamily	g-MARS	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	94.00
Class C Metabotropic glutamate/pheromone	**98.06**	95.15	91.26	76.21	97.09	93.69	82.70
Class D Fungal pheromone	89.23	86.15	81.54	89.23	83.07	**95.38**	76.20
Class E cAMP receptors	**100**	**100**	**100**	90	90	**100**	83.00
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

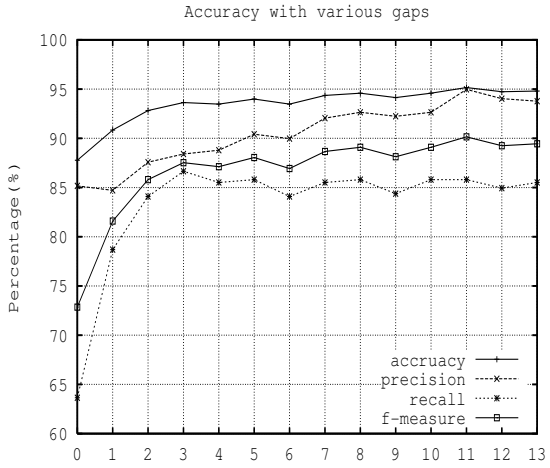

Fig. 1. g-MARS performance for varying gap on OMP vs. Inner, Extra, Cytoplasm dataset

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. 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, 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).

Running time . For classifiers working on large volumes of data, time efficiency is an important factor. We discussed the time complexity of g-MARS in Section 2.2. We also measured the running time for g-MARS on the OMP vs. Inner, Extra and Cyto-

plasm dataset with various gaps. The time includes the time 5-fold cross validation. and increases roughly linearly with the increment of the gap. For gap as 0, the executable time is less than 25 seconds and for the largest gap it only takes slightly more than 350 seconds, which is quite acceptable.

Conclusion and future work. 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. Compared to other work, it has the following merits: It is computationally efficient and can handle large volumes of proteins. It does not need prior knowledge to achieve good performance and can be generalized to any sequence classification problem. 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 superior accuracy for a range of protein datasets with diverse characteristics. Overall, g-MARS is a very practical algorithm to handle protein classification.

References

1. C. Chang and C. Lin. *LIBSVM: a library for support vector machines*, 2001.
2. 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.
3. R. Durbin, S. Eddy, A.Krogh, and Graeme Mitchison. *Biological sequence analysis—-Probabilistic models of proteins and nucleic acids.* Combridge University Press, 1998.
4. 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.
5. GPCRDB. http://www.gpcr.org.
6. M. Michael Gromiha and Makiko Suwa. A simple statistical method for discriminating outer membrane proteins with better accuracy. *Bioinformatics*, 21(7):961–968, 2005.
7. Dan Gusfield. *Algorithms on Strings, Trees, and Sequences - Computer Science and Computational Biology*. Cambridge University Press, 1997.
8. 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 *BIBE*, pages 113–120, 2005.
9. Tommi Jaakkola, Mark Diekhans, and David Haussler. Using the fisher kernel method to detect remote protein homologies. In *ISMB*, pages 149–158, 1999.
10. C. S. Leslie, E. Eskin, A. Cohen, J. Weston, and W. S. Noble. Mismatch string kernels for discriminative protein classification. *Bioinformatics*, 20(4), 2004.
11. C. S. Leslie, E. Eskin, and W. S. Noble. The spectrum kernel: A string kernel for svm protein classification. In *Pacific Symp. on Biocomputing*, pages 566–575, 2002.
12. Z. Liu. Predicting protein subcellular localization from homologs using machine learning algorithms. Master's thesis, Dept of Computer Science, University of Alberta, 2002.
13. 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 *KDD*, pages 436–445, 2003.
14. Junwen Wang and Sridhar Hannenhalli. Generalizations of markov model to characterize biological sequences. *BMC Bioinformatics*, 6(219), 2005.
15. 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.