REBMEC: Repeat Based Maximum Entropy Classifier for Biological Sequences

Pratibha Rani

Center for Data Engineering
International Institute of Information Technology
Hyderabad, India
pratibha_rani@research.iiit.ac.in

Vikram Pudi

Center for Data Engineering
International Institute of Information Technology
Hyderabad, India
vikram@iiit.ac.in

Abstract

An important problem in biological data analysis is to predict the family of a newly discovered sequence like a protein or DNA sequence, using the collection of available sequences. In this paper we tackle this problem and present REBMEC, a Repeat Based Maximum Entropy Classifier of biological sequences. Maximum entropy models are known to be theoretically robust and yield high accuracy, but are slow. This makes them useful as benchmarks to evaluate other classifiers. Specifically, REBMEC is based on the classical Generalized Iterative Scaling (GIS) algorithm and incorporates repeated occurrences of subsequences within each sequence. REBMEC uses maximal frequent subsequences as features but can support other types of features as well. Our extensive experiments on two collections of protein families show that REBMEC performs as well as existing state-of-the-art probabilistic classifiers for biological sequences without using domain-specific background knowledge such as multiple alignment, data transformation and complex feature extraction methods. The design of REBMEC is based on generic ideas that can apply to other domains where data is organized as collections of sequences.

1 Introduction

A critical problem in biological data analysis is to classify bio-sequences based on their important features and functions. This problem is important due to the exponential growth and accumulation of newly generated sequence data during recent years [36], which demands for automatic methods for sequence classification. Predicting the family of an unclassified sequence reduces the time and cost required for performing laboratory experiments to determine its properties like functions and structure because sequences belonging to the same family have similar characteristics.

International Conference on Management of Data
COMAD 2008, Mumbai, India, December 17–19, 2008

The known state-of-the-art solutions for this problem mainly use approaches such as Sequence Alignment [2, 22, 29], Hidden Markov Models (HMM) [11, 19], Probabilistic Suffix Trees (PST) [5, 12], and Support Vector Machines (SVM) [6, 21]. Recent approaches [24, 26] have been trying to improve SVM by incorporating domain knowledge, using complex features based on structures, and combining it with other classifiers.

In this paper, we propose a data mining based, simple but effective solution, which does not require domain knowledge, called *Repeat Based Maximum Entropy Classifier* (REBMEC)–a new maximum entropy based classifier which is able to incorporate *repeats* of subsequences within a sequence. REBMEC uses a novel framework based on the classical *Generalized Iterative Scaling* (GIS) [10] algorithm to find the maximum entropy model for a given collection of biological sequences.

Unlike other *Bayesian* classifiers like *Naive Bayes*, maximum entropy based classifiers do not assume independence among the features. These classifiers build the model of the dataset using an iterative approach to find the parameter values that satisfy the constraints generated by the features and the training data [35, 32, 9]. The maximum entropy principle has been widely used for discretization of numeric values of features [18], feature selection [34, 33], and various text related tasks like translation [7], document classification [27], and part-of-speech tagging [33]. Our approach is inspired by these works because comparison between biological sequence data and natural languages are commonplace [9]. REBMEC has the following desirable features:

1. It uses a novel framework based on GIS, to find the posterior probabilities, using the maximal frequent subsequences as features. In doing so it adapts GIS to deal with a large feature set.

2. It incorporates repeated occurrences of subsequences within each sequence of a family to compute feature probabilities.

3. It handles the problem associated with *Bayesian* clas-

sifiers due to a *nonuniform feature set* (where all features are not present in each class).

4. It uses an *entropy based feature selection* method to find the discriminating features for a class and to reduce the number of irrelevant features.

5. It is scalable with the database size as it does not compare the query sequences with the whole database.

6. It does not require domain knowledge based ideas such as the use of alignment based similarity (like in FASTA, BLAST and PSI-BLAST), complex feature extraction, or data transformation (like in SVM).

The remainder of this paper is organized as follows: Section 2 formally introduces the problem, and Section 3 introduces maximum entropy models. Section 4 provides critical definitions used in the paper and describes the methods for estimating feature probabilities. Section 5 presents the problem of *Bayesian* classifiers that arises when all features are not present in all classes. Section 6 describes the overall design of the proposed REBMEC classifier. Section 7 provides the dataset and implementation details. Section 8 describes all the experiments and results. Section 9 discusses these results. Section 10 presents the related work and Section 11 finally concludes our work.

2 Problem Definition

Given a training dataset $D = \{F_1, F_2, \ldots, F_n\}$ as a set of n families, where each *family*[1] is a collection of sequences, the goal of the classifier is to label a query sequence S with family F_i for which the posterior probability $P(F_i|S)$ is maximum. *Bayes formula* allows us to compute this probability from the prior probability $P(F_i)$ and the class-conditional probability $P(S|F_i)$ as follows:

$$P(F_i|S) = \frac{P(S|F_i)P(F_i)}{P(S)} \tag{1}$$

Since the evidence $P(S)$ is common for all families, it is ignored. $P(F_i)$ is the relative frequency of family F_i in D computed as $P(F_i) = \frac{N_i}{N}$, where N_i is the number of sequences in family F_i and N is the total number of sequences in the dataset. Hence the classification problem reduces to the correct estimation of $P(S|F_i)$, given the dataset D.

3 Maximum Entropy Models

The maximum entropy principle is well-accepted in the statistics community. It states that given a collection of known facts about a probability distribution, choose a model for this distribution that is consistent with all the facts, but otherwise is as uniform as possible. Hence, the chosen model does not assume any independence between its parameters that is not reflected in the given facts. It can be shown that there is always a unique, maximum entropy model that satisfies the constraints imposed by the training set and choice of features, and this model will have the exponential form [32]:

$$p(x) = \pi \prod_{i=1}^{n} (\mu_i)^{f_i(x)} \tag{2}$$

where $f_i(x) = 1$ if $f_i \subseteq x$ and 0 otherwise, and π is a normalization constant to guarantee that $\sum p(x) = 1$. Each parameter μ_i can be viewed as the weight of the feature f_i in the model.

In a maximum entropy based classifier, the estimation of class-conditional probabilities is done without assuming independence among the features. Once features have been selected, the task of modeling is reduced to finding the parameter values that satisfy the constraints generated by the selected features and the training data. The parameter values cannot be obtained directly and must be estimated by an iterative, hill-climbing method. Two such methods are available: the classical Generalized Iterative Scaling (GIS) [10] and Improved Iterative Scaling (IIS) [30].

3.1 Parameter Estimation

Both GIS and IIS algorithms require a "constraint set" CS on the data distribution X of the domain. This constraint set is generated by the selected features. Both algorithms begin with all parameters initialized to one, unless a previous solution for a similar set of features is available as a starting point. Both algorithms then update the parameters iteratively, each iteration resulting in a new probability distribution that is closer to satisfying the constraints imposed by the expectation values of the selected features. GIS requires that for any event the sum of features will be a constant. This requirement can be met for any set of features by introducing a "correction" feature f_l, where $l = |CS| + 1$, and adding it to the constraint set [32] such that:

$$f_l(x) = C - \sum_{j=1}^{|CS|} f_j(x) \tag{3}$$

The constant C is chosen to be equal to the maximum value for the sum of features 1 through $|CS|$, i.e., the size of the feature set available for the dataset [32, 23]. IIS differs from GIS in that it does not require model features to sum to a constant. But study of performances of iterative methods [23] shows that although IIS runs faster than GIS, the additional bookkeeping overhead required by IIS more than cancels any improvements in speed offered by accelerated convergence. Since both methods result in equivalent models, we chose to build our classification framework on GIS.

3.1.1 The GIS Algorithm

The GIS algorithm first initializes a parameter μ_j to '1' for each constraint and executes the following procedure until

[1] In this paper, the terms "family" and "class" are used interchangeably.

convergence:

$$\mu_j^{(n+1)} = \mu_j^{(n)} \left[\frac{P(f_j)}{P^{(n)}(f_j)} \right]^{\frac{1}{C}}$$

where

$$P^{(n)}(f_j) = \sum_{x \in X} p^{(n)}(x) f_j(x)$$

$$p^{(n)}(x) = \pi \prod_{j=1}^{l} (\mu_j^{(n)})^{f_j(x)}$$

$$C = \text{some constant}$$

$$f_j(x) = 1 \text{ if } f_j \subseteq x$$

$$= 0 \text{ otherwise}$$

The variable n in the above system of equations denotes the iteration number. $P^{(n)}(f_j)$ is the expected support of f_j in the nth iteration while $P(f_j)$ is the actual support of f_j calculated from the training dataset. Convergence is achieved when the expected and actual supports of every f_j are nearly equal. π is a normalization constant which ensures that $\sum_{x \in X} p(x) = 1$.

4 Estimating Feature Probabilities

In this section, we describe how feature probabilities may be estimated from the training data. We begin with a few definitions:

Definition 1. The *Sequence count* of a feature X_j in family F_i is the number of sequences of family F_i in which feature X_j is present at least once.

Definition 2. The *Repeat count* of a feature X_j in family F_i is the sum of the number of occurrences of that feature in each sequence of the family.

Definition 3. A feature X_j is *frequent* in family F_i *iff*

$$\textit{Sequence count of } X_j \text{ in } F_i \geq \sigma$$

where σ is the *Minsup count* for family F_i, and is calculated using the user given support threshold $minsup$ and N_i (total number of sequences in family F_i) as:

$$\sigma = N_i \times minsup$$

Definition 4. Let $\digamma = \{X_1, X_2, \ldots, X_{|\digamma|}\}$ be the set of all frequent features extracted from family F_i. Feature $X_j \in \digamma$ is *maximal frequent* in family F_i *iff*

$$\nexists X_k \in \digamma \text{ such that } X_k \supset X_j$$

Either *Sequence* or *Repeat count* may be used to estimate the probability $P(X_j|F_i)$ of a feature X_j in a family F_i.

Using *Sequence count* is simple: $\frac{Sequence\ count \text{ of } X_j}{N_i}$ is a good estimate of $P(X_j|F_i)$, where N_i is the number of sequences in F_i. Though this is simple and efficient, it does not account for multiple occurrences of X_j in a sequence.

The alternative is to use *Repeat count* that uses all the occurrences of a feature. We use following method proposed in [31] to find $P(X_j|F_i)$ using *Repeat count*:

1. Find the number of *slots* available for X_j in family F_i (containing N_i sequences).
   - If we consider that the features may overlap:

$$slots_{ij} = \sum_{k=1}^{N_i} \big[\text{length of } S_k - \text{length of } X_j + 1\big] \tag{4}$$

   - If we consider non overlapping features:

$$slots_{ij} = \sum_{k=1}^{N_i} \left\lfloor \frac{\text{length of } S_k}{\text{length of } X_j} \right\rfloor \tag{5}$$

2. Find the probability of feature X_j in family F_i as:

$$P(X_j|F_i) = \frac{\textit{Repeat count of } X_j \text{ in } F_i}{slots_{ij}} \tag{6}$$

Equations 4 and 5 find the total slots for feature X_j in family F_i by summing the available slots in each sequence S_k of F_i. Next, the feature probability is estimated as the fraction of times X_j actually occurs over the slots.

5 Problems with Bayesian Classifier

Bayesian classifiers like *Naive Bayes* (NB) represent the query sequence S as a feature vector $\overrightarrow{X} = \{X_1, X_2, \ldots, X_m\}$ and use the feature probabilities $P(X_j|F_i)$s to estimate the class-conditional probability $P(S|F_i)$. The class-conditional probability is used to compute the posterior probability of each family as:

$$P(F_i|S) \propto P(F_i)P(S|F_i) \tag{7}$$

5.1 Problem 1: Features Not Represented in the Training Data

Since calculation of $P(X_j|F_i)$ is based on the presence of X_j in the *training data* of class F_i, a problem can arise if X_j is completely absent in the training data of class F_i. The absence of X_j is quite common because training data is typically too small to be comprehensive, and not because $P(X_j|F_i)$ is really zero. This problem is compounded by the resulting zero probability for any sequence S that contains X_j. Evidence based on other subsequences of S may point to a significant presence of S in F_i. Due to this problem, the existing Bayesian formulation described in Equation 7 cannot be applied directly on biological sequences when frequent subsequences are used as features. Known solutions to this problem are:

1. Use a *nonuniform* feature vector, i.e., use different feature vectors of query sequence S for each class which include only those features of S which are present in that class. Then set $P(S|F_i) = 0$ only if *none* of the features of S is present in class F_i.

   This solution has a drawback: classes with *more* matching features of S could be computed as having *less* posterior probability due to the multiplication

of more feature probabilities whose values are always less than one. This results in wrong classification and is illustrated in Example 1 shown in Figure 1.

Example 1. Suppose C_1 and C_2 are two classes with 10 samples each, so that the prior probabilities of the classes are $P(C_1) = P(C_2) = \frac{1}{2}$. A query sample S with feature vector $\{X_1, X_2, X_3, X_4\}$ has two matching features in class C_1 with probabilities

$$P(X_1|C_1) = \frac{1}{10} \text{ and } P(X_3|C_1) = \frac{3}{10}$$

and four matching features in class C_2 with probabilities

$$P(X_1|C_2) = \frac{1}{10}, P(X_2|C_2) = \frac{2}{10},$$

$$P(X_3|C_2) = \frac{3}{10} \text{ and } P(X_4|C_2) = \frac{2}{10}$$

Using Equation 7, the posterior probabilities of the classes are obtained as

$$P(C_1|S) = \frac{3}{200} \text{ and } P(C_2|S) = \frac{6}{10000}$$

Since $P(C_1|S) > P(C_2|S)$, the query sample gets classified into class C_1, although intuitively we know that class C_2 is more suitable because it contains more matching features than class C_1.

Figure 1: An example showing the drawback of using a non-uniform feature vector.

2. Incorporate a small sample-correction into all probabilities, such as *Laplace correction* [18]. The *Laplace correction* factor requires changing all the probability values, so it is not feasible for datasets with a large feature set like biological sequence datasets.

3. If a feature value does not occur in a given class, then set its probability to $\frac{1}{N}$, where N is the number of examples in the training set [18].

The *Simple Naive Bayes* (Simple NB) classifier uses *Sequence count* with solution (1) to obtain the model of the dataset. In REBMEC we use a different solution proposed in [31] and described in Section 6.3, which outperformed the Simple NB classifier.

5.2 Problem 2: Out of Range Probability Values

Probability values obtained using equations such as Equations 6 and 8 are very small. When these very small values are multiplied to obtain the class-conditional probability to be used in Equation 7, the product can go below the available minimum number range of the processor. This is a problem with all *Bayesian* classifiers which work with large feature sets and assume independence among the features and hence directly multiply the feature probabilities to get the class-conditional probabilities. An appropriate scaling factor or log scaled formulation is used to avoid this problem.

We used log scaled formulation to avoid this problem for the Simple NB classifier. The proposed classification algorithm of REBMEC implicitly uses a log scaled approach for finding class-conditional probabilities that can handle small probability values and hence easily deals with this problem.

6 The REBMEC Classifier

Like all *Bayesian* classifiers, REBMEC also works in two phases: training phase and classification phase. It also uses a feature selection phase as part of the training phase to select important features and prune irrelevant features. So, the REBMEC classifier runs in three phases:

1. **Feature Extraction**: This is the training phase in which first *maximal frequent subsequences* are extracted as features from each family and stored with their *Repeat* and *Sequence counts*. Then for each family, the *Repeat* and *Sequence counts* for maximal features from other families, which are not maximal in this family, are also stored. This is to ensure that all families share the same feature set.

2. **Feature Selection**: The extracted feature set is pruned in this phase using an *entropy based selection criterion*. The result is a smaller set of features and their *Repeat* and *Sequence counts* within each family. The feature extraction and selection phases are executed only once to train the classifier. After this the original dataset is no longer required and the classifier works with the reduced feature set left after pruning.

3. **Classification**: This phase is executed for labeling a query sequence with the family having the highest posterior probability. The classifier first separates all the features belonging to the query sequence from the available feature set from the second phase, to make a *query-based uniform feature set*. It then uses this uniform feature set to find the posterior probability of each family and outputs the one with the highest posterior probability.

6.1 Feature Extraction

Any kind of features can be supported in the classification algorithm of REBMEC, but we have used *maximal frequent subsequences* as features. Use of these simple features avoids the need for complex data transformations and domain knowledge which is required by the existing sophisticated feature mining algorithms [14, 20] for biological sequences. It was observed in [31] that frequent subsequences capture everything that is significant in a collection of sequences. But to keep the feature set small and to satisfy the following necessary criteria for features of any classifier [20], using maximal frequent subsequences as features is a better option [31]:

1. Significant features: We ensure this by considering only *frequent* features (i.e., $Sequence\ count \geq Minsup\ count$).

2. Non-redundant Features: We ensure this by using *maximal* frequent subsequences.

3. Discriminative Features: For ensuring this, we use the *entropy* based selection criteria described in Section 6.2 after extraction of features.

We extracted maximal frequent subsequences as features using an *Apriori*-like method which uses the same user given minimum support threshold *minsup* for all families. We extracted all possible features from a family by setting the maximum length of the feature to be extracted as the length of the largest sequence of the training dataset.

Frequent subsequence extraction from a biological sequence dataset is a time consuming and memory intensive process. We optimized this process by avoiding extraction of infrequent subsequences by storing information of their location in a bit-vector. This *bit-vector based optimization of frequent subsequence extraction* method, proposed in [31], is explained below.

This method is based on the idea that the presence of a '1' in a bit-vector indicates that a frequent subsequence of length l can be extracted from the corresponding position in the sequence. The presence of a '0' indicates that the subsequence of length l at the corresponding position in the sequence is *infrequent*. It follows that subsequences longer than l from this position will also be infrequent. Hence the bit will remain '0'.

So, after initializing a bit-vector of '1's for each sequence in a family, which is of the same length as the sequence, the procedure starts extracting frequent subsequences of length one and iteratively proceeds to longer subsequences. In the first phase of each iteration, candidate subsequences of length l are counted. In the second phase, the bit positions corresponding to frequent subsequences of length l are set to '1', to be considered in the next iteration.

6.2 Entropy Based Selection of Discriminating Features

As is typical of frequent pattern mining, the feature extraction phase produces too many features and creates the problem of *curse of dimensionality*. This problem increases as the $minsup$ decreases, since the number of extracted features increases exponentially as $minsup$ decreases. We alleviate this problem by applying a feature selection phase that selects only *discriminating* features [20] for each class.

Our feature selection criterion is based on entropy. Entropy based criteria like *information gain* and *gain ratio* have been widely used to select features for classifiers [18]. Since our aim is to find discriminating features for each family, we use low values of $H(D|X_j = present)$, i.e., *entropy of the dataset in the presence of a feature* as the selection criterion:

$$H(D|X_j = present) = -\sum_{i=1}^{N} \big[P(F_i|X_j = present) \times \log[P(F_i|X_j = present)]\big]$$

where

$$P(F_i|X_j = present) = \frac{\textit{Sequence count} \text{ of } X_j \text{ in } F_i}{\sum_k \textit{Sequence count} \text{ of } X_j \text{ in } F_k}$$

Analysis of this criterion gives us the following observations:

1. $H(D|X_j = present) = 0$ when a feature X_j is present in one and only one family.

2. $H(D|X_j = present)$ is higher when a feature X_j is present in all families.

This criteria is opposite of the *information gain* criteria as it selects features with low entropy values thereby selecting discriminating features. For selecting features we use a user-given threshold H_{th} to compare with the calculated value of $H(D|X_j = present)$, and select all the features satisfying the criteria $H(D|X_j = present) \leq H_{th}$ while pruning the others.

Experimentally we found that for very low *minsup* values, using threshold $H_{th} = 0$ gives good results in the classification phase. But for other *minsup* values good results are obtained by setting H_{th} as $\frac{1}{2}H(D)$ or $\frac{1}{3}H(D)$, where $H(D)$ is the *total entropy of the dataset* which is defined as:

$$H(D) = -\sum_i P(F_i) \log[P(F_i)]$$

This happens because with $H_{th} = 0$, many important features get pruned. In our experiments, the above *entropy based selection* not only found discriminating features for all families, but also reduced the number of features by 36% for low $minsup$ values, as shown in Table 1.

6.3 Classification Phase

REBMEC uses a very simple assumption to handle the problem of *zero probabilities* and the problem arising from the use of a *nonuniform feature set* (discussed in Section 5.1). It assumes that the probability of any feature to be present in any family *is never zero*. So for the features of other families which are not present in a given family, it uses a correction probability ϵ, which is the *minimum possible probability computed using Repeat counts* for a feature. It is obtained as:

$$\epsilon = \frac{1}{\text{Sum of the lengths of sequences of the largest family}} \quad (8)$$

6.3.1 Classification Algorithm of REBMEC

For classifying a query sequence S, REBMEC finds the uniform feature set F, which is the set of features present in S, collected from all families. It uses Equation 6 for finding probabilities of features present in a family and uses correction probability ϵ as the probability of features not present in that family.

It then uses the features in set F to make the constraint set for each family. Each feature X_j with its probability $P(X_j|F_i)$ in family F_i forms a *constraint* that needs to be satisfied by the statistical model for that particular family. Also, the "correction" feature f_l with expectation value $E(f_l)$ obtained using Equation 3 is added to this set. Note that unlike other feature probability values, this value ranges from 0 to C. Thus, for each family F_i, there is a constraint set $CS_i = \{(X_j, P(X_j|F_i)|X_j \in F) \cup (f_l, E(f_l))\}$. Since there could be multiple models satisfying these constraints, the proposed algorithm *ComputeProb*, like GIS, selects the one with *maximum entropy* and finds the parameters of that model. In doing so, it finds the class-conditional probability $P(S|F_i)$ of that family.

REBMEC then finds the posterior probability of all families using the log scale version of Equation 7 as follows:

$$LP(F_i|S) = LP(S|F_i) + LP(F_i) \tag{9}$$

$$\begin{aligned} \text{where } LP(F_i|S) &= \log[P(F_i|S)] \\ LP(S|F_i) &= \log[P(S|F_i)] \\ LP(F_i) &= \log[P(F_i)] \end{aligned}$$

Finally, it classifies the query sequence into the family with the largest posterior probability. The pseudo-code for the method discussed above is shown in Figure 2.

1. Find features from all families which are present in the query sequence S and make *uniform feature set* $F = \{X_1, X_2, \ldots, X_m\}$.
2. **for each** family F_i do
   (a) **for each** feature $X_j \in F$ do
   if $X_j \in F_i$: compute $P(X_j|F_i)$ using Equation 6
   else: set $P(X_j|F_i) = \epsilon$
3. Run **ComputeProb** to obtain all $LP(S|F_i)$s.
4. Compute all $LP(F_i|S)$s using Equation 9.
5. Find family F_k having the largest value of $LP(F_i|S)$.
6. Classify S into F_k.

Figure 2: Classification algorithm of REBMEC.

6.3.2 The ComputeProb Algorithm

ComputeProb is a GIS based method which, unlike GIS, computes the class-conditional probabilities instead of storing the parameter values of each constraint. Figure 3 shows pseudo-code of the *ComputeProb* algorithm and is described below.

To make the large feature set manageable, it divides the feature set F into small sets using *Hamdis* as the similarity measure to find the set of similar features. This similarity measure is a simple modification of the *Hamming Distance* to take into account features of different lengths and is defined as:

$$\begin{aligned} Hamdis(X_1, X_2) = &\text{ No. of positions differing in symbols} \\ &+ |\text{length}(X_1) - \text{length}(X_2)| \end{aligned}$$

The algorithm selects one feature from F and calculates $Hamdis$ for all other features with respect to the selected feature. Then it groups k features with least $Hamdis$, together with the selected feature to make the small feature set F_f. This process is repeated till there are less than k features left in F, which are grouped together along with the *correction* feature.

1. Using $Hamdis$ divide the feature set F into small sets as $F' = \{F_1, F_2, \ldots, F_M\}$, such that $\bigcup_i F_i = F$ and $F_i \cap F_j = \phi$
2. **for each** family F_i :
3. Initialize $LP(S|F_i) = 0$ *#Class conditional Probability*
4. **for each** small feature set $F_f \in F'$:
5. Set $last = \left(2^{|F_f|}\right) - 1$
6. **for** $k = 0$ to $last$:
7. Initialize $LP(T_k) = \log\left(\frac{1}{last+1}\right)$
8. **for each** feature $X_j \in F_f$:
9. Initialize $\mu_j = 0$
10. **while** all constraints are not satisfied
11. {
12. **for each** feature $X_j \in F_f$:
13. Initialize $Sum_j = 0$
14. **for** $k = 0$ to $last$:
15. **if** T_k contains X_j :
16. Update $Sum_j = Sum_j + \exp(LP(T_k))$
17. Set $LP(X_j|F_i) = \log(P(X_j|F_i))$
18. Update $\mu_j = \mu_j + LP(X_j|F_i) - \log(Sum_j)$
19. **for** $k = 0$ to $last$:
20. **for each** feature $X_j \in F_f$:
21. **if** T_k contains X_j :
22. Update $LP(T_k) = LP(T_k) + \mu_j$
23. Initialize $normSum = 0$
24. **for** $k = 0$ to $last$:
25. Update $normSum = normSum + \exp(LP(T_k))$
26. Set $\mu_0 = \frac{1}{normSum}$ *#Normalization factor*
27. **for** $k = 0$ to $last$:
28. Update $LP(T_k) = LP(T_k) + \log(\mu_0)$
29. }
30. Update $LP(S|F_i) = LP(S|F_i) + LP(T_{last})$
31. Return $LP(S|F_i)$

Figure 3: The ComputeProb algorithm.

For computing the class-conditional probability of a family, *ComputeProb* finds $LP(S|F_i)$ for each small set of features and later combines them by assuming independence among the sets. It uses bit-vectors T_k to represent the presence/absence of $|F_f|$ features of the set F_f. Specifically, in steps 6 to 9, it initializes the parameters μ_j and probabilities $LP(T_k)$. In steps 12 to 22, it updates the μ_j and $LP(T_k)$ values using the probabilities of features obtained from the training data. In steps 23 to 28, it finds the normalization constant μ_0 and applies it to the $LP(T_k)$ values. Finally, in step 30, it updates the $LP(S|F_i)$ value using the obtained value of $LP(T_{last})$ for that feature set, where the bit-vector T_{last} represents that all features of the set F_f are present in the query sequence S. Note that the $LP(S|F_i)$ values returned by this algorithm are in log scale. Also note that during implementation T_ks need not be stored but can be computed on the fly.

6.3.3 Additional Issues

Calculation of the expectation $E(F_l)$ of the correction feature f_l in a family F_i, using Equation 3, requires scanning all the sequences of that family. We observe that, using Equation 3, the minimum calculated value of $E(f_l)$ in a family will be $(C-|F|)$ when all the features of the feature set F are present in all the sequences of that family. And the maximum calculated value of $E(f_l)$ in a family will be C when none of the features of the feature set F are present in any sequence of that family. Based on these observations we used the minimum expectation value $(C - |F|)$ as the approximate expectation value of the correction feature f_l in each family. This removed the need for scanning all the sequences of a family in the classification phase for calculating the expectation of f_l. In practice we found this approximation to be good.

In our experiments we observed that if the correction feature f_l is not added to the constraint set with proper expectation value, then the algorithm is not able to compute correct class conditional probabilities; so using the correction feature properly is a very important part of the algorithm. We also observed that the correction feature can be added either to each small group of features F_f with approximate expectation value $(|F| - |F_f|)$ or only to one group with value $(C - |F|)$. Both methods give exactly the same results which means that both methods produce the same effect on the parameter values. Since adding the correction feature to each group of features increases the overall running time, it is better to add it to only one group with appropriate expectation value.

Like other *Bayesian* methods, GIS based methods also deal with very small parameter and probability values, so they also need to tackle "out of range" parameter values discussed in section 5.2. In case of GIS based methods, this problem becomes even more serious due to the iterative nature of these methods. To deal with it, we have designed *ComputeProb* using log scale. In our experiments we observed that due to the large value of the constant C, the iteration process overshoots the convergence point; so to make the increment value smaller for each iteration, we have not used constant C in the calculation of increment value (in step 18). As discussed in [33] and observed in our experiments the number of iterations required for the model to converge can be hard-coded and the algorithm can be made to stop once it reaches those many iterations, so the steps 10 to 29 are iterated for a fixed number of times.

7 Performance Model

In this section, we describe the datasets and the necessary implementation details. We have used two collections of protein families to evaluate the performance of REBMEC. The first collection is a large collection of 10922 *G protein-coupled receptors* (GPCRs) taken from the March-2005 Release 9.0 of GPCRDB [15] (http://www.gpcr.org/7tm). Since the GPCR families are divided in hierarchies of subfamilies, we have used the sequences of the highest level of families called the super families. After the removal of unclassified and redundant sequences, the dataset consists of 8435 sequences arranged in 13 families. This collection is a skewed dataset which has the peculiar property that the largest family called *class A* contains 6031 sequences which is 71% of the total sequences of the dataset.

The second collection of 3146 proteins arranged in 26 families was obtained from the Feb-2008 Release 55.0 of SWISS-PROT [8] using the list of SWISS-PROT protein IDs obtained from Pfam [4] version 22.0. After the redundant sequences were removed from each family, the number of sequences were down to 2097. This set of families has already been used in [5, 12, 13], so using it allows a direct comparison with their methods. We should note that, due to the constant refinement in the topology of the Pfam database, there are significant differences in the families common to the two collections.

All the classifiers were assessed based on the standard *Accuracy* measure, which gives the percentage of correctly classified test sequences:

$$Accuracy = \frac{\textit{NumCorrect}}{\textit{NumTested}} \times 100\%$$

All the experiments were performed on a 1.9 GHz AMD Athlon 64 processor machine with 385 MB of main memory, running Linux Fedora Core 4. All the programs were written in the Perl language.

For comparison with the *Naive Bayes* classifier, we implemented the Simple NB classifier discussed in Section 5. For comparison with the maximum entropy classifier, which uses *Sequence count*, we implemented the Maxent-A model which exactly follows all the steps used in REBMEC classifier and uses the same classification algorithm but uses *Sequence count* to find the probability of a feature. It uses the correction probability $\epsilon = \frac{1}{\text{No. of sequences of the largest family}}$ for the features not present in a family. In all experiments we used the same set of features for these classifiers that we used for the REBMEC classifier.

Based on our observation of convergence of the μ values, we fixed the number of iterations as 50 and the number of features in each group as 10 for implementing the *ComputeProb* algorithm.

For finding the accuracy of the classifiers, we divided the dataset into *training* and *test* sets in the ratio 9:1 with *stratification* (to include representatives from all families in the test set in their proper proportion), but to keep the test set small we included at most 50 sequences from each family. We used the same *minsup* for all families of a dataset to extract the maximal frequent subsequences from the training set. We experimented with different *minsup* values and found the best support value for each classifier according to the accuracy achieved with features obtained from that support value.

We did not put any restriction on the maximum length of maximal frequent subsequence to be extracted. We set it as the length of the largest sequence of the dataset. This

enabled us to extract all the possible maximal frequent subsequences from the dataset. Our experiments showed that the extraction process terminates long before reaching the maximum length, because as the subsequence length increases, its possibility of being frequent decreases. We used maximal subsequences of length greater than two as features, to avoid trivial frequent subsequences of lengths one and two.

8 Experimental Results

In this section we present the results. Table 1 gives the number of features used in the experiments for different $minsup$ values. The *Without Pruning* column gives the number of features when the feature set is not pruned and the *With Pruning* column gives the number of features remaining in the feature set after pruning them on the basis of *entropy*. For 5% $minsup$ we used threshold $H_{th} = 0.0$ and for other $minsup$ values we used $\frac{1}{2}$(*entropy of the dataset*) as threshold to prune the features. In all the experiments we used the features remaining in the feature set after pruning.

Table 1: Number of features used in experiments with the Pfam dataset for different $minsup$ values

$minsup$ (%)	Without Pruning	With Pruning
5	24005	15381
10	10743	6740
30	2627	2285
50	1131	1100

Table 2: Classification results of Simple NB, Maxent-A and REBMEC for the collection of 211 proteins of 26 families from Pfam 22.0 for different $minsup$ values

Classifier	$minsup$ (%)	Accuracy (%)
Simple NB	5	0.47
	10	0.0
	30	2.84
	50	**10.43**
Maxent-A	5	**66.82**
	10	48.34
	30	23.22
	50	16.11
REBMEC	5	**90.52**
	10	84.36
	30	77.73
	50	69.19

Table 2 summarizes the results of experiments done on the classifiers with different $minsup$ values.

Tables 3 and 4 show the family-wise and average accuracies of the classifiers with corresponding minimum support values. The *Size* column of Table 3 and the *Non Redundant Size* column of Table 4 give the number of sequences of each family after removing duplicated sequences from the family (before dividing it into training and test sets). In Table 4 we have included the published results of the C classifier of [13] and the results of the matching families of the PST classifier published in [5]. Please note that the test sets and evaluation methods used by them are different from ours and so the results should not be compared strictly. Due to the changes in the naming convention of Pfam families we were not able to find results for some families from the published results of PST classifier and hence we have not reported the average accuracy of the PST classifier.

9 Discussion

In this section we discuss the merits of REBMEC as seen in the experimental results.

From the results of Table 3, it is evident that the Simple NB classifier is biased towards the largest family and in its presence it is not able to recognize sequences of other families. But REBMEC is able to break the biasing effect of large classes and so performs very well on the skewed dataset also. The results indicate that there are some families like *Class E* and *Ocular albinism proteins* for which all the classifiers perform the same. Observation of the feature set shows that these families have many discriminative features. And there are some families like *Class A* and *Insect odorant receptors* which have very few discriminative features. Only REBMEC gives acceptable performance for these families.

The results of Table 2 clearly show that REBMEC performs much better than the simple NB classifier and Maxent-A model for all $minsup$ values. This indicates that REBMEC can perform well on datasets with large feature sets, whereas the other *Bayesian* classifiers can not.

As the $minsup$ value decreases, the performance of REBMEC improves and is highest at 05% $minsup$. In case of the Simple NB classifier, performance improves as $minsup$ increases because the number of features decreases. The Maxent-A classifier performs better than the Simple NB classifier and its performance increases as $minsup$ decreases, but it never gives accuracy comparable to the accuracy of REBMEC.

Since the Maxent-A model also uses the same framework used by REBMEC but with *Sequence count*, the performance of REBMEC indicates that use of *Repeat count* is able to model the families in a better way than *Sequence count*. This confirms the observation made in [31]. Hence use of *Repeat count* is recommended over *Sequence count* to obtain a good classification model for sequence data like biological sequences.

Results presented in Table 4 on the Pfam dataset indicate that performance of REBMEC is comparable to the performances of C classifier of [13] and PST [5]. It performs better than the C classifier for large families with more number of sequences. Clearly the PST classifier is the consistent performer among the three classifiers, whose performance is almost the same for all families. The fea-

Table 3: Family-wise and Average Classification results of Simple NB, Maxent-A and REBMEC for the collection of 265 GPCRs of 13 families from GPCRDB

Family Name	Size	Simple NB (%)	Maxent-A (%)	REBMEC (%)
Class A	6031	74.0	52.0	90.0
Class B	309	0.0	96.7	96.7
Class C	206	0.0	100.0	100.0
Class D	65	0.0	66.7	50.0
Class E	10	0.0	100.0	100.0
Ocular albinism proteins	8	0.0	100.0	100.0
Frizzled family	130	0.0	92.3	92.31
Insect odorant receptors	236	0.0	4.2	62.5
Plant Mlo receptors	52	0.0	100.0	100.0
Nematode chemoreceptors	755	0.0	44.0	100.0
Vomeronasal receptors	286	10.3	82.7	96.5
Taste receptors T2R	237	0.0	87.5	91.7
Class Z Archaeal opsins	110	9.0	100.0	100.0
Average Accuracy (%)		15.47	67.17	91.7
$minsup$ (%)		50	5	10

Table 4: Family-wise and Average Classification results of Simple NB, Maxent-A, REBMEC, Ferreira et al.'s C model and PST for the collection of 211 proteins of 26 families from Pfam 22.0

Family Name	Actual Size	Non Redundant Size	Simple NB (%)	Maxent-A (%)	REBMEC (%)	C (%)	PST (%)
7tm-1	64	64	0.0	33.3	100.0	100.0	93.0
7tm-2	32	32	0.0	66.7	66.7	100.0	94.4
7tm-3	29	29	0.0	66.7	66.7	100.0	83.3
AAA	229	210	28.57	76.2	100.0	97.7	87.9
ABC-tran	63	57	0.0	33.3	83.3	100.0	83.6
ATP-synt-ab	151	151	0.0	86.7	100.0	98.9	96.7
ATP-synt-A	30	30	0.0	33.3	66.7	89.3	92.4
ATP-synt-C	35	33	0.0	66.7	66.7	97.0	96.7
c2	394	286	0.0	55.2	82.76	91.3	92.3
CLP-protease	88	88	0.0	77.8	88.9	94.1	87.9
COesterase	126	126	0.0	100.0	100.0	87.3	91.7
cox1	23	23	0.0	100.0	100.0	90.9	83.8
cox2	32	32	0.0	100.0	100.0	100.0	98.2
Cys-knot	24	24	0.0	100.0	100.0	100.0	93.4
Cytochrom-B-C	9	9	0.0	0.0	100.0	100.0	79.2
Cytochrom-B-N	8	8	0.0	100.0	100.0	85.7	98.2
HCV-NS1	10	10	0.0	100.0	100.0	100.0	–
Oxidored-q1	33	33	0.0	100.0	100.0	93.5	97.1
PKinase	54	54	0.0	40.0	**60.0**	**96.2**	**85.1**
PPR	558	100	0.0	90.0	**100.0**	**22.6**	–
RuBisCO-large	16	16	0.0	100.0	100.0	81.3	98.7
rvt-1	156	156	0.0	100.0	**100.0**	**79.6**	**88.4**
RVT-thumb	41	41	0.0	100.0	100.0	91.2	–
TPR-1	562	274	55.17	24.1	**93.1**	**54.9**	–
zf-C2H2	196	101	0.0	60.0	70.0	88.4	92.3
zf-CCHC	183	110	0.0	63.6	81.82	100.0	88.6
Accuracy (%)			10.43	66.82	90.52	90.0	–
$minsup$ (%)			50	5	5	–	–

ture extraction process of each family is fine tuned in [13] by using different *minsup* values for different families and a sophisticated feature extraction algorithm [14] is used. We note that REBMEC attains comparable performance without using domain specific ideas and incorporation of such ideas could further increase the performance of REBMEC.

The families of the Pfam dataset were constructed on the basis of sequence similarity using HMM [4], while the families of GPCRDB were constructed manually on the basis of the function of proteins [15]; so similarity among sequences of a family of the Pfam dataset is higher than the GPCRDB dataset. Due to this, classifiers like PSTs which use *Variable Length Markov Model* perform better on the Pfam dataset. Comparing results of Tables 3 and 4 shows that although the GPCRDB dataset is larger in size and skewed than the Pfam dataset, REBMEC performs better on the GPCRDB dataset than the Pfam dataset. So REBMEC can be better than other sequence classifiers for large and skewed datasets with less similarity among sequences.

Note that we have used a very naive approach to find similarity between features because we wanted to keep the algorithm simple. The performance of REBMEC will increase if more sophisticated similarity methods based on domain knowledge are used to choose features of a group. We also experimented with grouping features randomly and found that results are the same. This indicates that the features are almost independent of each other.

The *Bayesian* classifier of [3] requires the feature length to be supplied by the user. Other classifiers like PSTs, SMTs and the C classifier also require parameters other than $minsup$ to be supplied by the user for the feature extraction process. It is observed that performance of the classifiers are very sensitive towards these parameters. REBMEC does not require feature length or any parameter other than $minsup$ for the feature extraction process and the parameter required for pruning can be set easily according to the *entropy of the dataset* as described in Section 6.2.

***Scalability and Computational Requirements*:** The classification time of REBMEC and Maxent-A is much larger than the other classifiers. For example, REBMEC takes more than an hour to classify each test sequence whereas the other classifiers take only a fraction of second or at most a few minutes. The reason for this is the iterative nature of the GIS-based procedure. However, the advantage of REBMEC is that it is a theoretically robust method that makes no assumptions about the underlying data distribution. Hence REBMEC can be used as a good benchmark to compare other classifiers. The memory consumption of REBMEC in the classification phase is also quite low because it does not store bit-vectors T_k, and instead computes them on the fly (see Section 6.3.2) This is unlike the HMMs, PSTs and SVM based classifiers which require large memory.

REBMEC has the advantage of very low computational requirements for the training phase as this phase involves only finding *counts* of features in each family. HMMs, PSTs, SMTs and SVM based classifiers have high memory requirements for the training phase and become computationally very expensive when the number of classes increases [5, 12]. Another point worth mentioning is that SVM based classifiers require carefully selected positive and negative samples for each class in the training phase while REBMEC uses only positive samples.

10 Related Work

The principle of maximum entropy has been widely used for a variety of natural language processing tasks like translation [7], document classification [27], identifying sentence boundaries [33], prepositional phrase attachment [33], part-of-speech tagging [33], and ambiguity resolution [33]. The authors of [25] study the multinomial models for document classification, which capture the word frequency information in each document, and show that these models perform better than the multi-variate Bernoulli model. Use of *Repeat count* in REBMEC is similar to the multinomial models but it follows a very different approach for finding the feature probabilities which are used to build the model. Basically, the multinomial models represent training samples as bag of words whereas in REBMEC we model them as sequences.

The authors of [35] present the ACME classifier, which uses maximum entropy modeling for categorical data classification with mined frequent itemsets as constraints. The authors also propose the approach of dividing the feature set into independent clusters of features for reducing the computational cost.

A sequence modeling method using mixtures of conditional maximum entropy distributions is presented in [28]. This method generalizes the mixture of first order Markov models by including the long term dependencies, known as triggers, in the model. [9] uses maximum entropy for modeling protein sequence, using unigram, bigram, unigram cache and class based self triggers as features. Both [28, 9] use the history of symbols to predict the next symbol of a sequence.

Examples of ***Bayesian Sequence Classifiers*** can be found in [3, 13, 16, 17]. The authors of [3] propose that the NB classifier can be used for protein classification by representing protein sequences as class conditional probability distribution of *k-grams* (short subsequences of amino acids of length *k*). The length of *k-grams* used in this method is a user supplied parameter and the performance of the classifier is very sensitive towards this parameter.

The approach used in [13] is to use the unlabeled sequence to find rigid gap frequent subsequences of a certain minimum length and use these to obtain two features which are combined in the NB classifier. This approach uses a computationally expensive subsequence extraction method [14] which is guided by many user supplied parameters. This method defers the feature extraction phase till classification time which makes it computationally expensive for large datasets.

The authors of [16] introduce a novel word taxonomy based NB learner (WTNBL-ML) for text and sequences.

This classifier requires a similarity measure for words to build the word taxonomy. The authors of [17] present a promising recursive NB classifier RNBL–MN, which constructs a tree of Naive Bayes classifiers for sequence classification, where each individual NB classifier in the tree is based on a multinomial event model (one for each class at each node in the tree).

Probabilistic Suffix Tree based Sequence Classifiers [5, 12] predict the next symbol in a sequence based on the previous symbols. Basically a PST [5] is a variable length Markov Model, where the probability of a symbol in a sequence depends on the previous symbols. The conditional probabilities of the symbols used in PSTs rely on exact subsequence matches, which becomes a limitation, since substitutions of symbols by equivalent ones is often very frequent in proteins. The proposed classifier of [12] tries to overcome this limitation by generalizing PSTs to SMTs with wild-card support, which is a symbol that denotes a gap of size one and matches any symbol on the alphabet. An experimental evaluation in [5] shows that PSTs perform much better than a typical PSI-BLAST search and as well as HMM. As bio-sequence databases are becoming larger and larger, data driven learning algorithms for PSTs or SMTs will require vast amounts of memory.

HMM based Sequence Classifiers [11, 19] use HMM to build a model for each protein family based on multiple alignment of sequences. HMM models suffer from known *learnability hardness results* [1], exponential growth in number of states and in practice, require a high quality multiple alignment of the input sequences to obtain a reliable model. The HMM based classifiers use algorithms which are very complex to implement and the models they generate tend to be space inefficient and require large memory.

Similarity based Sequence Classifiers [2, 22, 29] compare an unlabeled sequence with all the sequences of the database and assess sequence similarity using sequence alignment methods like FASTA [29], BLAST [22] or PSI-BLAST [2] and use *K nearest neighbor approach* to classify the sequence. But as the number of sequences in biodatabases is increasing exponentially, this method is infeasible due to the increased time required to align the new sequence with the whole database.

SVM based Sequence Classifiers [6, 21, 24, 26] either use a set of features of protein families to train SVM or use *string kernel* based SVMs, alone or with some standard similarity measure like BLAST or PSI-BLAST or with some structural information. These classifiers require a lot of data transformation but report the best accuracies. Since SVM is basically a binary classifier, to handle a large number of classes, it uses the *one against the rest* method, which becomes computationally very expensive as the number of classes increases.

11 Conclusions and Future Work

In this paper, we proposed a new maximum entropy based classifier, REBMEC, which uses a GIS based novel framework to build the classification model and incorporates *Repeat count* of features present in the query sequence. It adapts GIS to deal with a large feature set. It uses the simplest possible features of sequence families in the form of maximal frequent subsequences and handles the problem associated with Bayesian classifiers. We also discussed the issues of inclusion of the correction feature and calculation of increment values, which need to be handled properly in GIS-based methods.

By experiments on two datasets, we demonstrated that REBMEC's performance is comparable to the PST and the C classifier, and is superior to the other Bayesian classifiers on large and skewed datasets. REBMEC has the advantage of very low computational requirements as it does not use any complex feature extraction method, domain knowledge or data transformation. It does not require any user supplied parameters which can affect the performance other than $minsup$ and the pruning $threshold$. We demonstrated that the use of *Repeat count* gives a better model of sequence families than *Sequence count*.

Future work includes (1) finding good similarity measures and methods to divide features in small sets with minimum or no dependence among the features of different sets, (2) evaluating REBMEC's performance with sophisticated feature extraction methods which can extract gapped subsequences supporting wild-cards, and (3) evaluating REBMEC's performance on other domains which have data in sequential form.

References

[1] N. Abe and M. K. Warmuth. On the computational complexity of approximating distributions by probabilistic automata. *Machine Learning*, 9:205–260, 1992.

[2] S. F. Altschul et al. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Research*, 25:3389–3402, 1997.

[3] C. Andorf, A. Silvescu, D. Dobbs, and V. Honavar. Learning classifiers for assigning protein sequences to gene ontology functional families. In *Proc. of the Fifth Int. Conf. on KBCS*, pages 256–265, 2004.

[4] A. Bateman et al. The Pfam protein families database. *Nucleic Acids Research*, 36(Database-Issue):281–288, 2008.

[5] G. Bejerano and G. Yona. Modeling protein families using probabilistic suffix trees. In *Proc. of RECOMB*, pages 15–24, 1999.

[6] A. Ben-Hur and D. Brutlag. Remote homology detection: a motif based approach. *Bioinformatics*, 19 Suppl 1:26–33, 2003.

[7] A. L. Berger, S. D. Pietra, and V. J. D. Pietra. A maximum entropy approach to natural language processing. *Computational Linguistics*, 22(1):39–71, 1996.

[8] B. Boeckmann et al. The SWISS-PROT protein knowledgebase and its supplement TrEMBL in 2003. *Nucleic Acids Research*, 31(1):365–370, 2003.

[9] E. C. Buehler and L. H. Ungar. Maximum entropy methods for biological sequence modeling. In *Proc. of BIOKDD*, pages 60–64, 2001.

[10] J. N. Darroch and D. Ratcliff. Generalized iterative scaling for log-linear models. *Annals of Mathematical Statistics*, 43:1470–1480, 1972.

[11] S. R. Eddy. HMMER: Profile hidden Markov modelling. *Bioinformatics*, 14(9):755–763, 1998.

[12] E. Eskin, W. S. Noble, and Y. Singer. Protein family classification using sparse markov transducers. *Journal of Computational Biology*, 10(2):187–214, 2003.

[13] P. G. Ferreira and P. J. Azevedo. Protein sequence classification through relevant sequence mining and bayes classifiers. In *Proc. of EPIA*, pages 236–247, 2005.

[14] P. G. Ferreira and P. J. Azevedo. Query driven sequence pattern mining. In *Proc. of SBBD*, pages 1–15, 2006.

[15] F. Horn et al. GPCRDB information system for G protein-coupled receptors. *Nucleic Acids Research*, 31(1):294–297, 2003.

[16] D. Kang, A. Silvescu, and V. Honavar. RNBL-MN: A recursive naive bayes learner for sequence classification. In *Proc. of PAKDD*, pages 45–54, 2006.

[17] D. Kang, J. Zhang, A. Silvescu, and V. Honavar. Multinomial event model based abstraction for sequence and text classification. In *Proc. of SARA*, pages 134–148, 2005.

[18] S. B. Kotsiantis and P. E. Pintelas. Increasing the classification accuracy of simple bayesian classifier. In *Proc. of AIMSA*, pages 198–207, 2004.

[19] A. Krogh et al. Hidden markov models in computational biology: Applications to protein modeling. *Journal of Molecular Biology*, 235:1501–1531, 1994.

[20] N. Lesh, M. J. Zaki, and M. Ogihara. Mining features for sequence classification. In *Proc. of KDD*, pages 342–346, 1999.

[21] C. Leslie, E. Eskin, and W. Noble. Mismatch string kernels for SVM protein classification. In *Proc. of NIPS*, pages 1417–1424, 2002.

[22] D. J. Lipman et al. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, 1990.

[23] R. Malouf. A comparison of algorithms for maximum entropy parameter estimation. In *Proc. of Sixth Conf. on Natural Lang. Learning*, pages 49–55, 2002.

[24] K. Marsolo and S. Parthasarathy. Protein classification using summaries of profile-based frequency matrices. In *Proc. of BIOKDD06*, pages 51–58, 2006.

[25] A. McCallum and K. Nigam. A Comparison of Event Models for Naive Bayes Text Classification. In *Proc. of AAAI-98 Workshop on Learning for Text Categorization*, pages 41–48, 1998.

[26] I. Melvin et al. Multi-class protein classification using adaptive codes. *Journal of Machine Learning Research*, 8:1557–1581, 2007.

[27] K. Nigam, J. Lafferty, and A. McCallum. Using maximum entropy for text classification. In *Proc. of IJCAI-99 Workshop on ML for Info. Filtering*, pages 61–67, 1999.

[28] D. Pavlov. Sequence modeling with mixtures of conditional maximum entropy distributions. In *Proc. of ICDM*, pages 251–258, 2003.

[29] W. R. Pearson and D. J. Lipman. Improved tools for biological sequence comparison. *Proc. National Academy Sciences USA*, 85(8):2444–2448, 1988.

[30] S. D. Pietra, V. J. D. Pietra, and J. D. Lafferty. Inducing features of random fields. *IEEE Tran. on Pattern Analysis and Machine Intel.*, 19(4):380–393, 1997.

[31] P. Rani and V. Pudi. RBNBC: Repeat Based Naive Bayes Classifier for Biological Sequences. To appear in Proc. of ICDM'08: IEEE Int. Conf. on Data Mining, 2008.

[32] A. Ratnaparkhi. A simple introduction to maximum entropy models for natural language processing. Technical report, IRCS Report 97-98, Univ. of Pennsylvania, 1997.

[33] A. Ratnaparkhi. *Maximum Entropy Models for Natural Language Ambiguity Resolution*. PhD thesis, University of Pennsylvania, 1998.

[34] N. Tatti. Maximum entropy based significance of itemsets. In *Proc. of ICDM*, pages 312–321, 2007.

[35] R. Thonangi and V. Pudi. Acme: An associative classifier based on maximum entropy principle. In *Proc. of ALT*, pages 122–134, 2005.

[36] J. T. L. Wang, M. J. Zaki, H. Toivonen, and D. Shasha, editors. *Data Mining in Bioinformatics*. Springer, 2005.