RBNBC: Repeat Based Naive Bayes Classifier for Biological Sequences

by

Pratibha Rani, Vikram Pudi

Report No: IIIT/TR/2008/126

Centre for Data Engineering
International Institute of Information Technology
Hyderabad - 500 032, INDIA
September 2008
RBNBC: Repeat Based Naive Bayes Classifier for Biological Sequences

Pratibha Rani Vikram Pudi
Center for Data Engineering
IIIT Hyderabad, India
pratibha_rani@research.iiit.ac.in, vikram@iiit.ac.in

Abstract

In this paper, we present RBNBC, a Repeat Based Naive Bayes Classifier of bio-sequences that uses maximal frequent subsequences as features. The design of RBNBC is based on generic ideas that can apply to other domains where the data is organized as collections of sequences. Specifically, RBNBC uses a novel formulation of Naive Bayes that incorporates repeated occurrences of subsequences within each sequence. Our extensive experiments on two collections of protein families show that RBNBC performs as well as existing state-of-the-art probabilistic classifiers for bio-sequences. This is surprising as RBNBC is a pure data mining based generic classifier that does not require domain-specific background knowledge such as multiple alignment, data transformation and complex feature extraction methods. We note that such domain-specific ideas could further increase the performance of RBNBC.

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 [29], 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 functions and structure because sequences belonging to the same family have similar characteristics.

The known state-of-the-art solutions for this problem mainly use approaches such as Sequence Alignment [2, 23, 28], Hidden Markov Models (HMM) [11, 19], Probabilistic Suffix Trees (PST) [5, 12] and Support Vector Machines (SVM) [6, 7, 22]. Recent approaches [24, 25, 27] 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 pure data mining based, simple but effective solution, which does not require any domain knowledge, called Repeat Based Naive Bayes Classifier (RBNBC)—a new Bayesian classifier which is able to incorporate repeats of subsequences within a sequence. Naive Bayes is well known as a surprisingly successful classification method that has outperformed much more complicated methods in many application domains. However a direct implementation of Naive Bayes will not work well for bio-sequences. We have adapted it to work for bio-sequences. Our algorithm drastically improves the accuracy from 32% (for the direct Naive Bayes) to 98%.

RBNBC has the following desirable features, which can be incorporated in any Bayesian classifier: (1) It uses a novel formulation of Naive Bayes to incorporate repeated occurrences of subsequences within each sequence of a family. (2) Unlike direct Naive Bayes, it works for a nonuniform feature set (where all features are not present in each class). (3) It uses a bit-vector based optimization that drastically reduces the time required to extract frequent subsequences from a large dataset of sequences. (3) It uses an entropy based feature selection method to find the discriminating features for a class and to reduce the number of irrelevant features. (4) Being a Bayesian classifier, it is scalable with the database size and with the number of classes. (5) It does not require domain knowledge based ideas such as 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. Section 3 provides critical definitions used in the paper and describes the proposed method for estimating feature probabilities. Section 4 solves a problem of Naive Bayes that arises when all features are not present in all classes. Section 5 describes the overall design of the proposed RBNBC classifier. Section 6 describes all the experiments and results and Section 7 discusses these results. Section 8 presents the related work and Section 9 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\(^4\) 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:

\[
P(F_i|S) = \frac{P(S|F_i)P(F_i)}{P(S)}
\]  

(1)

Since the denominator \( P(S) \) is common for all families, it is left out. \( P(F_i) \) is trivial to compute as relative frequency of family \( F_i \) in \( D \). Hence the classification problem is translated to the correct estimation of \( P(S|F_i) \) from \( D \).

3 Estimating Feature Probabilities

Starting with a few definitions, in this section we describe our method of estimating feature probabilities.

**Definition 1.** 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.** 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

\[
\text{Sequence Count of } X_j \text{ in } F_i \geq \sigma
\]

where \( \sigma \) is the MinsupCount for family \( F_i \) calculated using the user given support threshold \( \text{minsup} \) and total number of sequences \( n_i \) in family \( F_i \) as: \( \sigma = n_i \times \text{minsup} \)

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

\[
\exists X_k \in F \text{ such that } X_k \supset X_j
\]

Since Sequence Count does not account for multiple occurrences of \( X_j \) in a sequence, we present following method using Repeat Counts to estimate the probability \( P(X_j|F_i) \) of a feature \( X_j \) in a family \( F_i \). In our study we found that Repeat Count results in better accuracy as it uses all the occurrences of a feature. Use of multiple occurrences of a feature is similar to the multinomial event model \[26\] used in text classification but we follow a very different approach to find the feature probabilities.

1. Find the number of slots available for \( X_j \) in family \( F_i \) containing \( n_i \) sequences.
   - If the features are overlapping then:
     \[
     \text{slots}_{ij} = \sum_{k=1}^{n_i} [\text{length of } S_k - (\text{length of } X_j - 1)]
     \]  
     (2)
   - If the features are non overlapping then:
     \[
     \text{slots}_{ij} = \sum_{k=1}^{n_i} \left\lfloor \frac{\text{length of } S_k}{\text{length of } X_j} \right\rfloor
     \]  
     (3)

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

\[
P(X_j|F_i) = \frac{\text{Repeat Count of } X_j \text{ in } F_i}{\text{slots}_{ij}}
\]  

(4)

Equations 2 and 3 find the total slots for feature \( X_j \) in family \( F_i \), i.e., the total number of times \( X_j \) can, in principle, occur in \( F_i \). This is done by summing the available slots from each sequence of the family. Next, the feature probability is estimated as the fraction of times \( X_j \) actually occurs over the slots.

4 Problems with Naive Bayes

For a query sequence \( S \) represented as a feature vector \( X = \{ X_1, X_2, \ldots, X_m \} \), Naive Bayes (NB) assumes strong independence among the features and estimates \( P(S|F_i) \) by multiplying the feature probabilities. It calculates the posterior probability of each family as:

\[
P(F_i|S) = P(F_i) \prod_{j=1}^{m} P(X_j|F_i)
\]  

(5)

Along with the advantages of simplicity and speed, the NB classifier has the merit that even in cases where the independence assumption is not strictly satisfied it performs surprisingly well and equivalent to state-of-the-art classifiers on a variety of datasets, including complex real world datasets \[9, 18, 30\].

One of the problems with the above formulation is that when very small feature probability values are multiplied in Equation 5, the product can go below the available minimum number range of the processor. An appropriate scaling factor or log scaled formulation is used to avoid this problem. The second problem is discussed below.

4.1 Problem of 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 NB formulation described in Equation 5 cannot be applied directly on bio-sequences when frequent subsequences are used as features. Known solutions are:

1. To 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 classified 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.

2. To incorporate a small sample-correction into all probabilities, such as the Laplace correction factor [10, 18], which requires changing all the probability values. So it is not feasible for datasets with a large feature set like bio-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].

We experimented with two models of the NB classifier for bio-sequences—model A using solution (1) and model B using solution (3)—and found that model B performed better than model A. In RBNBC we use another solution described in Section 5.3, which outperformed both A and B models.

5 The RBNBC Classifier

The RBNBC 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. This results in a smaller set of features and their

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 5, 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.

**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 feature set left after pruning.

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

5.1 Feature Extraction

Many sophisticated feature mining algorithms [14, 20, 21] exist for bio-sequences, but we have used simple features, avoiding the need for complex data transformations and domain knowledge. We believe that frequent subsequences capture everything that is significant in a collection of sequences. This assumption has borne out well in the experimental results.

Since the number of extracted frequent features increases exponentially as $\text{minsup}$ decreases, to reduce the feature
set we have opted to use maximal frequent subsequences as features. There may be some loss in information by using maximal frequent subsequences as features, however, they satisfy the following criteria set by [21], which are necessary for features of any classifier: (1) Significant features: we ensure this by considering only frequent features (i.e., \( \text{Sequence Count} \geq \text{Minsup Count} \)). (2) Non-redundant Features: we ensure this by using maximal frequent subsequences as features. (3) Discriminative Features: for ensuring this, we use the entropy based selection criteria described in Section 5.2 after extraction of features.

We extracted maximal frequent subsequences using an Apriori-like method using the same minimum support threshold for all families. To extract all possible features, we set \( \text{maxlen} \)–maximum length of the features to be extracted—as the length of the largest sequence of the training set.

### 5.1.1 Bit-Vector based Optimization of Frequent Subsequence Extraction

We have optimized the time consuming and memory intensive frequent subsequence extraction process by avoiding extraction of infrequent subsequences by storing information of their location in a bit-vector. This optimization proved to be very effective and reduced the feature extraction time from days to hours.

```plaintext
for each sequence \( S \) of the family:
    Initialize a bitvector \( B_S \) of ’1’s of length equal to that of \( S \)
for \( l = 1 \) to \( \text{maxlen} \): # length of subsequences to consider
    for each sequence \( S \) of the family:
        if \( p = 1 \):
            extract feature \( X \) of length \( l \) starting from position \( p \)
            increment counters measuring occurrences of \( X \)
            set \( p = 0 \)
        for each extracted feature \( X \) of length \( l \):
            if SequenceCount of \( X \) \( \geq \) Minsup Count:
                for each sequence \( S \) containing \( X \):
                    for each occurrence \( q \) of \( X \) in \( S \):
                        set \( p = 1 \) in \( B_S \) corresponding to the position of \( q \)
```

![Figure 2: Optimization of frequent subsequence extraction.](image)

The procedure first initializes a bit-vector of ’1’s for each sequence in a family which is of the same length as the sequence. Then it starts extracting frequent subsequences of length one and iteratively proceeds to longer subsequences. 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’.

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.

### 5.2 Feature Selection

As is typical of frequent pattern mining, the feature extraction phase creates the problem of curse of dimensionality which increases as the \text{minsup} decreases. Entropy based criteria [18] like information gain and gain ratio have been used to tackle this problem by selecting only the important features.

Since our aim is to find discriminating features [21] for each family, we use \( H(D|X_j = \text{present}) \), i.e., entropy of the dataset in the presence of a feature as the selection criterion:

\[
H(D|X_j = \text{present}) = -\sum_{i=1}^{N} \left[ P(F_i|X_j = \text{present}) \log[P(F_i|X_j = \text{present})] \right]
\]

where

\[
P(F_i|X_j = \text{present}) = \frac{\text{Sequence Count of } X_j \text{ in } F_i}{\sum_k \text{Sequence Count of } X_j \text{ in } F_k}
\]

Analysis of this criterion gives following observations: (1) \( H(D|X_j = \text{present}) = 0 \) when a feature \( X_j \) is present in one and only one family. (2) \( H(D|X_j = \text{present}) \) is higher when a feature \( X_j \) is present in all families.

For selecting features we compare a user-given threshold \( H_{th} \) with the calculated value of \( H(D|X_j = \text{present}) \), and select all the features satisfying the criteria \( H(D|X_j = \text{present}) \leq H_{th} \) while pruning the others.

Experimentally we found that for very low \text{minsup} values, using threshold \( H_{th} = 0 \) gives good results in the classification phase. But for other \text{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 40% for low \text{minsup} values, as shown in Table 1.

### 5.3 Classification

RBNBC 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 4.1). 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 \( \epsilon \), which is the minimum possible probability computed using repeat counts for a feature. It is obtained as:

\[
\epsilon = \frac{1}{\text{Sum of the lengths of sequences of the largest family}}
\]

For classifying a query sequence \( S \), RBNNB finds the uniform feature set \( F \), which is the set of features present in \( S \), collected from all families. It then uses Equation 4 for finding probabilities of features present in a family and uses \( \epsilon \) as the probability for features not present in that family. It uses these probabilities to compute the posterior probability of all families using Equation 5. 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 3.

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 \( X_j \in F_i \) : use Equation 4 to compute \( P(X_j|F_i) \)
      else : set \( P(X_j|F_i) = \epsilon \)
   b. Use Equation 5 to compute \( P(F_i|S) \).
3. Find family \( F_k \) having the largest value of \( P(F_i|S) \).
4. Classify \( S \) into \( F_k \).

Figure 3: Classification phase of RBNNB.

6 Experiments and Results

We have used two collections of protein families to evaluate the performance of RBNNB. The first collection is a large collection of 8435 \( G \) protein-coupled receptors (GPCRs) arranged in 13 families called superfamilies, taken from the March-2005 Release 9.0 of GPCRDB [15] (http://www.gpcr.org/7tm). 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 2097 proteins arranged in 26 families was obtained from the Feb-2008 Release 55.0 of SWISSPROT [8] using the list of SWISSPROT protein IDs obtained from Pfam [4] version 22.0. 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: All the experiments were performed on a 1.9 GHz AMD Athlon 64 processor machine with 385 MB RAM, running Linux Fedora Core 4. All the programs were written in the Perl language.

For comparison with the Naive Bayes classifier, we implemented models A and B described in Section 4. In all experiments we used the same set of features for all the classifiers.

We divided the dataset into training and test sets in the ratio 9:1 with stratification. From the training set we extracted all possible maximal frequent subsequences for each family using the same \( \text{minsup} \) for all families. We used maximal subsequences of length greater than two as features to avoid trivial frequent subsequences of lengths one and two.

Table 1 gives the number of features for different \( \text{minsup} \) values, used in the experiments with GPCRDB dataset. 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\% \( \text{minsup} \) we used threshold \( H_{th} = 0.0 \) and for other \( \text{minsup} \) values we used \( \frac{1}{2}(\text{entropy of the dataset}) \) as threshold to prune the features.

<table>
<thead>
<tr>
<th>Table 1. Number of features for GPCRDB</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{minsup} (%) )</td>
</tr>
<tr>
<td>--------------------------</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>30</td>
</tr>
<tr>
<td>50</td>
</tr>
</tbody>
</table>

Table 3 summarizes the results of experiments done on the classifiers with different \( \text{minsup} \) values using the feature set (before and after pruning).

Tables 2 and 4 present the results of experiments using the set of features after pruning. The tables show the family-wise and average accuracies of the classifiers with corresponding minimum support values. The Size column of Tables 2 and 4 give the number of sequences of each 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]. We should remind that the test sets and evaluation methods used by them are different from ours and so the results should not be compared strictly. Due
Table 2. Family-wise and Average Classification results of NB models (A and B) and RBNBC for the collection of 848 GPCRs of 13 families from GPCRDB

<table>
<thead>
<tr>
<th>Family Name</th>
<th>Size</th>
<th>Model A (%)</th>
<th>Model B (%)</th>
<th>RBNBC (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class A</td>
<td>6031</td>
<td>83.2</td>
<td>57.5</td>
<td>98.35</td>
</tr>
<tr>
<td>Class B</td>
<td>309</td>
<td>0.0</td>
<td>96.7</td>
<td>93.55</td>
</tr>
<tr>
<td>Class C</td>
<td>206</td>
<td>0.0</td>
<td>100.0</td>
<td>100.0</td>
</tr>
<tr>
<td>Class D</td>
<td>65</td>
<td>0.0</td>
<td>66.7</td>
<td>66.67</td>
</tr>
<tr>
<td>Class E</td>
<td>10</td>
<td>0.0</td>
<td>100.0</td>
<td>100.0</td>
</tr>
<tr>
<td>Ocular albinism proteins</td>
<td>8</td>
<td>0.0</td>
<td>100.0</td>
<td>100.0</td>
</tr>
<tr>
<td>Frizzled family</td>
<td>130</td>
<td>0.0</td>
<td>92.3</td>
<td>92.31</td>
</tr>
<tr>
<td>Insect odorant receptors</td>
<td>236</td>
<td>0.0</td>
<td>4.2</td>
<td>37.5</td>
</tr>
<tr>
<td>Plant Mlo receptors</td>
<td>52</td>
<td>0.0</td>
<td>100.0</td>
<td>100.0</td>
</tr>
<tr>
<td>Nematode chemoreceptors</td>
<td>755</td>
<td>0.0</td>
<td>42.7</td>
<td>90.67</td>
</tr>
<tr>
<td>Vomeronasal receptors</td>
<td>286</td>
<td>10.3</td>
<td>82.7</td>
<td>89.65</td>
</tr>
<tr>
<td>Taste receptors T2R</td>
<td>237</td>
<td>0.0</td>
<td>87.5</td>
<td>95.83</td>
</tr>
<tr>
<td>Class Z Archaeal opsins</td>
<td>110</td>
<td>9.0</td>
<td>100.0</td>
<td>100.0</td>
</tr>
</tbody>
</table>

Average (%)                          | 60.14 | 60.38       | 95.17       |

minsup (%)                            | 50    | 5           | 5           |

7 Discussion

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

From the results of Table 2, it is evident that the NB model A classifier, which uses a nonuniform feature set, is biased towards the largest family but RBNBC is able to break the biasing effect of large families and so performs very well on the skewed dataset also.

The results of Table 3 show that performance of NB models A and B improves when feature set is pruned while RBNBC outperforms them for all minsup values and it does not matter whether the feature set is pruned or not. This indicates that the performance of RBNBC is not dependent on the pruning phase and so we can conclude that RBNBC can perform well on datasets with large feature sets, whereas the other NB classifiers can not. The NB model B classifier, which uses a uniform feature set with Sequence Count and a small correction probability, performs better than NB model A. But it is never able to compete with RBNBC, which uses Repeat Count. This shows that Repeat Count gives better model of the sequence families than Sequence Count.

Results presented in Table 4 on the Pfam dataset indicate that performance of RBNBC is comparable to the performances of C classifier of [13] and PST [5]. C classifier uses a sophisticated feature extraction algorithm [14] and fine tunes the feature extraction process of each family by using different minsup values for different families. We note that RBNBC attains comparable performance without using any domain specific ideas or fine tuning and if such ideas were
Table 4. Family-wise and Average Classification results of NB models (A and B), RBNBC, Ferreira et al.’s C model and PST for the collection of 211 proteins of 26 families from Pfam 22.0

<table>
<thead>
<tr>
<th>Family Name</th>
<th>Size</th>
<th>NB Model A (%)</th>
<th>NB Model B (%)</th>
<th>RBNBC (%)</th>
<th>C (%)</th>
<th>PST (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>7tm-1</td>
<td>64</td>
<td>0.0</td>
<td>33.3</td>
<td>100.0</td>
<td>100.0</td>
<td>93.0</td>
</tr>
<tr>
<td>7tm-2</td>
<td>32</td>
<td>0.0</td>
<td>66.7</td>
<td>66.7</td>
<td>100.0</td>
<td>94.4</td>
</tr>
<tr>
<td>7tm-3</td>
<td>29</td>
<td>0.0</td>
<td>66.7</td>
<td>66.7</td>
<td>100.0</td>
<td>83.3</td>
</tr>
<tr>
<td>AAA</td>
<td>210</td>
<td>0.0</td>
<td>76.2</td>
<td>100.0</td>
<td>97.7</td>
<td>87.9</td>
</tr>
<tr>
<td>ABC-tran</td>
<td>57</td>
<td>0.0</td>
<td>33.3</td>
<td>83.3</td>
<td>100.0</td>
<td>83.5</td>
</tr>
<tr>
<td>ATP-synt-ab</td>
<td>151</td>
<td>0.0</td>
<td>86.7</td>
<td>100.0</td>
<td>98.9</td>
<td>96.3</td>
</tr>
<tr>
<td>ATP-synt-A</td>
<td>30</td>
<td>0.0</td>
<td>33.3</td>
<td>66.7</td>
<td>92.4</td>
<td></td>
</tr>
<tr>
<td>ATP-synt-C</td>
<td>33</td>
<td>0.0</td>
<td>66.7</td>
<td>66.7</td>
<td>97.7</td>
<td>96.3</td>
</tr>
<tr>
<td>c2</td>
<td>286</td>
<td>0.0</td>
<td>55.2</td>
<td>87.6</td>
<td>91.3</td>
<td>92.3</td>
</tr>
<tr>
<td>CLP-protease</td>
<td>88</td>
<td>0.0</td>
<td>77.8</td>
<td>88.9</td>
<td>94.1</td>
<td>87.9</td>
</tr>
<tr>
<td>COesterase</td>
<td>126</td>
<td>0.0</td>
<td>100.0</td>
<td>100.0</td>
<td>91.7</td>
<td></td>
</tr>
<tr>
<td>cox1</td>
<td>23</td>
<td>0.0</td>
<td>100.0</td>
<td>100.0</td>
<td>90.9</td>
<td>83.8</td>
</tr>
<tr>
<td>cox2</td>
<td>32</td>
<td>0.0</td>
<td>100.0</td>
<td>100.0</td>
<td>100.0</td>
<td>98.2</td>
</tr>
<tr>
<td>Cys-knot</td>
<td>24</td>
<td>0.0</td>
<td>100.0</td>
<td>100.0</td>
<td>100.0</td>
<td>93.4</td>
</tr>
<tr>
<td>Cytochrom-B-C</td>
<td>9</td>
<td>0.0</td>
<td>0.0</td>
<td>100.0</td>
<td>79.2</td>
<td></td>
</tr>
<tr>
<td>Cytochrom-B-N</td>
<td>8</td>
<td>0.0</td>
<td>100.0</td>
<td>85.7</td>
<td>98.2</td>
<td></td>
</tr>
<tr>
<td>HCV-NS1</td>
<td>10</td>
<td>0.0</td>
<td>100.0</td>
<td>100.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Oxidored-q1</td>
<td>33</td>
<td>0.0</td>
<td>100.0</td>
<td>93.5</td>
<td>97.1</td>
<td></td>
</tr>
<tr>
<td>PKinase</td>
<td>54</td>
<td>0.0</td>
<td>40.0</td>
<td>60.0</td>
<td>96.2</td>
<td>85.1</td>
</tr>
<tr>
<td>PPR</td>
<td>100</td>
<td>0.0</td>
<td>90.0</td>
<td>100.0</td>
<td>22.6</td>
<td></td>
</tr>
<tr>
<td>RubisCO-large</td>
<td>16</td>
<td>0.0</td>
<td>100.0</td>
<td>100.0</td>
<td>81.3</td>
<td>98.7</td>
</tr>
<tr>
<td>rvt-l</td>
<td>156</td>
<td>0.0</td>
<td>100.0</td>
<td>100.0</td>
<td>79.6</td>
<td>88.4</td>
</tr>
<tr>
<td>RVT-thumb</td>
<td>41</td>
<td>0.0</td>
<td>24.1</td>
<td>100.0</td>
<td>91.2</td>
<td></td>
</tr>
<tr>
<td>TPR-1</td>
<td>274</td>
<td>55.17</td>
<td>93.10</td>
<td>54.9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>zf-C2H2</td>
<td>101</td>
<td>0.0</td>
<td>60.0</td>
<td>70.0</td>
<td>88.4</td>
<td>92.3</td>
</tr>
<tr>
<td>zf-CCHC</td>
<td>110</td>
<td>0.0</td>
<td>63.6</td>
<td>81.8</td>
<td>100.0</td>
<td>88.6</td>
</tr>
<tr>
<td><strong>Average (%)</strong></td>
<td>10.43</td>
<td>66.82</td>
<td>90.52</td>
<td>90.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>minsup (%)</strong></td>
<td>50</td>
<td>5</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

incorporated, the performance of RBNBC could increase.

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 2 and 4 shows that although GPCRDB dataset is larger in size and skewed than Pfam dataset, RBNBC performs better on the GPCRDB dataset than the Pfam dataset. So RBNBC can be better than other sequence classifiers for large and skewed datasets with less similarity among sequences.

Classifiers like PSTs, SMTs, Bayesian classifier of [3] and the C classifier require parameters other than minsup such as feature length, sequence length etc. to be supplied by the user for the feature extraction process. It is observed that performances of the classifiers are very sensitive towards these parameters. RBNBC does not require 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 5.2.

8 Related Work

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 \( k \), where length \( k \) is a user supplied parameter.

A query sequence driven computationally expensive subsequence extraction method [14], guided by many user supplied parameters, is used in [13] to find rigid gap frequent subsequences of a certain minimum length. These subsequences are used to obtain two features which are combined in the NB classifier. 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. It 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, 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 SMT [12] classifier generalizes PSTs to SMTs by incorporating wildcard support, which is a symbol that denotes a gap of size one and matches any symbol on the alphabet. As biosequence 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 are very complex to implement and the generated models tend to be space inefficient and require large memory.

**Similarity based Sequence Classifiers** [2, 23, 28] compare an unlabeled sequence with all the sequences of the database and assess sequence similarity using sequence alignment methods like FASTA [28], BLAST [23] or PSI-BLAST [2]. But as the number of sequences in bio-databases 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, 7, 22, 24, 25, 27] either use a set of features of protein families to train SVM or use 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. Also, these classifiers require carefully selected positive and negative samples for each class in the training phase.

9 Conclusions

In this paper, we proposed a new *Naive Bayesian* classifier which uses the simplest possible features of sequence families by extracting maximal frequent subsequences from each family and classifies a query sequence by incorporating the *Repeat Count* of features present in the query sequence. We discussed the problems associated with NB classifiers and proposed very simple but effective solutions to handle them by using a *uniform feature set* and a small correction probability. Experiments on two datasets demonstrated that RNBC’s performance is comparable to the PST and the C classifier and it outperforms the other NB classifiers which use *Sequence Count*. Experiments also showed that RNBC performs very well even on large and skewed dataset.

References


