Localization Site Prediction for Membrane Proteins by Integrating Rule and SVM Classification

Senqiang Zhou Ke Wang
School of Computing Science
Simon Fraser University
{szhoua@cs.sfu.ca, wangk@cs.sfu.ca}

ABSTRACT

We study the localization prediction of membrane proteins for two families of medically important disease-causing bacteria, called Gram-Negative and Gram-Positive bacteria. Each such bacterium has its cell surrounded by several layers of membranes. Identifying where proteins are located in a bacterial cell is of primary research interests for antibiotic and vaccine drug design. This problem has three requirements. *First*, with any subsequence of amino acid residues being potentially a dimension, it has an extremely high dimensionality, few being irrelevant. *Second*, prediction of a target localization site must have a high precision in order to be useful to biologists, i.e., at least 90% or even 95%, while recall is as high as possible. Achieving such a precision is made harder by the fact that target sequences are often much fewer than background sequences. *Third*, the rationale of prediction should be understandable to biologists for taking actions. Meeting all these requirements presents a significant challenge in that a high dimensionality requires a complex model that is often hard to understand.

The support vector machine (SVM) model has an outstanding performance in a high dimensional space, therefore, addresses the first two requirements. However, the SVM model involves many features in a single kernel function, therefore, does not address the third requirement. We address all three requirements by integrating the SVM model with a rule-based

model, where the understandable if-then rules capture "major structures" and the elaborated SVM model captures "subtle structures". Importantly, the integrated model preserves the precision/recall performance of SVM and, at the same time, exposes major structures in a form understandable to the human user. We focus on searching for high quality rules and partitioning the prediction between rules and SVM so as to achieve these properties. We evaluate our method on several membrane localization problems. The purpose of this paper is not improving the precision/recall of SVM, but is *manifesting* the rationale of a SVM classifier through *partitioning* the classification between if-then rules and the SVM classifier, and *preserving* the precision/recall of SVM.

Index terms. H.2.8.a Bioinformatics (genome or protein) databases, H.2.8.b Clustering, classification, and association rules

1. INTRODUCTION

In the last decade, biologists have accumulated a huge amount of biological sequences thanks to the high throughput genome sequencing projects. Proteins are one kind of such sequences. Each protein is composed of a linear sequence of amino acid residues. The same 20 amino acid residues (i.e., 20-letter alphabet) are used virtually for all proteins on the earth. Though a protein molecule has 3D shape, amazingly it is generally true that the 3D location of every atom of a protein molecule in a living organism is fully determined by this linear sequence over the 20-letter alphabet [5]. Since proteins play critical roles in determining the structures and functions of all living organisms [30], classifying these sequences into corresponding functional families is an important task for biologists.

One of the most important protein classification problems is to predict the subcellular localization of proteins [12]. For proper functioning, a protein has to be transported to the correct intra- or extra-cellular compartments in a soluble form, or attached to a membrane that surrounds

the cell; hence the subcellular localization of a protein plays a key role with regard to its functions. In particular, the study of a family of disease-causing bacteria, collectively known as Gram-Negative bacteria, has shown that such bacteria have a distinct cell structure. In a Gram-Negative bacterial cell, a protein may be resident at one of 5 primary localization sites as illustrated in Figure 1. Proteins are synthesized in the cytoplasm and may remain there, or be transported to the inner membrane, the periplasm, the outer membrane, or the extra-cellular environment. The ability to identify the localization site from the sequence information alone would allow researchers to quickly prioritize a list of proteins for potential drug and vaccine targets [29].

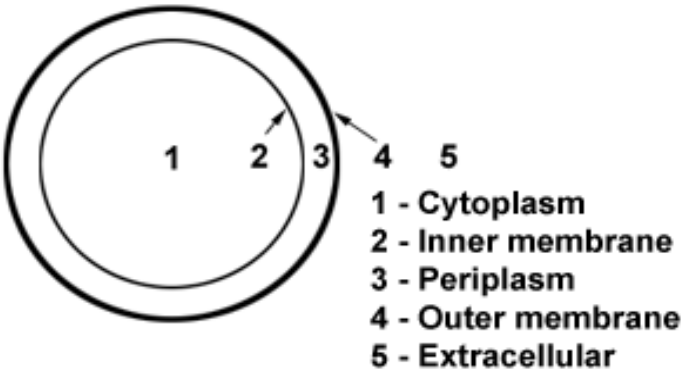

Figure 1. The five primary localization sites in a Gram-Negative bacterial cell

The above problem can be summarized as predicting the localization site for a given protein from its amino acid sequence with the following requirements.

- **High precision**. The precision of predicting the target localization site must be "very high", in most cases at least 90% or even 95%, while the recall is as high as possible. This means that whenever a protein is predicted to be located at the target site, the biologist wants to be fairly sure that the prediction is correct [29]. This is in contrast to the rare-class classification in the data mining literature where a high recall is the priority, such as identifying buyers in direct marketing or intruders in intrusion detection. Achieving a high precision is made harder by

the fact that the target examples are often much fewer than the examples in the contrasting class.

- **Interpretability of models**. Relevant patterns that summarize what triggers the prediction in a concise form are useful for the biologist to perform further analysis and devise actions. For example, as outer-membrane proteins are exposed on the surface of the cell, they represent potential drug and vaccine targets, and the ability to identify and summarize such potential targets would allow researchers to quickly prioritize a list of proteins for further study. To address this issue in a domain-independent manner, our notion of interpretability refers to the *syntax simplicity* of the model, such as the number and size of rules, not anything that requires intimate domain knowledge.
- **High dimensionality**. With any subsequence of amino acids being potentially a feature, it is common to have tens or even hundreds of thousands of features that are necessarily relevant. Typically, combinations of features must be used to achieve a high precision because any individual feature tends to appear in both the target class and the contrasting class. Searching such combinations in a high dimensional space requires pruning a large portion of search space.

Meeting all these requirements presents a significant challenge because an inherently high dimensional problem requires a complex model that is hard to understand. Recent research progress shows that SVMs (Supper Vector Machines) [31] demonstrate superior performance gains and robustness in many applications over traditional methods[1]. One striking property of SVMs is the ability to produce the unique global minimum of the error function [9]. Another striking property is that its ability to learn is independent of the dimensionality of the feature space [17] because SVMs measure the complexity of hypotheses based on the margin with which they separate the data, not the number of features. These properties make the SVM model an

[1] http://www.clopinet.com/isabelle/Projects/SVM/applist.html

ideal candidate for addressing the requirements on high precision and high dimensionality. However, the SVM model does not address the interpretability requirement: it involves thousands of features in a single kernel function, making it impossible to see a simple relationship between the prediction and features that trigger it. A rule-based model such as ID3 and C4.5, on the other hand, presents the logic of prediction in the user-friendly if-then rule format, but is inferior in performance for high dimensional problems. For example, a length-2 rule Education=H∧Gender=M -> "Yes" says that if education level is high and gender is male, the class is "Yes".

Our contributions. To address all the above requirements, we integrate the SVM model with the rule-based model. The idea is to partition the SVM classification so that a small number of rules capture "simple, major structures" and the SVM model captures "complex, subtle structures". The integrated model, called *rule-SVM (rSVM),* places the rules at the top and the SVM at the bottom: to classify a protein, the SVM classifier is applied only if there is no matching rule. Therefore, the rules *steal* classification from the SVM. The purpose of rSVM is to ***expose*** the rationale of (part of) the SVM classification through understandable if-then rules while ***preserving*** the accuracy (i.e., recall and precision) of the SVM classifier. Importantly, our focus is not on improving the accuracy of SVM; we simply preserve it. Rather, our focus is on manifesting the rationale of SVM through extracting if-then rules and preserving the accuracy of SVM. To preserve the accuracy, it is important not to replace the SVM classifier entirely with rules, but to replace only the part of classification that can be expressed by simple rules accurately; the SVM still covers the remaining classification that is too complex for rules. In this sense, our approach ***combines*** SVM with rules.

The above focus differentiates our work from those in the literature. He et al. [46] uses a serial connection of a decision list and a linear separator to improve accuracy, where a decision list is a ranked list of features (not rules). In our terms, such features are length-1 rules. In our experiences, such features are too general to be accurate, especially in a high dimensional space. We observed that a combination of several lower ranked features, i.e., a length-k rule for k>1,

often has a higher precision than a single highly ranked feature, as demonstrated by the fact that most quality rules extracted in our experiments have length longer than 1. She et al. [29] uses a 2-level rule-based classifier to improve recall, at the expense of reduced precision. However, the performance is inferior to SVM. Neural network has the similar "black-box" problem as SVM, and there has been work on extracting if-then rules from neural network. Those works attempted to replace a neural network classifier entirely with rules [1], which has to pay the cost of unmatched performance.

The Recursive Feature Elimination (RFE) technique [51] extracts meaningful features by using SVM to rank features and pruning lower ranked features iteratively. Two points should be noted about the RFE technique. First, extracted features still do not tell exactly what features are responsible for each class because the final SVM still involves all remaining features in a single kernel function. Second, RFE serves our purpose of exposing the rationale of classification only if extracted features are few enough for the human user to understand them easily, say a few dozens. This seems to be the case for the cancer data sets studied in [51] where it was shown that 7129 features can be reduced to 64 features without sacrificing the performance. Two points should be noted about this result. First, the remaining 64 features still do not tell exactly what features are responsible for each class. Second, as the authors pointed out, "both data sets proved to be relatively easy to separate" (quoted from the paper, Section 4.1). That is, these data sets inherently have very few relevant features. However, for the domain considered in our work, the number of features required is typically in the order of thousands to preserve the accuracy of SVM. In such cases, which we believe is more typical, it is not reasonable to expect the human user to understand the classification by examining a kernel function involving thousands of features. We deal with this problem by finding a small number of rules for the positive class where each rule involves a small number (3 or 4) of features.

In the rest of the paper, we review related works in Section 2, present an overview of our approach in Section 3, present the details of our approach in Section 4, 5, 6. In Section 7, we evaluate our approach on several localization prediction tasks. Finally, we conclude the paper.

2. RELATED WORK

2.1 Membrane protein localization

Membrane protein localization has been studied in many years in the biological domain. A simple method is calculating amino acid composition in proteins. [10] used single-residue statistics and [23] used adjacent-pair-residue statistics. The reason that these algorithms work is that there are several relatively simple relationships between a protein's localization site and the frequency of occurrence of particular amino acids. However, these algorithms cannot achieve higher accuracy. [8][18][19] proposed a more general method that counts the frequency of short subsequence of consecutive amino acid residues (called motifs) defined by regular expressions, with a maximum length to restrict the search space. Such motifs represent patterns localized to a short region. Our study suggests that patterns could span a global region where pieces of patterns are separated by gaps of arbitrary length.

Other works on the localization problem include neural network [11][15][27], Markov chain model [35], hidden Markov model [21], SVM [14][32]. PSORT-B in [13] combines several methods to improve the accuracy of prediction. Furthermore, the dataset used in these researches, except for [13] was extracted from entries in the well-known publicly accessible protein database SWISSPROT [38] with annotated localization information, and it has been known that many SWISSPROT annotations were not experimentally confirmed, but rather they were deduced through protein homology analysis or other protein localization prediction algorithms, and have been found to contain incorrect entries. In all these works, the prediction is a black-box because there was no attempt to make the prediction understandable.

A small amount of research has been done into the prediction of outer membrane proteins, including neural network-based methods [11][15], hydrophobicity analysis [39], and homology analysis and amino acid abundance [40][41]. The most recent approach, reported by Martelli et al. [42] is, to date, the most successful attempt. The only 12 non-redundant constitutive β-barrel

membrane proteins whose 3D structure have been resolved atomically were used to train the HMM and build a topological model of the outer membrane proteins. Once such topology is determined, classification is done by computing the probability of the protein sequence being emitted by this model. They achieved a fairly good recall of 84% and an overall accuracy of 89.4% on their testing dataset (note that this is different from the precision on outer membrane proteins alone, see below).

Unfortunately, none of the above methods for outer membrane prediction are publicly-available over the web. Additionally, most were trained on small datasets, some of which may contain only putative outer membrane proteins and were not tested on a dataset of known outer membrane proteins - rather they were used to screen genomes and the number of outer membrane proteins found was reported. Thus there is no way to critically evaluate any methods other than Martelli et al's HMM. Furthermore, all previous protein localization research evaluated the classification performance based on overall accuracy and weighted all locations equally. In our research, performance evaluation is based on the precision of predicting the target location. In terms of this measure, the method by Martelli et. al obtained a very low precision of 46% on their dataset, as calculated through other measures reported in their publication.

The association rule method in [29] has shown some promising results on outer membrane localization prediction, but the performance is not as good as the SVM classifier, and a large number of rules are used. For example, the classifier that achieves the best result uses 77 association rules, whereas our method produces a better result with 15 rules. Also, the association rule approach depends on careful tuning of minimum support and minimum confidence for rule generation. The choices of these parameters are not always clear but affect the performance of prediction.

2.2 Sequence mining

Most sequence mining algorithms find all patterns that pass a specified interestingness threshold and focus on the scalability issue for large databases [2][7][24][26][33][36][37][50]. Though we also face the scalability issue of finding quality rules, our primary concern is using such rules to preserve the accuracy of a SVM classifier and the interpretability of the resulting classifier. Instead of being interested in individual rules that pass a threshold, we are interested in several rules as a good collection for prediction, and we have to deal with the over-fitting issue arising from the prediction problem.

2.3 Rule-based methods

Traditional rule-based classifiers, such as C4.5 and ID3, perform poorly on high dimensional problems such as the one considered here, compared to the SVM model. The hybrid decision tree [43] builds an upper portion of the standard decision tree and embeds neural networks into some leaf nodes to accomplish the remaining prediction. Their work does not address the issue of preserving the performance of the neural network classifier. The perceptron decision tree generalizes the standard one-attribute split at each internal node by a hyperplane split. See [47] for example. Hence, each conjunct in a rule is a multivariate linear inequality. Though perceptron decision trees have demonstrated good results for some real world problems, they tend to over-fit the data by involving many variables in a hyperplane split, due to the increased flexibility. Also, rules generated by such splits are less interpretable.

Recently, association rules have been used for classification for high dimensional transactional data [4][20][34] and have shown promising results on outer membrane localization prediction [29]. However, this approach has several limitations: it depends on a carefully chosen minimum support; the performance is not as good as SVM; and the number of rules used is large, therefore, not easily understandable.

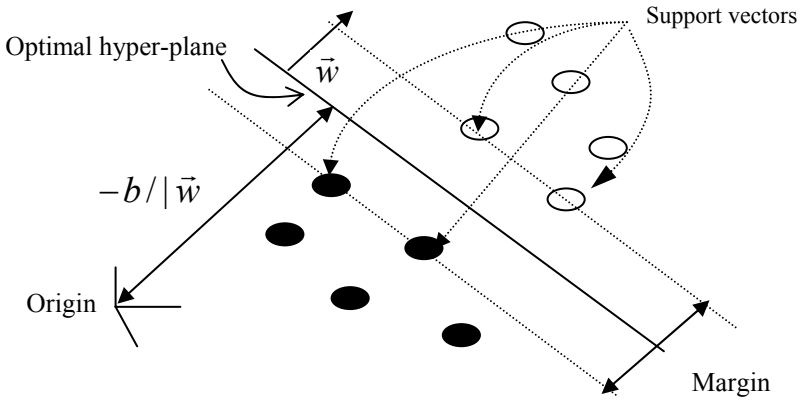

Figure 2. A linear SVM in a two-dimensional space

2.4 Extracting rules from neural networks and SVMs

Extracting understandable rules has been intensively studied for neural network classification [1]. The decomposition method focuses on extracting rules at the level of individual components of neural networks, such as clustering the hidden unit activation, searching for weighted links that caused hidden or output units to be active. The learning-based method extracts rules by using the neural network to generate examples for a rule-based method. The decomposition method can be used on the analysis of support vectors of the SVM [22] and the learning-based method learns what the SVM has learned [45]. All these works attempt to replace the neural network or SVM with the rules extracted, which often produces too many rules and unmatched performance. Our focus of preserving the performance of SVM, by employing only high quality rules and replacing only part of the SVM classification, differs from these works.

3. OVERVIEW

This section presents an overview of our approach. The detailed steps are presented in subsequent sections.

3.1 Background

A sequence is a string of items chosen from a fixed alphabet. For protein sequences, the alphabet consists of the 20 amino acid residues. A labeled sequence is associated with one of two classes: "positive" (+1) or "negative" (-1). A labeled sequence is positive/negative if its class is. Given a collection of labeled sequences D, D(train) and D(test) denote the split of the *training set* and the *testing set.* A classifier is built using D(train) and is evaluated by the precision and recall of classification on the testing sequences in D(test). The *precision* refers to the percentage of positive sequences among those that are classified as positive. The *recall* refers to the percentage of the sequences classified as positive among those that are indeed positive. A classifier is over-fitting if it is only accurate on D(train) but not on D(test). To avoid over-fitting, a classifier should use structures that are statistically significant, therefore, likely repeating in the whole population.

Support vector machines (SVMs) are based on the *Structural Risk Minimization* principle [31] from computational learning theory. The idea is finding a hyper-plane that separates the positive examples from negative ones in the training set while maximizing the distance of them from the hyper-plane. Figure 2 illustrates the maximum margin separating hyper-plane and the support vectors in the two dimensional space. The norm vector $\vec{w}$ represents the direction of the separating hyper-plane, which is determined through a set of *support vectors SV*. To handle non-linearly separable classes, a kernel function K is used to map the original data points into a higher dimensional space in which data points are linearly separable. A new sequence d is classified by the sign of the following decision function:

$$f(d) = \sum \lambda_i y_i K(d_i, d) + b, \quad d_i \in SV \qquad (1)$$

where $K(d_i, d)$ is the kernel function, y_i is the class label of the support vector d_i, λ_i is the weight of d_i, and b is the bias. For a SVM with a linear kernel, Equation (1) can be simplified as:

$$f(d) = \sum_{i=1}^{n} w_i * x_i + b \qquad (2)$$

where $d=<x_1, x_2 \ldots x_n>$, $\vec{w}=<w_1,w_2 \ldots,w_n>$ and w_i is the weight for the *ith* feature. d is classified as positive if $f(d)>0$, and as negative otherwise. In this case, finding the SVM classifier is determining the weight w_i and the bias b. We consider SVMs with a linear kernel function. Our previous studies show that the linear kernel function achieves better or similar results on outer membrane proteins [29]. For the non-linearly separable case, we can first use Equation (1) to transform the problem into a linearly separable problem and apply the method presented in this paper.

3.2 Our approach

We first map each sequence into a data point in a multi-dimensional space. Each dimension, also called a *feature,* is defined by a *frequent segment*, i.e., some consecutive items that occur in at least some minimum fraction of sequences specified by the *minimum support*. We find frequent segments only from positive sequences since we are interested in predicting the positive class. The details of finding frequent segments will be discussed in Section 4. Compared to the spectrum kernel [44] that uses features of a fixed length, our approach allows features of flexible length. Suppose that we have n features. We map a sequence to a 1/0 vector $<x_1, x_2 \ldots x_n>$ in the feature space: if the *ith* feature occurs in the sequence, $x_i=1$, otherwise, $x_i=0$. Alternatively, term frequency and inverse document frequency [28] can be used, in which case x_i is a value between 0 and 1. To focus on the main ideas, we use the 1/0 vector representation for simplicity.

As in most cases, we consider only rules that predict the positive class (but our work can be extended to rules for two classes). A *rule* is a set of features. A rule *matches* a sequence (or

vice versa) if the sequence contains all the features in the rule. Given a set of sequences, the *support* of a rule is the percentage of matched sequences among all the sequences, and the *confidence* of a rule is the percentage of positive sequences among all matched sequences. A rule classifies a matched sequence as the positive class.

We are interested in a classification model that has the performance of SVM classifiers but expresses "major structures" in the form of rules. We propose such a mode called rSVM.

Definition 1. Let M be a SVM classifier. Let $r_1, r_2 \ldots r_m$ be a set of rules. An *rSVM* classifier has the form:

$$r_1, r_2 \ldots r_m, M.$$

R(rSVM) denotes the set of rules $r_1, r_2 \ldots r_m$. R(rSVM) *steals* the classification from M as follows: if a sequence matches some rule r_i, classify the sequence as the positive class, otherwise, classify the sequence by M. An rSVM is required to satisfy three properties:

- *Performance*: R(rSVM) has a precision similar to that of M on D(test).
- *Significance*: R(rSVM) steals a large portion of classification from M. In other words, R(rSVM) shares a significant portion of recall.
- *Interpretability:* R(rSVM) contains a small number of simple rules.■

The intention is that the rule portion R(rSVM) captures simple and major structures, whereas M captures subtle structures that do not have simple rule representation. The performance requirement ensures that rSVM preserves the precision of the SVM, which also ensures that rSVM preserves the recall of the SVM because the sequences not covered by the rules are picked up by the SVM and those covered by the rules have a similar precision to the SVM. The significance and interpretability requirements ensure that the rules play active roles in manifesting a significant portion of classification. If the data does not have simple structures, R(rSVM) will be insignificant or empty and M will perform most or all classification. Our objective is to capture simple structures by rules whenever they exist. Likewise, if the structure is

mostly simple, we expect R(rSVM) to steal most or all classification, in which case M becomes insignificant.

We find a rSVM classifier in three phases. *SVM phase* maps training sequences to the feature space and builds the SVM classifier M using the standard software. *Rule phase* generates a set of high performance rules preserving the precision of M. *Stealing phase* determines the partition of classification between the rules and M. We explain each phase in details.

4. SVM PHASE

First, we find a set of frequent segments as the feature space. Frequent segments are mined from the positive sequences in D(train). To count the support of segments, we implemented the generalized suffix tree (GST) [33]. GST is the standard suffix tree modified for multiple sequences. A unique sequence id is appended to the end of each sequence, and the count stored at an internal node represents the number of *distinct* sequence ids in the sub-tree below the node, therefore, the support of the segment represented by the node. These counts can be obtained in a depth-first traversal of the standard suffix tree. The frequent segments are represented by the internal nodes having support above the minimum support.

To avoid losing useful features, a small minimum support should be used. We used the minimum support of 1% in all our experiments and it worked fine. SVM is quite robust in dealing with the high dimensionality and ignoring insignificant features (by assigning a small weight). In addition, our pruning strategies will prune insignificant features before the rule generation. Therefore, a small minimum support does not necessarily blow up the rule generation, but helps include potentially useful features.

Next, we map the sequences in D(train) to data points in the feature space, as described in Section 3.2, and apply the standard SVM software to produce the SVM classifier M, in particular, the weights w_i and the bias b. We use the *SVM-light*[2] implementation of SVMs in [16].

[2] http://svmlight.joachims.org

5. RULE PHASE

This phase generates statistically significant rules that preserve the precision of M. Before generating rules, we split D(train) into two disjoint parts randomly: D(build) and D(prune). We use D(build) for the rule generation and use D(prune) for making pruning decisions. We use the split of 4/5 vs. 1/5.

A major challenge in rule generation is the large search space due to tens of thousands of features. A limitation with most rule generation algorithms such as [3] and [6] is the heavy dependency on user-specified thresholds (i.e., minimum support) to prune rules. It is often difficult for the user to decide appropriate values for such thresholds, while this decision significantly impacts the performance of resulting classifiers. Note that we use minimum support in the SVM phase for feature generation, not rule generation, where the suffix tree algorithm is very efficient, and generated features are subject to the weighting by the SVM model and further pruning by our method. Instead of depending on user-specified thresholds, we present several strategies for pruning the search space of rule generation by exploiting the information provided by the SVM classifier M. We are interested in a rule only if it has at least the same precision as the SVM classifier M.

Definition 2. A rule *preserves* the SVM M if the precision of the rule is not less than the precision of M on the sequences in D(build) that match the rule. ■

If a preserving rule is "statistically significant", it will preserve the SVM precision over unseen sequences. We will consider a measure of statistical significance shortly. Unlike support in other works such as [3], our statistical significance is determined automatically, not by a threshold specified by the user.

Rule generation. The task is to find all statistically significant, preserving rules. To reduce the database scan, we search rules in a level-wise manner starting from shortest rules. Let

R_1 be the set of all size-1 rules $\{f_i\}$ over the features for which M has a non-zero weight w_i. Next, we extend every rule $\{f_i\}$ in R_1 by adding one feature f_j in R_1 to generate all size-2 statistically significant rules $\{f_i, f_j\}$, where $f_i \neq f_j$, denoted R_2. If a rule is not statistically significant, we will not include it in R_2 and not extend it further because any extended rule is not statistically significant. In general, at the *kth* iteration, we generate the size-(k+1) statistically significant rules, denoted R_{k+1}, using two rules $\{f_1, \ldots f_{k-1} f_k\}$ and $\{f_1, \ldots, f_{k-1} f_{k+1}\}$ in R_k. One scan of D(build) would check the statistical significance of generated rules. We continue this process until no statistically significant rule is generated. Then, we scan D(build) once to filter out all rules that do not preserve M. This rule generation can be expensive if the number of features is large. Below, we consider several strategies to prune the search space. The first two strategies are aimed at pruning features before the rule generation. The last two strategies are aimed at pruning rules during the rule generation.

5.1 Pruning redundant features

Our first observation is that if several features occur exactly in the same set of sequences in D(build), the rule generation is not able to distinguish them and we can remove all such features except one before the rule generation. A special case is that such features have sub-string/super-string relationships (e.g., “bc”, “abc”, “bcd”, “abcd”).

Strategy 1: For all features that occur exactly in the same set of sequences in D(build), we keep only one of them for the rule generation.

5.2 Pruning insignificant features

If a feature does not have a significant contribution, it can be pruned before the rule generation. One approach is to discard the features below some threshold of contribution, but this will get us into the trouble of determining a cutoff threshold. A better approach is using the information provided by M to prune insignificant features without any threshold.

Consider Equation (2). The further the weight w_i is from zero, the more influential the *ith* feature f_i is on the decision value $f(x)$. Therefore, we can sort the features f_i having non-zero w_i according to the influence $|w_i|$ into a list F and concentrate on the features in some prefix of F. (Other ranking criteria such as information gain can be used instead. But we believe that w_i is preferred because such weights are determined in the presence of all features.) To determine this prefix, for each prefix F', we consider the simplified SVM, denoted M', based on only the features in a prefix F'. M' is obtained from M by setting $w_i=0$ for all features f_i not in F'. Let E' be the error of M' on D(prune).

Strategy 2: Select the shortest prefix F' of F with the minimum E'.

In other words, F' is selected to minimize the error of the simplified SVM on D(prune). The use of D(prune) instead of D(build) is to avoid the over-fitting that tends to select the full list F.

ID	**Examples**	**Class**	**Predicted class**
		D(build)	
d1	f_1, f_2, f_3, f_5, f_8	+1	+1
d2	f_1, f_2, f_3	+1	+1
d3	f_1, f_3, f_6, f_7	+1	+1
d4	f_1, f_2, f_4, f_8, f_9	+1	-1
d5	f_1, f_4, f_5	-1	-1
d6	f_2, f_4, f_6	-1	-1
d7	f_3, f_4, f_7	-1	-1
d8	f_1, f_4, f_8, f_9	-1	-1
d9	f_3, f_4, f_6	-1	-1
d10	f_2, f_8	-1	-1
		D(prune)	

d11	f_1, f_3, f_4, f_6	+1	-1
d12	f_1, f_2, f_3, f_4, f_5	+1	+1
d13	f_2, f_3, f_4, f_6	-1	-1
d14	f_4, f_5, f_6, f_7	-1	-1
d15	f_2, f_3, f_4, f_7, f_8	-1	-1

Table 1. A sample data set

Example 1. We use D(train), split into D(build) and D(prune), in Table 1 to show Strategy 2. Ignore the last column at this moment. For simplicity, we show the features (i.e., $f_1, f_2, \ldots, f_9$) contained in each sequence instead of actual amino acids. Applying the software *SVM-light* to D(build), we obtain the SVM classifier M described by the weights w_i and the bias b in the table below, sorted by $|w_i|$. Consider the prefix $F'=\langle f_1 \rangle$. All the sequences in D(prune) are predicted as negative by the simplified SVM M' because $w_1+b<0$. So $E'=2$. Similarly, we can compute the error for all other prefixes. The shortest prefix with the minimum error is $\langle f_1, f_4, f_2, f_3 \rangle$, which has 1 error on *d11*. The remaining five features, i.e., f_5-f_9, are pruned from D(build) and D(prune). The classes predicted by this simplified SVM are listed in the column "Predicted class", which happens to be exactly the same as that of the original M based on all 9 features. ■

f_i	f_1	f_4	f_2	f_3	f_6
w_i	0.636	-0.581	0.554	0.554	0.118
f_i	f_7	f_5	f_8	f_9	b
w_i	0.118	0.036	0.018	0.018	-1.091

5.3 Pruning redundant rules

Consider two rules $r_1=\{f_1, f_2 \ \ldots f_k\}$ and $r_2=\{f_1, f_2 \ \ldots f_k, f_{k+1}\}$, where r_2 is obtained by extending r_1 with the feature f_{k+1}. If f_{k+1} is a sub-string of some feature in r_1, e.g., $r_1=\{$"*abcd*"$\}$ and $r_2=\{$"*abcd,* "*ab*"$\}$, the two rules will match exactly the same set of sequences, and we can

keep the shorter rule r_1 and prune the longer rule r_2. Now consider the case that f_{k+1} is a super-string of some feature in rule r_1, say f_1 without loss of generality. From the above discussion, we only need to consider $\{f_2, \ldots f_k, f_{k+1}\}$ instead of r_2. We summarize these observations in the next strategy.

Strategy 3. If a feature f_{k+1} has sub-string or super-string relationship with some feature in rule r_1, stop extending r_1 with f_{k+1}.

5.4 Pruning insignificant rules

Now we consider how to tell if a rule is statistically significant. A statistically significant rule should be accurate on the *whole population*, in addition to D(build). Suppose that a rule r_1 matches N_1 sequences in D(build), among which E_1 are classified wrongly. E_1/N_1 measures the observed error rate of r_1 on the sample D(build). We adopt the *pessimistic error estimation* used in [25] to estimate the real error rate of r_1 on the whole population. The idea is to regard these E_1 errors as observing E_1 events in N_1 trials, assuming that such events follow the binomial distribution. Given a confidence level CF, the probability that the real error rate of r_1 in the whole population exceeds the upper limit $U_{CF}(E_1,N_1)$ is no more than $CF/2$. The exact computation of $U_{CF}(E_1,N_1)$ is less important and can be found in the C4.5 code [25]. Generally, a smaller sample size N_1 is penalized by a larger upper limit $U_{CF}(E_1,N_1)$ to guarantee the specified confidence level CF. We take $U_{CF}(E_1,N_1)$ as the estimated error rate of r_1 on the whole population. The default value of CF in C4.5 is 25%.

Suppose that we extend the rule $r_1=\{f_1, f_2 \ldots f_k\}$ with E_1/N_1 to the rule $r_2=\{f_1, f_2 \ldots f_k, f_{k+1}\}$ with E_2/N_2. Note that $N_2 \leq N_1$. If $U_{CF}(E_1,N_1) \leq U_{CF}(E_2,N_2)$, we regard the rule r_2 as over-fitting, i.e., statistically insignificant, because it does not decrease the error rate on the whole population. Once r_2 is over-fitting, so is any rule obtained by extending r_2.

Strategy 4: If r_2 is an extension of r_1 such that $U_{CF}(E_1,N_1) \leq U_{CF}(E_2,N_2)$, stop generating r_2 and any extension of r_2 (because r_2 is statistically insignificant).

Example 2. Let CF be *25%*. Suppose that the features f_1, f_2, f_3, f_4 represent frequent segments "a", "b", "c", "ab", respectively. The rule $r_1=\{f_1\}$ matches 6 examples in D(build) in Table 1, with 2 being negative. So $N_1=6$, $E_1=2$, and the upper limit $U_{CF}(E_1,N_1)$ is estimated to be *0.56*. We extend r_1 into $r_2=\{f_1, f_2\}$. Note that f_2 has no sub-string or super-string relationship with f_1. r_2 matches 3 examples in D(build), all being positive. So $N_2=3$, $E_2=0$, and $U_{CF}(E_2,N_2)$ is estimated to be *0.37*. Since $0.37 < 0.56$, according to Strategy 4, rule r_2 is kept. Now consider extending r_2 to $r_3=\{f_1, f_2, f_3\}$. r_3 matches 2 examples in D(build), both being positive. So $N_3=2$, $E_3=0$, and $U_{CF}(E_3,N_3)$ is estimated to be *0.50*. Since $0.50 > 0.37$, rule r_3 is dropped and no further rule is extended from r_3. ■

6. STEALING PHASE

Let $R=\{r_1, r_2 \ldots r_k\}$ be the set of statistically significant, preserving rules found in the rule phase. Now we consider these rules collectively as a classifier. Some rules become redundant in the presence of other rules and can be pruned to improve the interpretability. Let us assume that the rules $r_1, r_2 \ldots r_k$ are sorted by the confidence on D(build), and in case of tie, sorted by support. To classify a sequence, the first matching rule in the list, if there is one, is applied because of higher confidence. Under this preference, rules towards the end of the list tend to classify fewer examples, therefore, have less contribution. We select a prefix of R for building the rSVM classifier.

Consider a prefix R' of R. Let $E(R')$ be the error of R' on the matching sequences in D(prune), and let $E(M,R')$ be the error of M on such sequences. $E(M,R')-E(R')$ measures the (possibly negative) performance gain of replacing M with R' over such sequences. To ensure that

as many sequences as possible are classified by rules, we select the longest prefix *R'* that maximizes *E(M,R')-E(R')*. Note that the selected prefix has a non-negative performance gain *E(M,R')-E(R')* because the empty prefix gives the zero performance gain. We then remove all the rules in the selected *R'* that classify no sequence in D(prune). Finally, we put *R'* on top of the SVM to construct the rSVM classifier:

R', M.

The ordering of rules in *R'* is no longer important because all rules predict the positive class.

Comment 1. The above selection criterion favors precision over recall, as motivated by the localization site prediction, because it maximizes *E(M,R')-E(R')*. However, we can easily adopt it to other selection criteria. In general, a selection criterion is maximizing a function of *E(M,R'), E(R'),* and *Sup(R')*, where *Sup(R')* is the fraction of the sequences in D(prune) classified by *R'*, i.e., the recall of *R'*. The user can explore a trade-off between precision and recall through specifying this function.

Comment 2. Another option is selecting a user-specified number of rules for building the rSVM. We did not consider this option because our primary goal is preserving the precision of the SVM classifier (while maximizing the recall). If the user selects too many rules, the claimed performance of SVMs will not be preserved. If the user selects too few rules, simple structures will not be fully extracted into if-then rules. Nevertheless, our method can be easily adapted to a specified number of rules if the user wishes to do so. Finally, our method may select more rules than what the user wants to see. But this is not a problem because the selected rules are presented in the sorted order and the user can decide a top portion of the list for viewing.

Prefix *R'*	The empty prefix	r_1	r_1, r_2
Examples classified in	none	*d11, d12*	*d11, d12, d13, d15*

D(prune)			
E(R')	0	0	2
E(M,R')	0	1	1
E(M,R')-E(R')	0	1	-1

Example 3. Continue on Example 2. Suppose that two rules, $r_1=\{f_1, f_3\}$ and $r_2=\{f_2, f_3\}$, are found in the rule phase. They have the same confidence in D(build), but the first rule has higher support. So, R in the sorted order is (r_1,r_2). The table above shows the classification of the examples in D(prune) by each prefix R'. The prefix $R'=(r_1)$ is selected because it is the longest prefix that maximizes $E(M,R')-E(R')$. The rSVM classifier is:

$$r_1, M.$$

On D(prune), r_1 correctly classifies the two positive examples (i.e., d11 and d12) and classifies no negative example as positive. In comparison, M incorrectly classifies one of the two positive examples (i.e., *d11*) as negative. Therefore, the use of r_1 has actually improved the performance of the SVM classifier on D(prune). In terms of interpretability, r_1 presents a more understandable structure of positive sequences than the SVM kernel function that involves 9 features. ■

7. EXPERIMENTS

The purpose of experiments is to evaluate the properties of rSVM classifiers and the effectiveness of various pruning strategies. We used the SVM-light[3] implementation [16] of SVMs, and compared the interpretability of rSVMs against the C4.5 classifier[4] [25], which is widely considered as accurate and understandable classifiers. For fair comparison, we chose the

[3] http://svmlight.joachims.org

[4] http://www.cse.unsw.edu.au/~quinlan

rule option C4.5 classifier that has fewer rules than the tree option. We used default settings in both systems and conducted experiments on a PC with 2.4G CPU and 1GB main memory.

The evaluation was conducted using two groups of membrane proteins from medically important disease-causing bacteria, namely, Gram-Negative bacteria [5] and Gram-Positive bacteria[6]. All proteins included in these data sets have been experimentally verified for their localization sites. Each group has several primary localization sites. One data set can be created by taking each primary localization site as the positive class and taking the remaining sites as the negative class. We chose the 5 data sets on which SVMs have at least 90% precision and 30% recall. The feature set was mined with the minimum support of 1% or 2 positive sequences, whichever is larger, and features of length less than 3 were discarded because they tended to occur in every sequence. Table 2 describes the data sets based on the average of the 5-fold cross validation. For example, the data set named "Neg-Inner" is from Gram-Negative bacteria and has "Inner membrane" as the positive class.

Data sets	# Seq.	# Pos. seq.	Seq. length	# Features	# Features per seq.
Neg-Inner	1572	292	410	34828	3817
Neg-Outer	1572	377	559	42079	3507
Neg-Extra	1572	191	469	115786	7904
Pos-Cellwall	576	61	1059	87727	9449
Pos-Extra	576	183	451	67381	3096

Table 2. Data statistics

[5] http://www.psort.org/dataset/datasetv1.html, Version 1.1. This is the version available at the time of experiments

[6] http://www.psort.org/dataset/, Version 2.0.

For comparison purpose, we considered several competing classifiers: SVM, Rule-alone, C4.5-prune and C4.5-all. SVM is the standard SVM classifier. Rule-alone is the rule list produced in the rule phase but cut off by minimizing the error on D(prune), with the negative class being the default class. Rule-alone serves the baseline for our rules without integration with SVMs. C4.5-all is the standard C4.5 classifier (of the rule option). C4.5-prune is the standard C4.5 classifier built using the feature set produced after pruning redundant and insignificant features as described in Section 5. We also followed [17] and built the C4.5 classifier using the top *p%* features (ranked by information gain), where p={1, 5, 10, 20, 50}. The results are either much worse or very close to C4.5-prune, so are not included here.

7.1 Performance and significance

Table 3 shows the precision/recall (in percentage) on D(test). rSVM preserves the precision and recall of SVM quite well. This is because the rule portion R(rSVM) has a precision comparable to the precision of SVM. Consequently, rSVM outperforms the C4.5 classifiers by a similar margin as SVM does. Rule-alone has a (slightly) higher precision than rSVM, i.e., 3%, because it was selected to minimize the error on the sequences it matches. However, this slight advantage is at the heavy expense of a much lower recall, i.e., 18% compared to 69% of rSVM. rSVM has at least as much recall as the SVM. This is a consequence of using positive rules only in rSVM: if a sequence is predicted as positive by the SVM, it will be predicted as positive by either R(rSVM) or the (same) SVM in the rSVM. Typically, the recall of rSVM is several percentage points higher than that of the SVM because additional structures were captured by rules.

The significance of the rSVM is measured by the portion of classification performed by rules, i.e., the recall of R(rSVM). This is shown under the R(rSVM) column in Table 3. The larger this recall is, the more classification is stolen by the rules and the more effective the rules are. Note that these rules are constrained to preserve the precision of SVM, so simply including

more rules in R(rSVM) does not help. On average, the recall of R(rSVM) is 30%, compared to the 69% recall of the rSVM. This means that about 43% of the classification (of the positive class) done by the rSVM was performed by rules, therefore, was manifested to the human user. As we will show shortly, these rules are quite compact and are understandable to the human user.

Data sets	**rSVM**		**SVM**	**Rule-alone**	**C4.5-prune**	**C4.5-all**
	R(rSVM)	**rSVM**				
Neg-Inner	97/41	98/88	99/85	98/17	61/51	60/43
Neg-Outer	100/22	98/81	98/80	100/15	70/72	69/68
Neg-Extra	96/23	93/44	94/42	100/17	60/49	58/45
Pos-Cellwall	91/37	89/68	91/67	90/19	57/46	50/43
Pos-Extra	93/28	92/68	96/54	98/23	55/48	50/41
Average	**95/30**	**94/69**	**95/65**	**97/18**	**60/53**	**57/48**

Table 3. Precision/Recall (%) on D(test)

Compared to the C4.5 classifiers, R(rSVM) is more than 36% more accurate in precision. This huge gain makes the rules of the rSVM more useful to the biologist, who wants to be damn sure that any prediction about the target localization is correct. Though the C4.5 classifiers have a larger recall, their quality is much less trusted, because of the significantly lower precision (i.e., 36% lower). Compared to Rule-alone, R(rSVM) is preferred due to the much higher recall (i.e., 12% higher) with only slightly lower precision (i.e., 2% lower).

We also compare with publicly available software tools TMHMM [48] and Phobius [49] that are primarily used to identify the presence and location of transmembrane helices in a protein. The presence of transmembrane helices indicates inner membrane proteins (also called the cytoplasmic membrane), and 3 or more transmembrane helices is a more reliable indication

[13]. Based on this property, TMHMM and Phobius produce Precision/Recall of 98/83 and 99/82 on the testing data of our 5-fold cross validation. While the precision is similar to that of rSVM and SVM, the recall is 3% to 6% lower. If we require only 2 or more transmembrane helices, these numbers are 94/91 and 95/88, and if we require only 1 or more transmembrane helices, these numbers are 66/96 and 87/96. Note that this method cannot predict the other localization sites where proteins do not necessarily contain transmembrane helices.

7.2 Interpretability

Table 4 compares x/y/z in R(rSVM), Rule-alone and C4.5-prune, where x is the number of rules, y is the average rule length, and z is the average feature length (C4.5-all has more rules than C4.5-prune, so is not included). The column "# Non-zero weight features in SVM" contains the number (and percentage) of non-zero weight features in the kernel function of the SVM classifier. R(rSVM) and Rule-alone have a rather small number of rules, i.e., 21 and 14 respectively, with short rules (i.e., average of 2.2 features per rule) and simple features (i.e., average of 5.5 amino acids per feature). Rule-alone has fewer rules than R(rSVM), but it comes with a much lower recall (see the above discussion). C4.5-prune uses much more rules, i.e., 50, and the features in these rules are much longer, i.e., the average of 25.8 amino acids per feature, than those in R(rSVM) and Rule-alone. These features were chosen by C4.5 because of high confidence in D(train), therefore, high information gain. But since these features have very low support, they did not perform well on D(test), which explains why C4.5-prune has a low precision (see Table 3). Our rule generation prunes rules containing such features due to statistical insignificance. The SVM classifier has tens of thousands of features in its kernel function even after removing all zero weight features. A complexity of this scale would bury any useful and simple structures that the biologist could use for further analysis and actions.

Data sets	R(rSVM)	Rule-alone	C4.5-prune	# Non-zero weight features in SVM
Neg-Inner	31 / 2.0 / 4.0	22 / 2.0 / 4.0	61 / 2.1 / 27.5	15176 (43.57%)
Neg-Outer	15 / 2.6 / 10.7	12 / 2.7 / 11.2	63 / 2.1 / 37.2	39895 (94.80%)
Neg-Extra	12 / 2.1 / 4.1	11 / 2.1 / 4.1	40 / 2.2 / 27.1	17902 (15.46%)
Pos-Cellwall	13 / 2.0 / 4.1	8 / 2.0 / 4.1	15 / 2.2 / 19.8	13345 (15.21%)
Pos-Extra	33 / 2.1 / 4.0	17 / 2.3 / 4.0	72 / 2.1 / 17.4	26591 (39.46%)
Average	**21 / 2.2 / 5.4**	**14 / 2.2 / 5.5**	**50 / 2.1 / 25.8**	**22581 (41.7%)**

Table 4. Comparison on interpretability

7.3 Pruning effectiveness

As shown in Table 2, there are tens and even hundreds of thousands of features, and each sequence contains more than 3000 features. Mining rules from such high dimensional data is extremely expensive and must rely on strong pruning strategies to reduce the search space. The column "Features kept" in Table 5 shows the percentages of features remaining at different stages, with respect to the initial number of features. The first number is the percentage of features after pruning redundant features (Strategy 1). The second number is the percentage of features after pruning those with zero weight in the SVM model. The third number is the percentage of features after pruning insignificant features (Strategy 2). Roughly speaking, almost 2/3 of features are redundant, 1/3 of non-redundant features have zero weight, and 1/3 of the remaining non-zero weight features are further pruned due to insignificance. As a result, the rules of R(rSVM) are searched using no more than 11% of the features that are used for training SVM. The column "Features per seq." in Table 5 shows that, by feature pruning (Strategy 1 and 2), the average number of features contained in a sequence is reduced to 2.1% of the number of

features in a sequence before the pruning. This significantly reduces the data size, the search space, and the rules generated.

Data sets	Features kept (%)	Features per seq. (%)
Neg-Inner	45.0 / 28.1 / 20.0	2.5
Neg-Outer	35.2 / 13.4 / 8.5	2.0
Neg-Extra	28.7 / 6.3 / 4.1	1.5
Pos-Cellwall	17.0 / 10.6 / 7.0	2.0
Pos-Extra	30.6 / 19.3 / 13.0	2.1
Average	**31.3 / 15.5 / 10.5**	**2.0**

Table 5. Effectiveness of feature pruning

With only 10% (of the features in Table 2) remaining after the feature pruning, the number of features ranges from 10^3 to 10^4. Without any rule pruning, the number of size-k rules is 10^{3*k} to 10^{4*k}. The maximum *k* for the rules in our rSVMs is 3. This amounts to the search space of 10^9 to 10^{12} rules if no rule pruning is done. Table 6 presents the number of rules generated at different phases in our algorithm. Compared to the above search space without rule pruning, the number of rules generated (denoted "# Significant" for statistically significant rules) is significantly reduced. Among the rules generated, about 1% to 10% are preserving rules (denoted "# Preserving"), and only about 0.01% are included in the final rSVM (denoted "# Final").

Data sets	# Significant	# Preserving	#Final
Neg-Inner	$6.0*10^6$	$5.6*10^4$	31

Neg-Outer	$8.7*10^6$	$1.9*10^5$	15
Neg-Extra	$1.1*10^6$	$1.2*10^4$	12
Pos-Cellwall	$5.1*10^6$	$8.6*10^4$	13
Pos-Extra	$3.4*10^6$	$2.5*10^4$	33

Table 6. Effectiveness of rule pruning

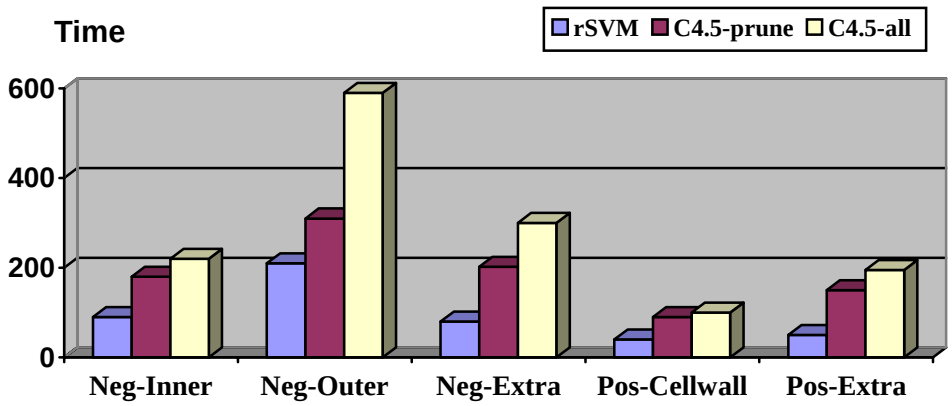

Figure 3. Time (in seconds)

Figure 3 compares the average CPU time (seconds) for building rSVM and C4.5 classifiers. The time for generating the feature set is the same for all algorithms and is not included. For rSVM, the time includes building the SVM model, rule generation and stealing phase. More than 70% of the time was spent on the rule phase. For this reason, the time for Rule-alone (not shown) is similar to the time for rSVM. For the C4.5 classifiers, the time includes building the decision tree and rule pruning. rSVM is more efficient than C4.5-prune. C4.5-all is too slow due to the high dimensionality of data.

8. CONCLUSION

Motivated by applications in antibiotic and vaccine drug design, we examined the subcellular protein localization problem for disease-causing bacteria. This problem has several demanding and conflicting requirements: high precision of prediction, interpretability of models, and high dimensionality of data. Our approach is integrating the precision-driven SVM model with the interpretable rule-based model, with each doing what they are best at. The SVM model focuses on classification involving subtle structures, whereas the rule-based model focuses on main structures that can be represented by concise if-then rules. The integrated model, called rSVM, *preserves* the performance of the SVM model and *exposes* simple structures in understandable rules. We addressed several technical issues in this integration: searching for precision-preserved rules, minimizing the dependency on user-specified thresholds or parameter tuning, splitting the classification between rules and SVM. The experiments on real subcellular protein localization tasks have demonstrated the effectiveness of rSVMs.

We conclude this paper with two observations. First, the presented approach makes no assumption about the internal structure of SVM other than providing the ranking of all features (i.e., the weight), which is used to guide the search of rules in the rule generation phase. Therefore, SVM can be replaced with any model that provides such ranking information. In particular, the basic SVM can be substituted with an improved variant and the improved accuracy will be preserved due to the preservation property. Second, though we have focused on the subcellular protein localization problem, the three requirements addressed, i.e., high precision, interpretability and high dimensionality, are universally found in many other problems and applications. For example, building an information directory requires not only correctly classifying text documents into relevant topics, but also annotating the topics with some summary information so that the directory can be browsed interactively by the information seeker. Our approach can be applied to extract such summary information as if-then rules for the target topic while preserving the striking performance of SVMs.

Acknowledgements. We like to thank Jennifer Gardy for helpful discussions and pointing out the Phobius tool to us. We also like to thank the reviewers for constructive comments and suggestions that have improved this paper.

9. REFERENCES

[1] R. Andrews, J. Diederich and A. Tickle. A survey and critique of techniques of extracting rules from trained artificial neural networks. *Knowledge-Base Systems*, 8(6), 373-389, 1995.

[2] J. Ayres, J. Gehrke, T. Yiu and J. Flannick. Sequential pattern mining using a bitmap representation. *SIGKDD*, 215-224, 2002.

[3] R. Agrawal, T. Imilienski and A. Swami. Mining association rules between sets of items in large datasets. *SIGMOD,* 1993.

[4] K. Ali, S. Manganaris and R. Srikant. Partial classification using association rule. *KDD,* 115-118, 1997.

[5] C.B. Anfinsen. Principles that govern the folding of protein chains. *Science* 181 (96), 223-230, 1973.

[6] R. Agrawal and R. Srikant. Fast algorithm for mining association rules. *VLDB,* 1994.

[7] R. Agrawal and R. Srikant. Mining sequential patterns. *ICDE, 1995.*

[8] A. Bairoch and P. Bucher. PROSITE: recent developments. *Nucleic Acids Research,* 22(17), 3583-3589, 1994.

[9] C.J.C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2, 121-167. Kluwer Academic Publishers, 1998.

[10] J. Cedano, P. Aloy, J.A. Perez-Pons and E. Querol. Relation between amino acid composition and cellular location of proteins. *Journal of Molecular Biology,* 266(3), 594–600, 1997.

[11] K. Diederichs, J. Freigang, S. Umhau, K. Zeth and J. Breed. Prediction by a neural network of outer membrane b-strand topology. *Protein Science,* 7, 2413-2420, 1998.

[12] F. Eisenhaber and P. Bork. Wanted: subcellular localization of proteins based on sequences. *Trends in Cell Biology,* 8, 169-170, 1998.

[13] J.L. Gardy, C. Spencer, K. Wang, M. Ester, G.E. Tusnady, I. Simon, S. Hua, K. deFays, C. Lambert, K. Nakai and F.S.L. Brinkman. PSORT-B: improving protein subcellular localization prediction for Gram-negative bacteria. *Nucleic Acids Research,* 31(13), 3613-3617, 2003.

[14] S. Hua and Z. Sun. Support vector machine approach for protein subcellular localization prediction. *Bioinformatics,* 17(8), 721-728, 2001.

[15] I. Jacoboni, P. Martelli, P. Fariselli, V. De Pinto and R. Casadio. Prediction of the transmembrane regions of β-barrel membrane proteins with a neural network-based predictor. *Protein Science,* 10, 779-787, 2001.

[16] T. Joachims. Making large-scale SVM learning practical. *Advances in Kernel Methods – Support Vector Learning*. MIT Press, 1998.

[17] T. Joachims. Text categorization with support vector machines: learning with many relevant features. *In Proceedings of the European Conference on Machine Learning*. Springer, 1998.

[18] J.R. Koza and D. Andre. Automatic discovery of protein motifs using genetic programming. *Evolutionary Computation: Theory and Applications*. Singapore: World Scientific.

[19] J.R. Koza, F.H. Bennett III and D. Andre. Classifying proteins as extra-cellular using programmatic motifs and genetic programming. *Proceedings of the IEEE World Congress on Computational Intelligence*, 1998.

[20] B. Liu, W. Hsu and Y. Ma. Integrating classification and association rule mining. *KDD,* 1998.

[21] P. Martelli, P. Fariselli, A. Krogh and R. Casadio. A sequence-profile-based HMM for predicting and discriminating β barrel membrane proteins. *Bioinformatics,* 18(1), S46-S53, 2002.

[22] H. Nunez, C. Angulo and A. Catala. Rule extraction from support vector machines. *In Proceedings of European Symposium on Artificial Neural Networks,* 2002.

[23] J. Nakashima and K. Nishikawa. Discrimination of intercellular and extra-cellular proteins using amino acid composition and residue-pair frequencies. *Journal of Molecular Biology,* 238, 54 – 61, 1994.

[24] J. Han, B. Asl, Q. Chen, U. Dayal and M. Hsu. Freespan: Frequent pattern-projected sequential pattern mining. *SIGKDD*, 355-359, 2000.

[25] J.R. Quinlan. C4.5: *Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA, 1993.

[26] I. Rigoutsos and A. Floratos. Combinatorial pattern discovery in biological sequences: the TEIRESIAS algorithm. *Bioinformatics*, 14(1), 55-67, 1998.

[27] A. Reinhardt and T. Hubbard. Using neural networks for prediction of the subcellular location of proteins. *Nucleic Acids Research*, 26(9), 2230-2236, 1998.

[28] G. Salton and C. Buckley. Term weighting approaches in automatic text retrieval. *Information Processing and Management*, 24(5), 513-523, 1988.

[29] R. She, F. Chen, K. Wang, M. Ester, J.L. Gardy and F. Brinkman. Frequent subsequence-based prediction of outer membrane proteins. *SIGKDD*, 2003.

[30] L. Stryer. *Biochemistry*. 4th Edition. W.H. Freeman, NY, 1995.

[31] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer, NY, 1995.

[32] J.P. Vert. Support vector machine prediction of signal peptide cleavage site using a new class of kernels for strings. *Proceedings of the Pacific Symposium on Biocomputing,* 649-660, 2002.

[33] J. Wang, G. Chirn, T. Marr, B.Shapiro, D. Shasha and K. Zhang. Combinatorial pattern discovery for scientific data: some preliminary results. *SIGMOD*, 1994.

[34] K. Wang, S. Zhou and Y. He. Growing decision tree on support-less association rules. *SIGKDD,* 2000.

[35] Z. Yuan. Prediction of protein subcellular locations using Markov chain models. *FEBS Letter,* 451, 23-26, 1999.

[36] J. Yang, W. Wang, P.S. Yu, J. Han. Mining long sequential patterns in a noisy environment. *SIGMOD*, 406-417, 2002.

[37] M. J. Zaki. SPADE: An efficient algorithm for mining frequent sequences. *Machine Learning Journal, Special Issue on Unsupervised Learning*, 42(1/2), 31-60, 2001.

[38] A. Bairoch and B. Boeckmann. The SWISS-PROT protein sequence data bank: current status. *Nucleic Acids Research,* 22(17), 3578-3580, 1991.

[39] T. Schirmer and S. Cowan. Prediction of membrane-spanning β-strands and its application to maltoporin. *Protein Science*, 2, 1361-1363, 1993.

[40] W. Wimley. Toward genomic identification of β-barrel membrane proteins: Composition and architecture of known structures. *Protein Science*, 11, 301-312, 2002.

[41] Y. Zhai and M. Saier. The β-barrel finder (BBF) program, allowing identification of outer membrane β-barrel proteins encoded within prokaryotic genomes. *Protein Science*, 11, 2196-2207, 2002.

[42] P. Martelli., P. Fariselli., A. Krogh, and R. Casadio. A sequence-profile-based HMM for predicting and discrimating β barrel membrane proteins. *Bioinformatics*, 18(1) 2002, S46-S53, 2002.

[43] Z. H. Zhou and Z. Q. Chen. Hybrid decision tree. *Knowledge-based systems*, 15(8), 515-528, Elsevier, 2002.

[44] C. Leslie, E. Eskin, W. S. Nobel. The spectrum kernel: a string kernel for SVM protein classification. *Proceedings of Pacific Symposuim on Biocomputing*, 564-575, 2002.

[45] N. Barakat and J. Diederich. Learning-based rule-extraction from support vector machines. *12th International Conference on Computer Theory and Applications*, 2004.

[46] H. Yu, K.C.C. Chang, J. Han. Heterogeneous learner for web page classification. *ICDM* 2002.

[47] K. P. Bennett, N. Cristianni, and D. Wu. Enlarging the margins in perceptron decision trees. *Machine Learning*, Vol. 41, 295-313, 2000.

[48] TMHMM Server v. 2.0, Prediction of transmembrane helices in proteins, http://www.cbs.dtu.dk/services/TMHMM/

[49] Phobius, A combined transmembrane topology and signal peptide predictor, http://phobius.cgb.ki.se/index.html

[50] K. Wang, Y. Xu, J. Yu. Scalable sequential pattern mining for biological sequences. *CIKM*, 2004.

[51] I. Guyon, J. Weston, S. Barnhill and V. Vapnik. "Gene selection for cancer classification using support vector machines." Machine Learning. Vol. 46 , Issue 1-3, 2002 , 389 - 422