Multivariate Time Series Classification with Temporal Abstractions

Iyad Batal
Department of Computer Science
University of Pittsburgh, PA
iyad@cs.pitt.edu

Lucia Sacchi and **Riccardo Bellazzi**
Dipartimento di Informatica e Sistemistica
University of Pavia, Italy
{lucia.sacchi, riccardo.bellazzi}@unipv.it

Milos Hauskrecht
Department of Computer Science
University of Pittsburgh, PA
milos@cs.pitt.edu

Abstract

The increase in the number of complex temporal datasets collected today has prompted the development of methods that extend classical machine learning and data mining methods to time-series data. This work focuses on methods for multivariate time-series classification. Time series classification is a challenging problem mostly because the number of temporal features that describe the data and are potentially useful for classification is enormous. We study and develop a temporal abstraction framework for generating multivariate time series features suitable for classification tasks. We propose the *STF-Mine* algorithm that automatically mines discriminative temporal abstraction patterns from the time series data and uses them to learn a classification model. Our experimental evaluations, carried out on both synthetic and real world medical data, demonstrate the benefit of our approach in learning accurate classifiers for time-series datasets.

Introduction

Data classification is an important machine learning problem with a wide variety of applications. A number of classification models and techniques have been proposed and applied to the analysis of various dataset. These include methods such as decision trees, Bayesian networks, nearest-neighbor classifiers, or support vector machines. However, the success of classification methods depends heavily on the quality of data and data features used by the models. Consequently, feature selection and feature construction methods in combination with a classification model often determine the success of the machine learning approach in extracting useful and accurate classification models.

Advances in data collection and data storage technologies have led to emergence of complex multivariate datasets where examples are not simple data points; but instead the examples are traces of complex behaviors characterized by time series. Take for example medical datasets. The patient's electronic medical record provides complex temporal characterization of the patient case, including the sequences of lab values, observations, actions and related responses.

Complex temporal datasets pose numerous data-analysis challenges. Perhaps the most critical is the problem of temporal characterization of examples in data such that the most salient features important for the classification task are preserved. As an example, consider a problem of building a classifier to diagnose or predict the patient's condition using past patient's data. Ignoring the temporal aspect of data, the patient case can be relatively easily described using the most recent set of values, e.g. "a low blood pressure", or "a high white blood cells count". However, this information may be limited in describing the patient's state. For example, the information important for the diagnosis may include simple trends, such as "increase in the blood pressure" or more complex temporal patterns such as "low blood pressure following the prescription of a certain medication". Clearly, more complex temporal information may improve our ability to better diagnose the case. Similar temporal classification tasks may arise in other domains with time series data such as speech and gesture recognition, stock market analysis, intrusion detection, or industrial plants monitoring.

The caveat of dealing with temporal information is that there are many different temporal patterns to consider and only a very small fraction of these may be useful for classification. Hence, techniques capable of identifying temporal features useful for classification are badly needed. The objective of our work is to develop methods that let us automatically generate temporal features for classification of multivariate time series data. To model many possible temporal relations we adopt and rely on the temporal abstractions framework (Shahar 1997). The framework provides a qualitative description of time series using value and trend abstractions, and their combinations using temporal logic relations.

To identify temporal abstraction patterns most useful in predicting the class labels of the series we propose the *STF-Mine* (Segmented Time series Feature Mine) algorithm. The algorithm builds upon the *Apriori* algorithm (Agrawal and Srikant 1994) used in frequent pattern mining and extends it to the problem of mining frequent temporal abstraction patterns. After identifying the most frequent temporal patterns for each class, *STF-Mine* selects those patterns that are highly discriminative for target classes, and uses them to define a new feature space for representing the temporal data.

The rest of the paper is organized as follows. Section 2 explains our temporal abstraction methodology. We start by illustrating how time series are converted into state sequences using segmentation. Next, we give a formal definition of the temporal abstraction patterns used in our work. After that,

we explain how the *Apriori* algorithm can be modified to mine the frequent temporal patterns. Lastly, we explain our criteria for selecting the discriminative temporal pattern that will be used by the classifier. In section 3, we present the experimental evaluation of our approach. Finally, we conclude in section 4.

Methodology

A time series is a series of observations over a period of time. When the observed space is multidimensional, the time series becomes multivariate. Multivariate time series classification is a supervised learning problem aimed at labeling multivariate series of variable length. In our approach, we reduce this complex data to a simple feature-vector representation by suitably transforming time series into vectors of fixed length. After this transformation, we apply any of the existing classification methods to learn and predict the class of future time series examples.

In a nutshell, the proposed *STF-Mine* algorithm starts from the raw time series and passes through different stages of representation, which lead to the extraction of classification features. *STF-Mine* takes as input pre-classified training time series and it outputs a set of frequent and discriminative patterns. These patterns are then used to map each example into a Boolean vector and the classifier is trained on this new representation.

Our methodology consists of the following steps:

1. Segment the time series to obtain a qualitative description of each series.
2. Generate the frequent patterns from the segmented series of each class.
3. Select the frequent patterns that are highly discriminative between the classes.
4. Transform the data into a vector format, where the features are the patterns extracted in step 3; then learn the classifier on the transformed data.

The following subsections explain these steps in detail.

Qualitative Time Series Abstraction

In order to mine the frequent and discriminative patterns from data, we first need to obtain a qualitative representation of the raw time series data. Given a multivariate time series instance, the time series of each variable is converted into a sequence of abstract states s_i, where each state represents a property that holds during an interval $[b_i, e_i]$. The alphabet Σ represents all possible values for the states. For example, temporal trend abstraction, which allows to describe the time series in terms of its local trends, uses $\Sigma = \{increasing, decreasing, steady\}$.

The transformation of the raw time series data to temporal abstractions is not straightforward and more than one solution may exist. In this work, we use (1) value abstractions (with values *high*, *normal* and *low*) and (2) trend abstractions (with values *increasing*, *steady* and *decreasing*) as basic temporal patterns. For trend abstractions, we segment and label the series by using the sliding window method (Keogh et al. 2003), which keeps expanding each segment until its error exceeds a user-specified threshold. The trend is then defined by the slope of the fitted segment, where each of the three trend values (increasing, decreasing and steady) corresponds to a different slope range.

Temporal Patterns

In order obtain a temporal description of the multivariate data, basic temporal abstractions (extracted from each variable in the previous step) are combined to form complex temporal patterns. For example, a domain expert may want to describe a pattern in a time series data such as: "an increase in variable X is followed by a decrease in variable Y". This is the idea behind the temporal abstraction framework (Shahar 1997). To construct these more complex patterns, we first need to specify the temporal relations between state intervals.

Temporal relations: Allen's temporal logic (Allen 1984) describes the relations for any pair of intervals using 13 possible relationships; these relations are illustrated in Figure 1.

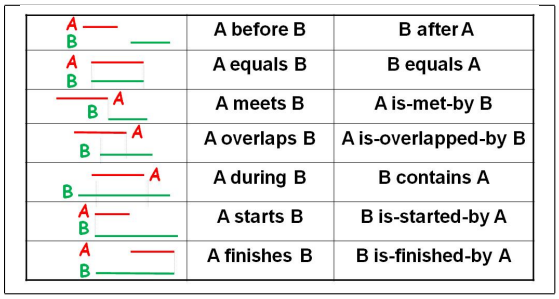

Figure 1: Allen's interval relationships

However, we believe that patterns from noisy interval data expressed using Allen's interval relations may not be robust. The reason for this is that most of the relations require equality of two or more interval end points. Thus, there is only a slight difference between overlaps, finishes, equal and during relations. In most real world datasets, information about time is not very precise and imposing such restrictions on the relations may hinder the discovery process. In addition, the result of the segmentation step is usually not very precise and can create some time misalignment. Therefore, we choose to model only two relationships: *before* and *overlaps*, which we redefine as follows:

Given two states, A and B with intervals $[a_1, a_2]$ and $[b_1, b_2]$:

- A *before* B iff $a_2 <= b_1$
- A *overlaps* B iff $a_1 <= b_1$ and $a_2 > b_1$, i.e. A starts before B and their intervals overlap.

Practically, in the definition of the *before* relation, we may want to impose a restriction on the length of the gap between a_2 and b_1. Later, we will see that this can be handled by defining the sliding window size of the mining algorithm.

After defining the interval relationships: REL=$\{before, overlaps\}$, we can define the temporal pattern recursively as follows:

- If X is a single state, then X is a temporal pattern.

- if X and Y are two temporal patterns, and $rel \in REL$, then $X\ rel\ Y$ is also a temporal pattern.

Using this simple definition, we see that a temporal pattern is defined as sequence of states (intervals) related using temporal relationships. We denote each pattern by a pair of vectors (S,R) where S_i corresponds to the ith state of the pattern, and R_i to the temporal relation between state S_i and state S_{i+1}: $R_i = rel(S_i, S_{i+1}) : rel \in REL$. We define the size of a pattern P to be the number of states it contains. If size(P)=k, we say that P is a *k-pattern*. In our notation, the symbol between brackets refers to the variable (time series) from which the state has been extracted. To give an example, assume that the data is two dimensional time series, where each instance has two series X and Y. Assume we are using trend abstractions. An example of a temporal 2-pattern could be: P=*increase*[X] *before decrease*[Y], which corresponds to an increase in variable X followed by a decrease in variable Y.

Interesting patterns are usually limited in their temporal extensions, i.e. we would not be interested in finding correlations between events far away from each other. Hence, we assume the user can specify the maximum pattern duration that is of interest, which is known as the window size (w). This w parameter serves as the width of the sliding window which is moved along the state sequences. The algorithm only considers the patterns that can be observed within this window. If the user does not specify w, the algorithm treats the whole time series as a single window. This case could be of interest when the time series data are short.

Let T denotes a multivariate time series instance after it is converted into a sequence of states, and E denotes the states of T visible within w. We say that the pattern P(S,R) holds in T within w, denoted as $P \in (T, w)$, if there is an injective mapping π from the states of P to the states of T such that:

$$\forall i \in \{1..size(P)-1\} : S_i = E_{\pi(i)} \text{ and } S_{i+1} = E_{\pi(i+1)} \text{ and } R_i = rel(E_{\pi(i)}, E_{\pi(i+1)})$$

We define the function $exists(P, T, w)$ as: $exists(P, T, w) = 1$ if $P \in (T, w)$ for at least one sliding window position of size w in T. Otherwise, $exists(P, T, w) = 0$.

Now, we define the support of a temporal pattern P in a time series database D using window size w as:

$$sup(P, D, w) = \sum_{\forall T_i \in D} \text{exists}(P, T_i, w)$$

In other words, the support of P is defined to be the number of instances from D where P can be observed within the window w.

Frequent Temporal Pattern Mining

In this section we describe the algorithm that mines the frequent temporal patterns from the multivariate state sequences (obtained using temporal abstraction). The algorithm is based on the *Apriori* algorithm proposed by (Agrawal and Srikant 1994) and is applied to each class of examples separately.

The algorithm relies on the *Apriori property* to reduce the search space. Let TP be the space of all temporal patterns of arbitrary size. The *Apriori property* in this context can be defined as:

$$\forall P, Q \in TP, \quad \text{if } P \subseteq Q \Rightarrow sup(P, D, w) \geq sup(Q, D, w) \text{ for any } D, w$$

that is, the support of a pattern is always less than or equal to the support of any of its subpatterns.

The frequent mining algorithm takes three inputs: 1) D_{c_i}: the set of all multivariate instances belonging to a specific class (c_i), 2) *min_sup*: a user defined threshold on the support of frequent patterns and 3) w: the sliding window width (the maximum pattern duration). The algorithm outputs all temporal patterns P_j where $sup(P_j, D_{c_i}, w) >=$ *min_sup*.

The algorithm performs level-wise search in order to find all frequent patterns. First, the set of frequent *1-pattern* (single state) is found by scanning the database. In the kth pass, the candidates that missed the minimum support threshold are removed and the candidate *(k+1)-patterns* are created from the remaining frequent *k-patterns*. This procedure is repeated until no more frequent patterns can be found. The *Apriori property* guarantees that the algorithm will not miss any frequent patterns.

Candidate Generation A candidate *(k+1)-pattern* is generated by joining two frequent *k-patterns* which share the same $k-1$ states as a prefix. Let us assume that we are joining the frequent patterns P_1 and P_2 to form the next level candidates. Let us denote the last states of P_1 and P_2 as a and b, respectively. The candidates generated from P_1 and P_2 will be completely specified by the relation between a and b. Since there are 4 possible relations that can relate the two states (a *before* b, a *overlaps* b, b *before* a and b *overlaps* a), joining P_1 and P_2 can potentially generate 4 possible candidates.

However, we do not have to generate all four candidates in every join. For instance, if the last states a and b are extracted from the same series, then they cannot be related by *overlap* relation. Besides, if the relation between a and the common prefix is *overlap* and the relation between b and the common prefix is related *before*, then we know that b should start after a. Thus we do not generate the candidates where a appears at the end. Furthermore, we can speed up the algorithm by pruning some of the candidates that contain infrequent subpatterns. To better illustrate candidate generation and pruning, we use the following example.

Example (Figure 2):
Let I, D, and S denote trend abstractions: *increasing*, *decreasing* and *steady*, respectively. Let b and o denote the *before* and *overlaps* relations.

Assume we have the following three frequent *2-patterns*: P_1=I[X] b D[Y], P_2=I[X] o I[Z], P_3=I[Z] o D[Y]. Since only P_1 and P_2 share a common prefix, they can be joined to generate the candidate *3-patterns*. However, we know that D[Y] should appear after I[Z] in the generated patterns. Thus, we only consider the two candidates C_1=I[X] o I[Z] o D[Y] and C_2=I[X] o I[Z] b D[Y]. Because the subpattern I[Z] b D[Y] contained in C_2 is not a frequent *2-*

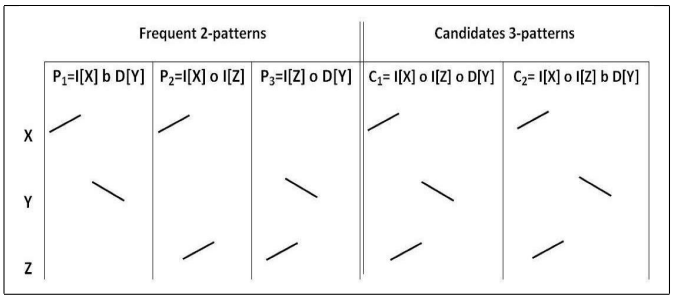

Figure 2: An example of candidate generation and pruning

pattern, C_2 cannot be a frequent *3-pattern* (according to the *Apriori property*). However, we cannot prune the candidate C_1 since the relation between $I[X]$ and $D[Y]$ is not specified (it could be either *before* or *overlaps*). Therefore, C_1 is the only candidate that survives the pruning stage.

Discriminative Pattern Selection

Our ultimate goal is to select temporal patterns (features) that let us discriminate well among target time-series classes. In other words, we seek patterns that are more frequent in one class and less frequent in other classes.

So far our algorithm has extracted frequent patterns for every class of time-series examples. However, if we keep all of these frequent patterns as features and feed them into the classification model, the high dimensionality can easily cause the classifier to overfit the training data. To alleviate the problem, we want to select only a small number of temporal patterns which are good predictors of the class label. We use the Pearson's chi-square (χ^2) to select predictive temporal patterns. χ^2 test measures the correlation between the pattern and class variable. We define the χ^2 statistics for pattern P_k as:

$$\chi^2(P_k) = \sum_{c\in\{C_1,..C_j\}} \sum_{p\in\{P_k,\bar{P_k}\}} \frac{(Pr(p,c) - Ex(p,c))^2}{Ex(p,c)}$$

$$Pr(p,c) = \frac{\#(p,c)}{N}, Ex(p,c) = \frac{\#(p)}{N} * \frac{\#(c)}{N}$$

In the equation, N is the total number of instances in the dataset. The symbol $\#$ represents counts. For example, $\#(P_k, C_j)$ is the number of instances from class C_j that contain the pattern P_k and $\#(\bar{P_k}, C_j)$ is the number of instances of C_j that do not contain P_k.

Notice that the entries which contribute the most to the χ^2 value are those whose actual probabilities are very different from the expected probability under the independence assumption.

Finally, we rank all frequent patterns according to their χ^2 values and we select a small set of these patterns be the features of our classifier.

Building the classifier

The purpose of this step is to convert the multivariate time series data into a feature-vector format, in order to be used with standard classification algorithms (such as SVM, Naïve Bayes, decision trees, or the linear discriminant analysis).

To do this, we map every instance T_i into a Boolean vector V_i of size equal to the total number of the temporal features extracted in the previous step. Each element $V_{i,j}$ in the vector corresponds to a specific pattern P_j and its value is set to 1 if P_j is present in T_i; and set to 0 otherwise. i.e. $V_{i,j}$=exists(T_i,P_j,w). These Boolean vectors are given to the classifier to build the classification model.

Experimental evaluation

In this section, we present results of our approach on both synthetic and real world data. We test the features extracted by the *STF-Mine* algorithm using both Naïve Bayes (NB) and SVM (Vapnik 1995) with linear kernel. We use the Maximum likelihood approach for learning the parameters of the NB model (Pedro and Pazzani 1997).

All the experiments follow a 20 fold cross validation scheme. To eliminate any classification bias, the temporal features are always selected from the training sets, that is, features extracted in different folds may be different.

Synthetic dataset

We chose to first experiment with synthetic data because it allows us to better understand the relationship between the classification accuracy and the characteristic of the data. The main objective of these experiments is to test the algorithm's ability to extract the classification features from state sequences (assuming the segmentation step was already done).

We chose to test the algorithm on a two dimensional dataset, where every instance contains two variables X and Y. Each of these variables is a sequence of 6 segments of equal length. The states of the segments are defined from the alphabet Σ=$\{A,B,C,D,E\}$.

We test our algorithm for binary classification, so we define the two classes C_1 and C_2 and we generate 50 instances from each of them. In the beginning, we randomly generate all the sequences in both classes from the 5 states. We then define two patterns: P_1=$A[X]$ *before* $B[Y]$ with a gap of zero between the two intervals (the segments touch each other), and P_2=$B[X]$ *before* $C[Y]$. These patterns will be artificially injected in class C_1 and C_2, respectively. We define the injection probability for a pattern P in a specific class to be the probability of injecting P in each instance of that class.

We designed two sets of experiments:

1- First, the instances of C_2 are randomly generated, and we varied the injection probability of P_1 in C_1.

2- Second, we inject P_2 in C_2 with probability of 0.4 and again we varied the injection probability of P_1 in C_1.

Because in the beginning the series of all instances are generated randomly, the probability that any instance will contain pattern P_1 by chance is 0.2. The reason is that the probability that P_1 occurs at a specific position is 1/5*1/5 (since there are 5 different states for each segment). There are 5 different slots where P_1 can appear in the instance (since there is 6 segments per variable). Therefore, the probability that P_1 occurs randomly in an instance is 5*1/5*1/5=0.2. The same random probability applies on P_2. Thus, we expect each of P_1 and P_2 to occur randomly in about 20% of the instances.

Results:
For the reported results, we set the sliding window size (w) to be two segments (we only count patterns that occur in consecutive segments). We set *min_sup* to be 25% of the number of instances in the class. We select the top 10 patterns (according to χ^2) to be the features of the classifier.

Figures 3 and 4 show the accuracy of SVM and NB for different injection probability of P_1 in C_1. Each graph has two plots: one where the instances of C_2 are random and one where we inject P_2 in C_2 with probability 0.4.

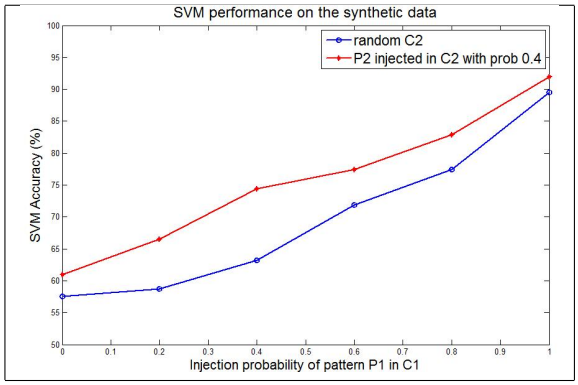

Figure 3: SVM performance using different injection probabilities of P_1 in C_1

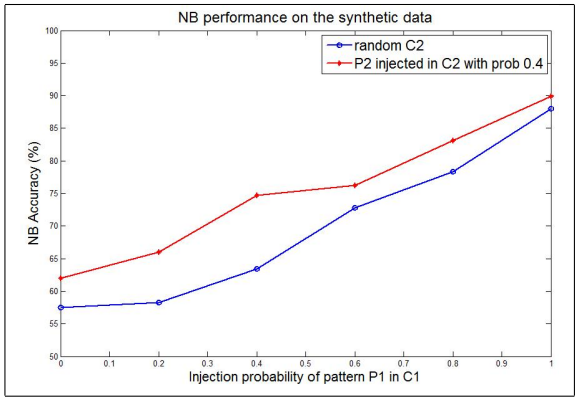

Figure 4: Naïve Bayes performance using different injection probabilities of P_1 in C_1

From the graphs, we can notice a similar behavior for both SVM and NB classifiers. As we expected, increasing the injection probability results in a higher classification accuracy.

HIT dataset

Heparin-induced thrombocytopenia (HIT) is a transient prothrombotic disorder induced by Heparin exposure with subsequent thrombocytopenia and associated thrombosis. HIT is a condition that is life-threatening if it is not detected and managed properly. The presence of HIT is tested by a special lab assay: Heparin Platelet Factor 4 antibody (HPF4).

Our objective in this experiment is to automatically learn from the data when an HPF4 test should be ordered for a patient under Heparin. In other words, given a specific point in time of a specific patient, which we refer to as *the anchor point*, we want to detect whether this patient start to exhibit the HIT symptoms, which requires an order of HPF4. One of the challenges in this type of medical data is that the lab results are usually **not regularly sampled in time**!

For our experiments, we include 220 patients for which the HPF4 test was ordered and set the anchor point to be the time HPF4 was ordered. We then choose 220 patients treated by Heparin, but who did not have an HPF4 test to be the controls. We set the anchor point randomly by the arrival of new platelet result, a key feature used in the HIT detection. We mix the cases and controls and test whether *STF-Mine* can help us deciding if HPF4 should be ordered for a specific patient at a specific point in time.

For every patient, we consider the platelet counts and the Hemoglobin test time series to extract the classification features. We consider two types of temporal abstractions: trend abstractions (increase, decrease, steady) and value abstraction (low, normal, high). The value abstractions are defined by the normal value range for the lab test. For platelets, the normal range is $[156 - 369] * 10^9$. For Hemoglobin, the normal range is $[12.9 - 16.9]$. Figure 5 shows a synthetic platelet series and the corresponding trend and value abstractions.

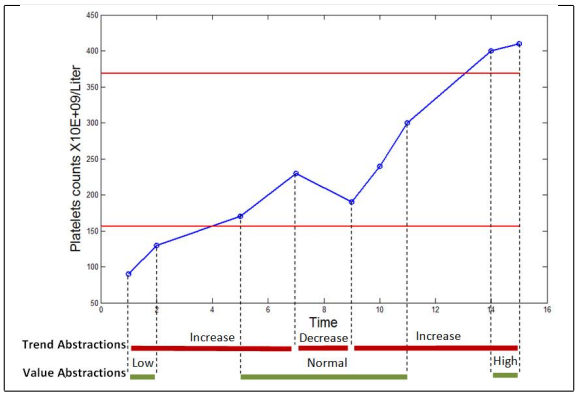

Figure 5: An example illustrating trend and value abstractions

Using the two abstractions, we obtain 4 state sequences from the platelets and hemoglobin series. So, we can see our data as a 4 dimensional data. Temporal pattern can be created across all of these dimensions. An example of a temporal pattern is: "decrease[platelets] overlaps low[platelets] before steady[hemoglobin]".

Results:
In this task, the recent values are the most important for classification. Thus, we consider the patient's data for the last 5 days before the anchor point. We set the window size to be 5 days, i.e. we extract the patterns from the whole 5 days data without sliding a window. We set *min_sup* to 10%, and again select the top 10 patterns for classification.

Since the vast majority of time series classification methods (Rabiner 1989; Bengio 1995; Xi et al. 2006) can only handle regularly sampled time series, we compare our approach against the baseline classifier that uses only the last two values of Platelets and Hemoglobin.

Using *STF-Mine* features, we were able to achieve an SVM accuracy of = 0.7944 and an NB accuracy of 0.7822. Figure 6 compares the ROC curves of the SVM classifier obtained using our temporal features with the baseline SVM

classifier. This ROC is the average of all the ROCs obtained from the 20 folds. It can be seen that the temporal features improve the quality of the classification model as compared to using only the most recent values.

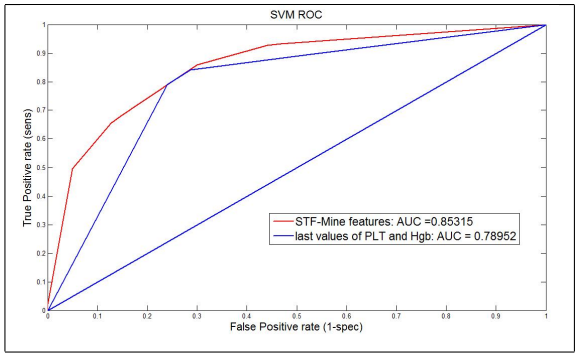

Figure 6: Comparing the ROC of SVM using the *STF-Mine* features and using the last two values of platelets and Hemoglobin

For significance test on the differences in AUCs, we run 20 folds cross validation 3 times. The p-value obtained was p=0.0083 for the independent t-test and p=0.12 for the cross-validation corrected t-test by (Nadeau and Bengio 2003).

We have obtained similar results when baseline and temporal features were tested with the Naïve Bayes classifiers (the AUC for baseline features was 0.76, and for the *STF-Mine* features it was 0.84). All these suggest the two sets of features differ in their classification performance and our temporal features are beneficial for the classification task.

Conclusion

Time series classification is becoming more and more important with the explosive increase in the number of time series databases. Traditional approaches for time series analysis and classification either use simple data features ignoring the temporal aspect of data or rely on hand-built temporal features defined a priori by a domain expert. However, vast amounts of data often prevent the expert from being able to grasp all possible patterns in the data. We have developed a new method that adopts the temporal abstraction framework to extract temporal features critical for the classification task. Our methodology integrates two powerful data mining paradigms: frequent patterns mining and inductive classification to solve the complex time series classification problem. An advantage of our methods is that it can work with irregularly sampled temporal datasets.

Many research issues related to our methodology remain open and may be further refined in the future. For example, our current approach trims the frequent patterns mined for different classes only after the frequent pattern generation process is complete. A more efficient solution may be possible by interleaving the feature generation and discriminative feature filtering steps.

Acknowledgements

This research was supported by the grant 1R21LM009102-01A1 from the National Library of Medicine.

References

Agrawal, R., and Srikant, R. 1994. Fast Algorithms for Mining Association Rules in Large Databases. In *Proceedings of VLDB*.

Agrawal, R., and Srikant, R. 1995. Mining sequential patterns. In *Proceedings of ICDE*.

Agrawal, R.; Faloutsos, C.; and Swami, A. 1993. Efficient Similarity Search in Sequence Databases. In *Foundations of Data Organization and Algorithms*.

Allen, F. 1984. Towards a general theory of action and time. *Artificial Intelligence, 23:123-154*.

Antunes, C., and Oliveira, A. 2001. Temporal Data Mining: an Overview. In *Proceedings of Workshop on Temporal Data Mining (KDD)*.

Bengio, Y. 1995. Neural Networks for Speech and Sequence Recognition. In *Intl. Thomsom publishing Inc*.

Das, G.; Lin, K.; Mannila, H.; Renganathan, G.; and Smyth, P. 1998. Rule Discovery from Time Series. In *Proceedings of ACM SIGKDD*.

Hoppner, F. 2001. Discovery of Temporal Patterns. Learning Rules about the Qualitative Behaviour of Time Series. In *Proceedings of PKDD*.

Keogh, E. J., and Pazzani, M. 1998. An Enhanced Representation of Time Series Which Allows Fast and Accurate Classification, Clustering and Relevance Feedback. In *Proceedings of ACM SIGKDD*.

Keogh, E.; Chu, S.; Hart, D.; and Pazzani, M. 2003. Segmenting Time Series: A Survey and Novel Approach. In *Data Mining in Time Series Databases*. World Scientific.

Lesh, N.; Zaki, M.; and Ogihara, M. 1999. Mining Features for Sequence Classification. In *Proceedings of ACM SIGKDD*.

Liu, B.; Hsu, W.; and Ma, Y. 1998. Integrating Classification and Association Rule Mining. In *Proceedings of ACM SIGKDD*.

Morchen, F. 2006. *Time Series Knowledge Mining*. Ph.D. Dissertation, Philipps-University Marburg.

Nadeau, C., and Bengio, Y. 2003. Inference for the generalization error. *Machine Learning, 52:239281*.

Pedro, D., and Pazzani, M. 1997. On the Optimality of the simple Bayesian Classifier under zero-one Loss. In *Machine Learning*.

Rabiner, L. R. 1989. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. In *IEEE*.

Sacchi, L.; Larizza, C.; Combi, C.; and Bellazzi, R. 2007. Data mining with Temporal Abstractions: learning rules from time series. In *Data Mining Knowledge Discovery*.

Shahar, Y. 1997. A Framework for Knowledge-Based Temporal Abstraction. *Artificial Intelligence, 90:79-133*.

Vapnik, V. 1995. The nature of statistical learning theory. In *Springer-Verlag New York*.

Xi, X.; Keogh, E.; Shelton, C.; Wei, L.; and Ratanamahatana, C. 2006. Fast time series classification using numerosity reduction. In *Proceedings of ICML*.