Temporal Sequence Classification in the Presence of Background Domain Knowledge

Sara Carolina Gouveia Rodrigues
sararodrigues@ist.utl.pt

Instituto Superior Técnico, Lisboa, Portugal

November 2015

Abstract

In the last decade there has been a huge interest in classifying temporal data. Temporal data is usually associated with high dimensionality, feature correlation and large amounts of noise, making the classification task a challenge for the research community. Sequence classification methods are one of the techniques applied to solve this problem taking into consideration the temporal order of the sequence. However these methods do not use domain knowledge to learn meaningful information. In this work, we describe the main approaches for sequence classification and propose the adaptation of one of these techniques, Shapelet Decision Tree, through the incorporation of domain knowledge in its training process. With this adaptation we aim for providing the new classifier with the ability of better dealing with the underlying relations between data that are known to exist in a given domain.

Keywords: Data Mining, Sequence Classification, Decision trees, Shapelets, Knowledge Representation

1. Introduction

Over the last decade, the interest of discovering hidden information from sequences has increased. A sequence can be thought of as a list of events with a particular order. They appear in a broad range of real-world applications, from engineering to scientific research, finance and medicine [19] [4].

Sequence Classification methods are used to map a sequence to a class label, taking into account the ordering dependencies of sequences [18]. These methods have still difficulty in learning meaningful information due to the high dimensionality, feature correlation and large amounts of noise that is associated with temporal data.

One of the key issues that can be addressed to find more accurate and more easily understandable models is the incorporation of domain knowledge [3]. We propose to adapt the Shapelet Decision Tree Algorithm by introducing existing domain knowledge in the learning process by working at different levels of abstractions with the use of taxonomies and exploring the known relationships between concepts in the data set. Our results show that the resulting models perform as good or better than the original algorithm while being considerably simpler and more easily understood.

The rest of the paper is organized as follows: in Section 2, we overview the related work and compare the state of the art methods. In Section 3 we explain the original Shapelet Decision Tree Algorithm and we present our proposed extension. Finally, in Section 4 we report the results obtained in terms of accuracy, tree size generated and execution time.

2. Background

A sequence can be thought of as a list of events with a particular order. An event $\alpha$, denoted as $\alpha = (a_1, a_2, ..., a_m)$, is a collection of items that comprise an alphabet $\Sigma$. Thereby, a sequence is denoted as $(\alpha_1 \rightarrow \alpha_2 \rightarrow ... \rightarrow \alpha_q)$, where $\alpha_i$ is an event. For example, DA$\rightarrow$F$\rightarrow$ACGD is a sequence, where DA is an event, and D is an item. The length of a sequence is the number of events that it contains and the width is the maximum size of any of its event.

Depending on the representation of its events a sequence can be considered a Symbolic Sequence or a Time Series Sequence [18]. The main difference between them is that time series data is continuous whereas symbolic sequence data is discrete [1]. This distinction makes the nature of the distance functions used and modeling methodologies quite different.

A Sequence Classifier $C$ is a function that, using a labeled dataset $D$, maps a sequence $s$ to a class label $l$, written as $C: S \rightarrow L$. Sequence Classification methods can be broadly classified into three large categories: Distance, Feature and Model based methods[18].
The main idea of Distance Based Classifiers is to compute a distance score between an unknown sequence and the existing classes and classify this instance to the nearest class.

For symbolic sequences the Alignment Based Distance is the most popular approach [18]. It is performed an arrangement between sequences and then a score is computed based on the number of insertions/deletions and substitutions. NeedlemanWunsch algorithm [12] and Smith-Waterman [16] are examples of global and local alignment algorithms.

For time series sequences the Euclidean Distance [9] and Dynamic Time Warping [11] methods are frequently used.

Model Based Sequence Classification method use a probabilistic perspective to classify a sequence. Their goal is to find the conditional probability of a given sequence s belonging to a class Ci, written as \( P(C_i | s) \). Hidden Markov Models [14] and Recurrent Neural Network (RNN) [6] fall into this category.

Feature Based methods transform each sequence into a feature vector, through feature selection and then traditional supervised classifiers are applied. The main goal of these methods is to select suitable features that are effective for the classification task.

For symbolic sequences we can use k-grams as features, contiguous subsequences of k items that preserve the order of the items in the sequence. For long sequences or for a large alphabet, there is a need to make feature selection to get the features that are good candidates [1]. A relevance score is computed for each feature and then the lowest scoring features are removed. Examples of scoring mechanism are support, confidence and information gain.

Since Time Series data is numeric the feature selection techniques for symbolic sequences cannot be easily applied. Ye et al. [20] proposed a method that can be applied directly on numeric time series, Time Series Shapelets. Instead of comparing the entire time series we only compare a small subsequence. Instead of comparing the entire time series we only compare a small subsequence that can be applied directly on numeric time series, Time Series Shapelets. Instead of comparing the entire time series we only compare a small subsequence that is particularly discriminatory, a Shapelet. To construct a classifier, the shapelet selection process is integrated with the construction of a decision tree.

In this section we describe how to find a Shapelet in a time series dataset and then in Section 3 we will describe how to integrate the shapelet primitive with the construction of the classifier. The most straightforward way for finding the shapelet is using a brute force method, Algorithm 1. Where D is the training dataset and minL and maxL are the minimum and maximum lengths of the shapelet, respectively.

The first step of the algorithm is to generate all possible subsequence candidates. This is done by the subroutine GenerateCandidates that slides a windows with all possible sizes across the dataset and extract all candidates.

Subsequently, after getting the pool of all possible candidates, we need to evaluate the utility of each candidate by computing the information gain of each one. Finally, the algorithm returns the candidate with the highest information gain, that is called the shapelet.

In order to compute the Information Gain of a candidate S we use the subroutine CheckCandidate, that starts by computing the Subsequence Distance between candidate S and each time series T from dataset D. Intuitively, this distance is simply the distance between the candidate and its best matching location, somewhere in the time series. Then, each of this computations are then stored and ordered in objectsHistogram.

Algorithm 1 Brute Force Algorithm for finding a shapelet

1: function FindShapelet(D, maxL, minL)
2:   candidates ← GenerateCandidates(D, maxL, minL)
3:   bestInfoGain ← 0
4:   for all S in candidates do
5:     gain ← CheckCandidate(D, S)
6:     if gain > bestInfoGain then
7:       bestInfoGain ← gain
8:       bestShapelet ← S
9:   end if
10: end for
11: return bestShapelet

Algorithm 2 Checking the utility of a candidate

1: function CheckCandidate(D, shapelet candidate S)
2:   objectsHistogram ← ∅
3:   for all T in D do
4:     dist ← SubsequenceDist(T, S)
5:     objectsHistogram ← T {by key dist}
6:   end for
7: return ComputeInformationGain(objectsHistogram)

Having the objectsHistogram sorted we have everything that is needed to compute the Information Gain (line 7). First we find the optimal Split Point, a distance threshold chosen to split D into two partitions, \( D_{left} \) and \( D_{right} \), such that the we get the purest partitions. Then Information Gain gain can be computed using the Entropy Equation [13].

Then this shapelet primitive can be integrated in a decision tree classifier in order to classify time series sequences.
2.1. Comparative Evaluation
To evaluate the different sequence classification algorithms discussed in the previous sections, we selected six data sets from the UCR time series repository [5]. We run the experiments using 5-folds Cross Validation and all experiments were implemented in Java within the Weka framework. Figure 1: Testing accuracy of six different classifiers

Figure 1 shows the accuracy of the Euclidean Distance with 1-NN, Dynamic Time Warping with 1-NN, Shapelet Decision Tree, Hidden Markov Models and Elman Recurrent Neural Network, all of them using the default settings.

K-NN with Dynamic Time Warping achieved the worst accuracy in these datasets and this seems reasonable since this classifier takes advantage of sequences that have gaps and do not have the same length and all these datasets do not have these properties. RNN achieved the highest performance in the majority of the situations addressed. However, 1-NN euclidean distance and Shapelets Decision Tree classifiers achieved highly competing results.

3. Shapelet Decision Tree based on Domain Knowledge
In the previous Section we discussed some of the Sequence Classification methods in the literature. However they still have some difficulty in learning meaningful information from the data. Most algorithms work on a purely statistical basis ignoring the relationships that can exist between the attributes that could otherwise be explored to produce less complex and more understandable models.

One of the key issues that can be addressed to solve the problems above is the incorporation of domain knowledge deep into existing Sequence Classification algorithms. The use of domain knowledge to improve the mining process tries to accomplish two main goals: to find more accurate and more easily understandable models [3]. Moreover, we argue that the best way to approach these problems is to use more general techniques, guided by domain knowledge, represented through domain ontologies. In the last decade, significant work has been done using domain ontologies for guiding the mining process. For example in pattern mining, where domain knowledge is injected in the mining process in order to obtain a smaller number of results, better aligned with user expectations [2] and in classification where a learning algorithm is guided by a concept hierarchy and can deal with data specified at different levels of abstractions [17]. However, these ideas remain to be explored in the context of Sequence Classification.

Following closely the proposals by Joo Vieira [17], we propose to extent the Shapelet Decision Tree Sequence Classification algorithm to make use of domain taxonomies aiming to build a simpler model, less prone to overfitting and with larger accuracy. We choose to adopt the Shapelet Decision Tree Algorithm since it achieves a highly competitive performance according to the comparative evaluation made. Additionally, the training process of this algorithm is similar with the previous efforts on making use of domain knowledge in classification. As in the previous work the learning algorithm works at different levels of abstraction and explores the known relationships between concepts in the data set.

3.1. Original Shapelet Decision Tree Algorithm
The Learning process of the Shapelet Decision Tree model, also called the decision tree induction, is shown in Algorithm 3.

Algorithm 3: Shapelet Decision Tree Learner

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:</td>
<td>function TrainNode(D, maxL, minL)</td>
</tr>
<tr>
<td>2:</td>
<td>if tuples in D are all of the same class C then</td>
</tr>
<tr>
<td>3:</td>
<td>classValue ← C</td>
</tr>
<tr>
<td>4:</td>
<td>return</td>
</tr>
<tr>
<td>5:</td>
<td>end if</td>
</tr>
<tr>
<td>6:</td>
<td>shapelet ← FindBestShapelet(D, maxL, minL)</td>
</tr>
<tr>
<td>7:</td>
<td>D_left = {}</td>
</tr>
<tr>
<td>8:</td>
<td>D_right = {}</td>
</tr>
<tr>
<td>9:</td>
<td>for all t ∈ D do</td>
</tr>
<tr>
<td>10:</td>
<td>dist ← SubsequenceDistance(Shapelet, t)</td>
</tr>
<tr>
<td>11:</td>
<td>if dist &lt; shapelet.splitPoint then</td>
</tr>
<tr>
<td>12:</td>
<td>D_left ∪ t</td>
</tr>
<tr>
<td>13:</td>
<td>else</td>
</tr>
<tr>
<td>14:</td>
<td>D_right ∪ t</td>
</tr>
<tr>
<td>15:</td>
<td>end if</td>
</tr>
<tr>
<td>16:</td>
<td>end for</td>
</tr>
<tr>
<td>17:</td>
<td>leftNode ← CreateTreeNode()</td>
</tr>
<tr>
<td>18:</td>
<td>rightNode ← CreateTreeNode()</td>
</tr>
<tr>
<td>19:</td>
<td>leftNode.TrainNode(D_left, maxL, minL)</td>
</tr>
<tr>
<td>20:</td>
<td>rightNode.TrainNode(D_right, maxL, minL)</td>
</tr>
<tr>
<td>21:</td>
<td>end function</td>
</tr>
</tbody>
</table>

The basic data structure used by the algorithm is a Tree Node. Since this algorithm works with a binary decision tree, each node holds a left and a right node, and also has a class value associated.
Before calling the training algorithm, a Tree Node is created, with its children nodes and class values empty. Then using this root node we call the TrainNode routine with the parameters: $D$, $maxL$ and $minL$. We refer to $D$ as a partition of training tuples and their associated class labels. The parameter $minM$ and $minL$ are the minimum and maximum size of the shapelet.

The first step of the algorithm is to check if the recursive termination condition is met. The recursive partitioning stops if the tuples in $D$ are all of the same class $C$. In this case the class value of the Tree Node is assigned to $C$ and the node becomes a leaf node.

If the termination condition is not met the algorithm starts splitting the training dataset, $D$. In order to partition the data we need to find the shapelet and its corresponding split point. This is done by the subroutine FindBestShapelet already discussed in Algorithm 1.

In order to find the best shapelet, we need to generate all possible candidates using the sliding windows approach. Then we find which candidate achieves the highest information gain and that candidate is the shapelet. In order to obtain the information gain of each candidate we compute the subsequence distance between the candidate and all training instances and then we order the resulting distances in a line. Subsequently, we find the best split point that separates the classes in the line and compute the information gain. This way the shapelet and its split point will act as the splitting criterion and will tell us which is the best way to separate the tuples in $D$ into individual classes.

Subsequently, since we are working with a binary tree we know before hand that we will split the dataset $D$ into two partitions, so $D_{left}$ and $D_{right}$ partitions are created. We then compute the subsequence distance between the shapelet and each training instance $T$. If the distance obtained is smaller than the split point the training instance goes to the left partition, otherwise it goes to the right partition. Afterwards, the children nodes are created and recursively called to be trained, until they reach the final termination condition, which is, when all the instances in a partition have the same label.

Finally, the classifier used to predict the class label of each time series uses the Model created by the Learner. Starting from the root of the Model, we compute the Subsequence Distance from the testing time series $T$ to the Shapelet node. If the distance is smaller than the split point we recursively use the left subtree and otherwise use the right subtree. This procedure continues until we reach the leaf node and return the predicted class label.

### 3.2. Shapelet Decision Tree Based on Domain Knowledge

At this moment we have explained how to construct a decision tree learner and classifier using the shapelet time series primitive. In this section we present our proposed extension to the Shapelet Decision Tree Algorithm that explores a hierarchy of concepts, representing the elements of the time series at different levels of abstraction.

To incorporate these ideas both the Learner and the Classifier have to change. The Learner has to build a model that at each time of the induction phase choose the best level of abstraction. Then the Classifier has to be able to use this more abstract model and still classify instances composed by concrete features.

The learning algorithm receives as input the training dataset $D$ and the domain knowledge in a form of a taxonomy $T$. $T$ is a taxonomy that relates the concepts of the domain in an hierarchical style through a axioms, as in OWL2 [7].

As a preprocessing step the algorithm incorporates all the training instances $D$ and the domain knowledge $T$ in the same data structure. We called this data structure the enhanced partition $D'$ and it corresponds to the set of $D_i$ datasets where each $D_i$ corresponds to the training dataset $D$ represented at different levels of abstraction, $i$. To obtain the $D'$ enhanced partition, we simply apply the taxonomy $T$ to each element of the time series instances in the training dataset $D$. This way we are abstracting the elements of the time series in each particular time instance.

Afterwards, our Learner can start building the decision tree using the enhanced partition $D'$ as the training partition that we are trying to split. Since the Learner suffers some modifications from the standard Shapelet Decision Tree Learner we describe it in Algorithm 4.

The termination condition is very similar with the standard algorithm. We need to check whether $D'$ is a pure partition or not, that is if all of its tuples belong to the same class $C$. Since every dataset in $D'$ contains the same training instances only represented in a different way, we just need to check if the first dataset $D_0$ is pure or not. In fact, this first dataset is the original training dataset, the concrete one without any abstraction applied.

Subsequently the algorithm tries to split the enhanced partition $D'$ by finding the best shapelet and its corresponding split point. This subroutine is also modified since we will search for the shapelet in all datasets in $D'$, that is in all different levels of abstraction. Besides getting the shapelet and its split point, the algorithm also returns the level of abstraction that the shapelet found belongs to, `abstractionLevel`. 

4
Based on Domain Knowledge

tended in order to deal with our modified Learner, Algorithm 4

Finally, each training instance is added to \( D'_l \) or \( D'_r \) children partition, according to the comparison between the subsequence distance obtained and the shapelet split point. As usual, if the distance obtained is smaller than the shapelet split point the instance goes to the left partition otherwise it goes to the right partition. One important aspect in this step is that the instance is added to the children partition in all levels of abstraction (lines 11 to 20).

The remain of the algorithm maintains the same as the original one. The children nodes are created and recursively computed until they reach the termination condition, that is, when all partitions have pure tuples.

**Algorithm 4 Shapelet Decision Tree Learner Based on Domain Knowledge**

```plaintext
1: function TrainNodeUsingTaxonomy (D', maxL, minL)
2:   if tuples in \( D'_0 \) are all of the same class C then
3:     classValue ← C
4:   return
5: end if
6:   shapelet,abstractionLevel ← FindBestShapeletUsingTaxonomy(D',maxL,minL)
7:   \( D'_l ← initArray(D'.size()) \)
8:   \( D'_r ← initArray(D'.size()) \)
9:   for all \( t \in D'[abstractionLevel] \) do
10:      dist ← SubsequenceDistance(shapelet,t)
11:      if dist < shapelet.splitPoint then
12:         for all i in \( D'_l \) do
13:            \( D'_{l,i} \cup t \)
14:        end for
15:      else
16:         for all i in \( D'_r \) do
17:            \( D'_{r,i} \cup t \)
18:        end for
19:      end if
20: end for
21: leftNode ← CreateTreeNode()
22: rightNode ← CreateTreeNode()
23: leftNode.TrainNode(\( D'_l, maxL, minL \))
24: rightNode.TrainNode(\( D'_r, maxL, minL \))
25: end function
```

The Shapelet Decision Tree Classifier is also extended in order to deal with our modified Learner, since it creates a model with shapelet nodes represented at different levels of abstraction. The main difference from the original classifier is that before computing the subsequence distance between the testing instance and the shapelet node, we need to check if both sequences are represented in the same level of abstraction. If they are not in the same level of abstraction we cannot compute the subsequence distance properly. In that case, we need to climb the taxonomy \( T \) to the shapelet level of abstraction and apply it to the testing instance.

Then the remain of the algorithm maintains the same, we recursively compute the subsequence distance between each testing instance and the shapelet node, and then choose the appropriate branch according to the comparison between the distance obtained and the shapelet split point. The recursion ends when we found a leaf, in that case the instance is classified with that class value.

### 3.3. Illustrative Example

We will use the Italy power demand dataset [5] to illustrate our proposed extension of the Shapelet Decision Tree Algorithm.

The first step of the algorithm is to gather the two types of knowledge, the observations that are in partition \( D \) and the domain knowledge that will be used to abstract these time series observations. This way, the first step of the algorithm is to create the enhanced partition \( D' \), the training dataset represented at different levels of abstraction.

In order to represent the training dataset into different levels of granularity we need to abstract the time series. This can be done with the use of taxonomies provided by domain experts or based on automatic data driven discretization methods. Since we do not have the domain knowledge for this dataset, we will illustrate our algorithm applying the Equal Width Discretization varying the number of bins for different levels of granularity. In Figure 2 is shown part of the enhanced partition \( D' \), representing two training instances in five levels of abstraction after applying the discretization method varying the number of bins from 10 to 15.

![Figure 2](image-url)  
Figure 2: Enhanced partition \( D' \), the training dataset represented at different levels of abstraction.

Then as in the original Shapelet Decision Tree algorithm, we need to find the shapelet in the training dataset. However, in our extension instead of just searching the shapelet in the concrete training
dataset, we will also search the shapelet in all different abstractions levels. This way, the shapelet found can belong to any $D'_i$ present in $D'$. Therefore, the method for finding the shapelet is very similar to Algorithm 1 with the exception of having to search in different partitions and retrieve the shapelet found and its corresponding level of abstraction.

Running this dataset we obtain the shapelets shown in Figure 3, their corresponding split points as well as the level of abstraction where they were found. The shapelets shown in red are different from the ones found by the original Algorithm, since they were found in different levels of abstraction allowing to choose a better splitting criteria.

![Figure 3: Shapelet Dictionary, respective split points and level of abstraction where shapelets were found using our proposed algorithm.](image)

Figure 3: Shapelet Dictionary, respective split points and level of abstraction where shapelets were found using our proposed algorithm.

After finding the shapelets and their corresponding split points the algorithm can use them as a splitting criteria for the training dataset, comparing the results of the subsequence distance and shapelet split point as usual. However with our extension we need to take into consideration that we are comparing the shapelet and the instances at the same level of abstraction. This way, before proceeding to the computation of the subsequence distance we choose the appropriate $D'_i$ for the training instance according to the level of abstraction $i$ that the shapelet belongs.

The final tree that was created is shown in Figure 4, it seems very similar with the tree created by the original Algorithm. However the nodes where shapelets with different abstractions were found created some differences in the decision tree model. Besides getting a smaller decision tree, the new shapelets found will also have different split points that can better discriminate the instances.

![Figure 4: Decision Tree obtained, each node is represented by a shapelet, it split point, the level of abstraction and the number of training instances that the node is holding.](image)

Figure 4: Decision Tree obtained, each node is represented by a shapelet, it split point, the level of abstraction and the number of training instances that the node is holding.

These minor modifications of the model can make it more simple and more accurate. The procedure to classify is very similar we continuously compute the subsequence distance between the unseen instance and the shapelet nodes and proceed to the appropriate branch. However with our extension we need to take into consideration again the level of abstraction between the shapelet and the testing instance. This way before computing the subsequence distance between the test instance and the shapelet we verify if they are represented in the same level of abstraction. If they are not represented in the same level of abstraction we apply the same taxonomy to the testing instance and then compute the subsequence distance.

In Figure 5 is shown the path taken until it reaches the leaf. In this case the classification obtained is correct. In fact the precision of the model increased from to 59,12% 64,05%.

![Figure 5: Classification of an unseen instance using the Decision Tree Model created by our proposed Algorithm.](image)

Figure 5: Classification of an unseen instance using the Decision Tree Model created by our proposed Algorithm.

3.4. Optimizations

The Shapelet Decision Tree Algorithm is a brute force algorithm and although it finds the optimal shapelet it suffers from high time complexity. There is a very large number of possible shapelet candidates that are generated. Suppose the dataset $D$ contains $k$ time series objects. We specify $maxL$ and $minL$ to decide the size of the windows used to generate the candidates. Obviously, $maxL$ has to be smaller than $m_i$, the length of the time series $T_i$ from the dataset $D_i$, where $1 \leq i \leq k$.

Considering a certain fixed length $l$, the number of shapelet candidates generated from the dataset is:
We decided to store the subsequence distances obtained for each candidate in an HashTree, a recursive tree structure of objects. For each candidate we create an HashTree that stores the subsequences element by element. That is, each element in the tree is also a key to the next node down in the tree. And each key has their corresponding subsequence distances. Before computing the next subsequence distance we check if this subsequence was already computed by transversing the tree. If it is stored we use that distance otherwise we need to compute it.

4. Results

We performed some experiments to compare the performance of the standard Shapelet Decision Tree Algorithm with our proposed extension that incorporate domain knowledge deep in the mining process.

We have designed and conducted all experiments such that they are easily reproducible. For the standard algorithm we used a Java implementation within the WEKA framework available at [8] to ensure no mistakes were made. Then we modify and extended this implementation to build our proposed algorithm and this way make use of domain knowledge in the learning and classification of sequences. Our implementation code can be obtained and executed at [15].

We emphasize the importance of statistical validity in the design of the experiments. Our case studies include several real-world datasets from various domains such as shape, motion capture, spectrography as well as synthetic datasets. These datasets were selected from the UCR time series repository [5]. The URC repository contains datasets that have have been largely used by the research community for comparing results with the different approaches in sequence classification [10].

We selected these particular UCR datasets since they have relatively few cases; even with optimization, the shapelet algorithm is time consuming. Besides this we also tried to gather a set of problems with different properties from each other. Our selection criteria was based on the number of classes and time series length. Therefore we chose a set of binary and multi class problems. Then we selected datasets with large, medium and small time series length.

Our Learner Algorithm also receives as input domain knowledge in order to abstract the time series observations. This domain knowledge can be provided by domain experts through taxonomies or based on automatic data driven discretization method. Since we do not have the domain knowledge for these datasets, we applied Equal Width Discretization method discretizing the training dataset into different abstractions, according to

\[ \sum_{T \in D} (m_i - l + 1) \]  

(1)

So the total number of candidates of all possible lengths is:

\[ \sum_{l=\min L}^{\max L} \sum_{T \in D} (m_i - l + 1) \]  

(2)

Then we check the utility of each candidate in order to find the shapelet. Checking the utility of one candidate requires its comparison with \( O(mk) \) subsequences. Hence, the overall time complexity of the algorithm is \( O(m^3k^2) \).

Although our proposed algorithm shows better accuracy in some datasets it suffers even more from time complexity. This happens since the candidate set generated is bigger than the one obtained by the original algorithm. Depending on the number of abstractions used, \( N \), the candidates pool size is:

\[ \sum_{l=\min L}^{\max L} \sum_{T \in D} (m_i - l + 1) \]  

(3)

This time complexity is huge and for this reason, we optimize our proposed algorithm applying some pruning techniques and a short memory to store distances and avoiding repeated computations.

The first optimization is called Early Abandon and it avoid unnecessary computations when computing the subsequence distance. In the brute force method, the distance from the time series \( T \) to the subsequence \( S \) is obtained by computing the Euclidean Distance of every subsequence of length \( |S| \) in \( T \) and \( S \) and choosing the minimum one. However, all we need to know is the minimum distance rather than all of the distances. Therefore, instead of computing the exact distance between every subsequence and the candidate, we can stop distance computations once the partial distance exceeds the minimum distance known so far.

The second proposed optimization is to store the subsequences distances computed in an HashTree. Our algorithm represents the training instances in different levels of abstraction depending of the taxonomy used or the number of bins used for the discretization. We then noticed that the training instances that are represented in more abstracted levels have their observations often repeated.

We can take this observation into our advantage when computing the information gain for each candidate. In order to compute the information gain for a candidate we need to compute the subsequence distance from it and all training instances. Since the training instances have numerous repeated observations we can store them and avoid computations.
the number of bins specified by the user. The different numbers of bins chosen a priori will create different levels of abstraction, since a discretization made by a small number of intervals is more abstract than a dataset discretized by a bigger number of intervals. There is no “best” way of choosing the number of bins, and different bin sizes can reveal different features of the data. For this reason we made several experiments in order to find out what were the appropriate number of bins to use. We find out that varying the number of bins from 10 to 15 achieved better performance in the majority of the datasets.

We run our experiments using 5-folds Cross Validation this way the performance comparison reported is averaged over five randomly selected training/testing split. The performance metrics that we gather when running the experiences are the accuracy, the size of the tree model created and the running time. All of this metrics are compared with the standard Shapelet Decision Tree Algorithm.

In this Section we present our experimental results in three stages. In Section 4.1 we compare the accuracy achieved in 12 datasets from the UCR repository [5] between the original Shapelet Decision Tree Algorithm and the Shapelet Decision Tree Algorithm based on Domain Knowledge. Then in Section 4.2, we evaluate the size of the tree models created in the learning phase in both algorithms. Finally, In Section 4.3, we evaluate the effectiveness of the implemented optimizations.

We wish to demonstrate three major points with our experiments. First, that the incorporation of different levels of abstraction with the use of taxonomies in the Shapelet Decision Tree Algorithm can build a simpler model that can better generalize being less prone of overfitting. Second, there exist real world problems for which our proposed approach can significantly be more accurate. For example the dataset that achieved the better result was the BeetleFly improving its classification from 75.00% to 82.50%.

The incorporation of domain knowledge can help build models that perform as good or better than the original algorithm while being considerably simpler and more easily understood.

4.1. Accuracy

The first experiment consists of running both the original shapelet decision tree algorithm and our proposed extension based on domain knowledge in 10 different datasets from the UCR repository using 5-fold cross validation.

The results are shown in in Figure 6 in a high level of abstraction for better visualization and interpretation.

Visually it can be difficult to say which algorithm is better, and counting wins, ties, and losses produces similarly ambivalent results. However, the results are consistent with our claim that our method is no worse than the standard Shapelet Decision Tree Algorithm and there exist real world problems for which our proposed approach can significantly be more accurate. For example the dataset that achieved the better result was the BeetleFly improving its classification from 75.00% to 82.50%.

The incorporation of domain knowledge can help build models that perform as good or better than the original algorithm while being considerably simpler and more easily understood.

4.2. Decision Tree Size

In this experience we evaluate the size of the tree model created during the learning phase of both algorithms: the original shapelet Decision Tree Algorithm and the Shapelet Decision Tree Algorithm based on Domain Knowledge.

The results are shown in Figure 7. The tree size was measured in terms of number of shapelet nodes that the model contains. We can see that our algorithm can produce smaller tree models in some cases. In these cases we also achieved better accuracies and this is expected since the model created is more simpler, more generalizable and less prone of overfitting.

We noticed that the datasets that improve their tree size using our extension were the ones with many training instances. This is expected since the ones with fewer instances create already small trees with only one shapelet node. In these cases our algorithm could not be better.

It is interesting to consider why using different abstraction levels can generalize the new data. We conjecture that it is the ability of abstractions to extract just the meaningful part of the signal. In contrast, the traditional shapelet decision tree algorithm take account for the exact values of the entire time series, including sections that may be unique to individual instances but not predictive of the concept.

Figure 6: Accuracy Comparison between Shapelet Decision Tree Algorithm and Shapelet Decision Tree Algorithm based on Domain Knowledge.
4.3. Performance

We test the scalability on the TwoLeadECG dataset from the UCR repository [5]. We choose this dataset since it is the largest class-labeled time series dataset that we used in our experiments.

The two speedup methods described in Section 3.4 are not independent of each other. Therefore, we measure the individual speedups for each of the techniques while deactivating the other and then measure the combined speedup obtained when using both optimizations.

This way we compared the performance between the original brute force approach, the early abandon optimization and the storage of subsequence distances in the HashTree.

Using this three approaches, we started by finding the shapelet in a subset of just ten time series objects. We then iteratively increase the size of the data subset. Figure 8 shows the results obtained with this experience.

The results show that using the brute force is still quicker than our proposed approaches. Still, early abandoning helps to reduce the searching time when there are few time series objects in the search space. The HashTree takes more time because it takes more time adding branches and transversing the tree than the simple brute force takes to just compute sums and subtractions between values.

5. Conclusions

We extend the original Shapelet Decision Tree Algorithm and present a Shapelet Algorithm that incorporates Domain Knowledge representing time series at different levels of abstraction. In the induction step the algorithm chooses the best abstraction level to use as shapelet and in the classification we convert the time series instance to the same level as the shapelet node.

Our extensive experiments shows that our proposed algorithm is no worse than the original Shapelet Decision Tree Algorithm and that exist real world problems for which our proposed approach can significantly be more accurate. In addition, our algorithm can produce in some cases, smaller tree models being more simpler, more generalizable and less prone of overfitting.

Although the results presented have demonstrated the effectiveness of our taxonomy driven approach, it can, of course, be further developed in a number of ways. The most obvious weakness is the relatively slow training time that has increased with the incorporation of different levels of abstractions. One possible way to mitigate this is by sampling the training set and only using a small subset to search for the shapelet. In addition, although we used decision trees because of their ubiquity, it is possible that shapelets could be used in conjunction with other classification techniques.

Acknowledgements

At the top of the list, I want to thank my advisor, Prof. Claudia Antunes, for always finding a slot for me on her busy schedule and for her sage advice, insightful criticisms, and support.

I am also thankful to my family and boyfriend for being kind and supporting.

References


