User Manual: TSAT

Summer 2018

Contents

1 Manual Overview

TSAT or the Time Series Analysis Tool is a software application that has enhanced the capabilities of GrammarViz 2.0 and 3.0 [1, 2, 3]. **TSAT** has three main features:

Supervised Classification Using labeled time series train an algorithm to classify unknown time series

Motif Discovery Finding repeated patterns within a time series

Anomaly Detection Finding rarely repeated patterns within a time series

This manual is to provide users, power users, and developers a detailed guide to using the Time Series Analysis Tool (**TSAT**). Background information and definitions are provided in section 2 along with an overview of the interface in section 2.1.

Users This is a manual for users who want to quickly learn the common features and how to use them. The relevant sections include section 3, 4, and 5 where the guide will provide step-by-step instructions on using Motif Discovery, Anomaly Detection, and Time Series Classification in **TSAT**. Some background material on machine learning is provided in section 2.3.3.

Power Users This is a manual for power users that have a background in machine learning. Therefore, in addition to the guide there is additional information into the time series algorithms in sections 2.3.1, 2.3.2, and 2.3.4.

Developers In addition to providing a detailed guide to TSAT's graphical user interface the manual also includes helpful information on the python interface for motif discovery, anomaly detection, and Time Series Classification in sections 2.2, 3.4, 4.2, and 5.9 and relevant notes in section 7.

2 Introduction to TSAT

The Time Series Analysis Tool or **TSAT** for short is a software application for analyzing time series. This tool provides both a Graphical User Interface or GUI described in section 2.1 and a python interface detailed in section 2.2. This section also provides an overview of the algorithms that are implemented in section 2.3.

2.1 Interface Layout

The main method for interacting with **TSAT** in this user guide will be through the graphical user interface, or the GUI. When TSAT is started the GUI should look like that in Figure 1.

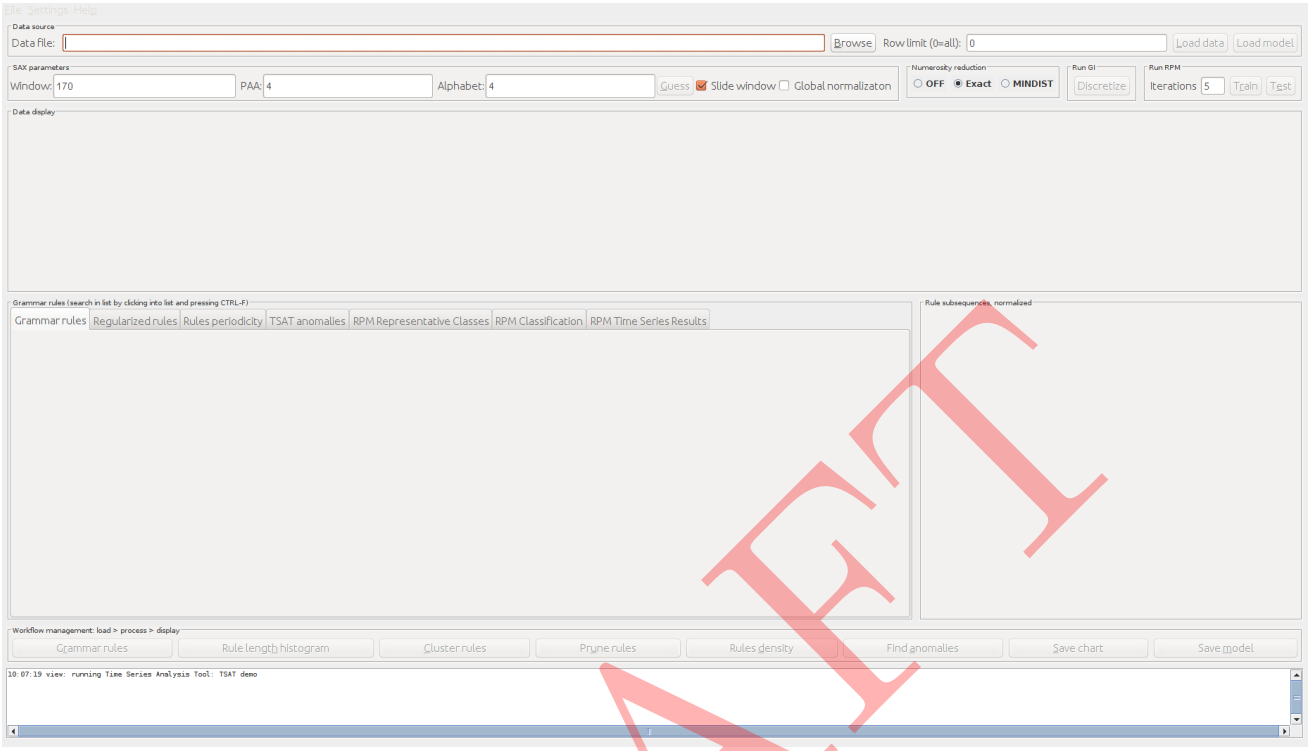

Figure 1: Initial state of the **TSAT** Graphical User Interface (GUI).

There are eleven regions in the main GUI used to set parameters, load files, and analyze results. The nine main regions in the GUI have a rectangular outline around them with a label. They include: "Data Source", "SAX Parameters", "Numerosity reduction", "Run GI", "Run RPM", "Data display", "Grammar rules", "Rule Subsequences, normalized", and "Workflow management." The other two regions are the top menu bar (with "File", "Settings", and "Help") and the text area at the bottom of the GUI. Presently, the items under "Help" do not have any meaningful functionality. The text area at the bottom of the GUI is used to log useful information about the state of GUI. That text area may also be referred to as the logging area.

2.2 Python Interface

In addition to a Graphical User Interface, **TSAT** also has a python interface. This interface consists of wrappers of the major functionality of **TSAT**'s GUI.

The python interface is found in the python directory as python/tsail.py. In order for the python interface to work, the **TSAT** jar must be located in:
`/target/tsat-1.0-SNAPSHOT-jar-with-dependencies.jar`
If it is not there, either its alternative location must be set within tsail.py by editing within tsail.py the following variable to the correct location:

```
TSAT_JAR_LOCATION =
    "../target/tsat-{\TSATVERSION}-SNAPSHOT-jar-with-dependencies.jar"
```

or **TSAT** can be recompiled by running:
`mvn package -Psingle`
which will generate the jar file in the correct location:

/target/tsat-1.0-SNAPSHOT-jar-with-dependencies.jar
In the python directory in TSAT you can run python and then type:

```
import tsail
```

Then you can call the functions like

```
tsail.buildMotifs(pathToTimeseries, outputFile, window_size=30, word_size=6,
    alphabet_size=4, strategy="EXACT", threshold=0.01, numworkers=2)
tsail.RRA(pathToTimeseries, outputFile, window_size=30, word_size=6,
    alphabet_size=4, threshold=0.01, discords_num=5)
tsail.RPMTrainTest(pathToTraining, pathToTest, outputFile, num_iters)
tsail.RPMTrain(pathToTraining, outputFile, num_iters)
tsail.RPMTest(pathToTest, modelFile, num_iters)
```

For motifs, anomaly detection and representative pattern matching respectively. In each of the chapters on Motif Discovery, Anomaly detection, and Time Series Classification there are instructions on how to use these functions. Examples of their usage are also provided within tsail.py.

2.3 Tools Introduction

TSAT provides implementations of a number of algorithms used to analyze time series including Representative Pattern Mining (RPM) to perform time series classification, Motif Discovery to find repeating patterns, and Discord Discovery to detect anomalies.

2.3.1 Motif Discovery

A motif is a reoccurring pattern within a time series (an example motif is shown in Figure 4) and both anomaly detection and representative pattern mining build from this concept. In order to identify motifs within a time series **TSAT** first converts the time series into a string (a sequence of words) and then performs context free grammar induction (GI). Specifically **TSAT** executes two main algorithms SAX (Symbolic Aggregate Approximation) with numerosity reduction and a user chosen GI algorithm either Sequitur or Re-Pair [2, 4, 5]. The motifs are then defined as the subsequences in the time series defined by the grammar rules. **TSAT** allows the user to explore the motifs by sorting by how frequently the rule is used in the root rule (labeled R0 in **TSAT**).

SAX SAX converts the time series into a string, or a sequence of characters. It does this by using a fixed size sliding window over the time series and performing Piecewise Aggregate Approximation (PAA) on each window and converting those values into words. This is called subsequence discretization [6].

First SAX splits the time series up into smaller time series by only looking at a fixed size window, or subsequence extraction as seen in Figure 2. So, if the time series is of length 1000 and the window length is 100 then each time series that SAX looks at is of length 100. SAX uses a sliding window with a step size of one. This means that the first time series given the same example spans from timestep 1 to 100 and the second time series is 2 to 101 and so there will

be 999 different time series of length 100. The formula is $n - w + 1$ where n is the length of the time series and w is the length of the sliding window (the **window length**).

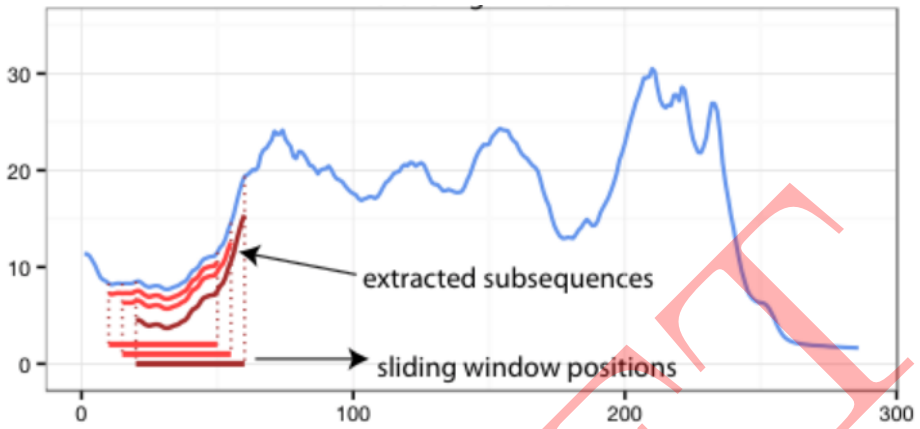

Figure 2: Subsequence extraction from a sliding window. Adopted from: http://grammarviz2.github.io/grammarviz2_site/morea/assets/sax-error.png

Then for each sliding window time series, PAA produces a word by first performing z-normalization on the values. This means converting the time series values to values with a mean and standard deviate of approximately 0 and 1 respectively. However, so as to not amplify the "under-threshold-noise" there is a **z-normalization threshold** value so that if the input time series' standard deviation is less than this value the z-normalization will not be applied. Then the algorithm splits the window up into m equal sized segments called the **PAA size** and for each of the segments it computes the mean value.

SAX maps each mean value to a letter in the alphabet and produces a word (a sequence of letters/characters). The number of characters, a, available in the alphabet is chosen by the user (the **alphabet size**). Since the values of z-normalized time series follow the Normal distribution [7], the breakpoints for each character can be determined by making a equal-sized areas under the Normal curve using lookup tables (illustrated along the y-axis in Figure 3). Then for each of the windows we have created a word. Figure 3 shows the process of converting a time series (without any sliding window) into a SAX word. It should be pointed out that the "observation that normalized subsequences have highly Gaussian distribution, is not critical to correctness of any of the algorithms that use SAX, including the ones in this work. A pathological dataset that violates this assumption will only affect the efficiency of the algorithms" [8].

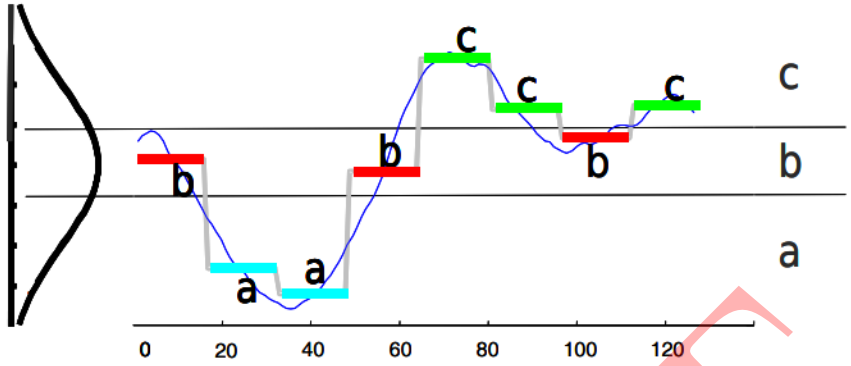

Figure 3: "A time series is discretized by first obtaining a PAA approximation and then using predetermined breakpoints to map the PAA coefficients into SAX symbols. In the example above, with n = 128, w = 8 and a = 3, the time series is mapped to the word baabccbc." Both the figure and the included caption are from [9].

Numerosity Reduction The list of words produced using this procedure is also compressed using numerosity reduction. Numerosity reduction reduces the size of this list of words by skipping duplicate words. Additionally, "numerosity reduction makes motif discovery more robust, as we are matching patterns based on their shapes, even if they do not have the exact same lengths" [10]. For example, a time series S_1

$$S_1 = aac_1\, aac_2\, abc_3\, abb_4\, acd_5\, aac_6\, aac_7\, aac_8\, abc_9\, \ldots$$

is converted to the much smaller string using numerosity reduction:

$$S2 = aac_1\ abc_3\ abb_4\ acd_5\ aac_6\ abc_9$$

where the subscripts are the window numbers.

Grammar Induction GI Then grammar induction is used to produce grammar rules, the motifs, from the SAX string. Both Sequitur and Re-Pair are context free grammar induction algorithms that are included in **TSAT**. **TSAT** uses Sequitur as its default. However, there are a number of differences according to [11]:

- Sequitur implementation is slower than Re-Pair
- Sequitur tends to produce more rules, but Sequitur rules are less frequent than Re-Pair rules
- Sequitur rule-corresponding subsequences vary in length more
- Sequitur rules usually cover more points than Re-Pair
- Sequitur rule coverage depth is lower than that of Re-Pair

A simple example (not a time series) of a context free grammar is to take the following string "a rose is a rose is a rose" this can be converted to the following grammar rules:

$$S \rightarrow AAB$$
$$A \rightarrow B \text{ is}$$
$$B \rightarrow \text{a rose}$$

Where S is the root rule, meaning that no rule uses this rule and A and B are grammar rules that are used in S. Also, note that the grammar forms a hierarchy and therefore will not contain cycles. As the above example illustrates the lengths of the motifs can be of varying lengths as the rule may contain other rules (note that in an actual time series each word would be the same length).

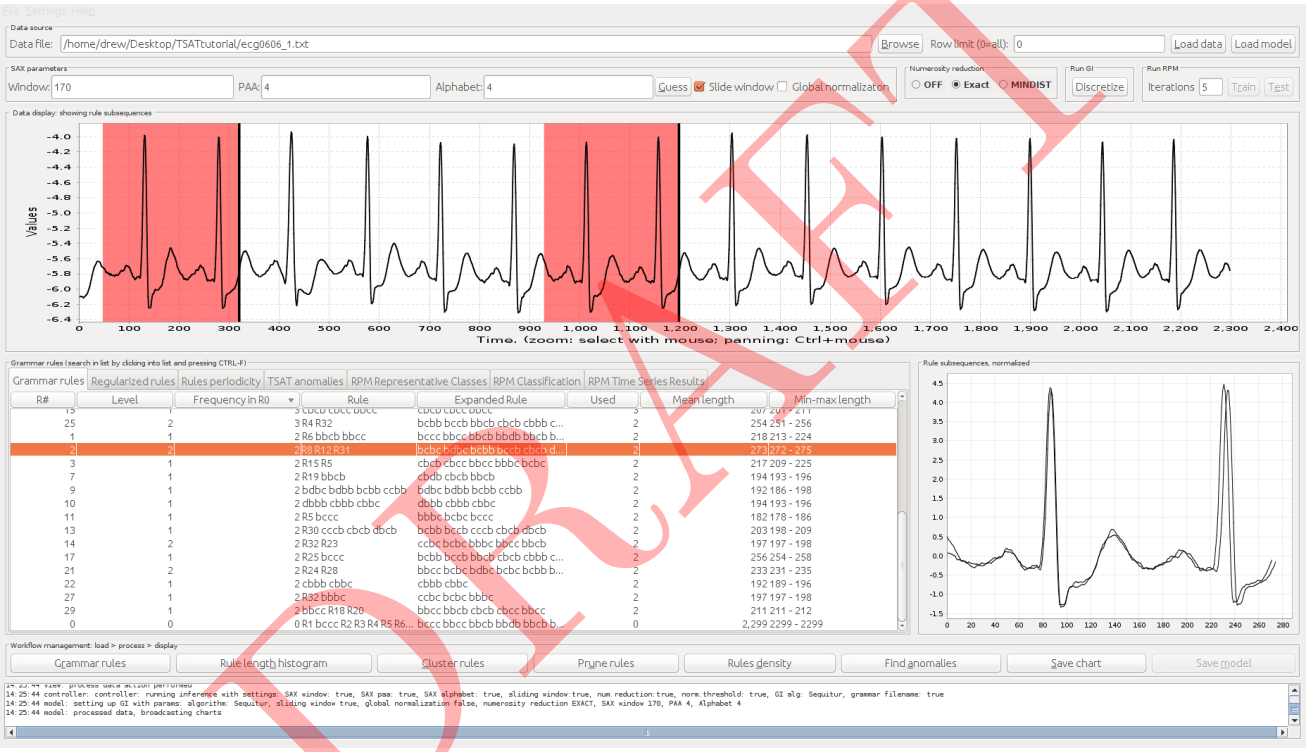

Figure 4: Example motif found using **TSAT** with the subsequences highlighted in the Data Display and shown in the Rules Subsequences areas.

Setting the Parameters One example motif found by **TSAT** in ECG data is shown in Figure 4. There are only three parameters, alphabet size, PAA length, and window size. Past studies have empirically shown that an alphabet size of 3 or 4 will work in most settings and that the PAA length depends on the data. The PAA length (also known as word size) tends to be a smaller value for smooth and slow changing time series and a larger value for more complex time series [8]. "Note however, that grammar induction step effectively mitigates improper sliding window selection" [12].

2.3.2 Anomaly Detection

Anomalies in **TSAT** are surprising patterns or more specifically grammar rules that occur rarely in the time series. **TSAT** implements two anomaly detection algorithms taking an exact approach and an approximate approach [13]. Specifically the **Rare Rule Anomaly** (RRA) algorithm and

the **rule density curve** algorithm. Each method uses the grammar rules generated by Sequitur or Re-Pair to extract the corresponding subsequences in the time series. However, each method uses these subsequences in a different way.

RRA defines anomalous subsequences as *discords* or the subsequences whose euclidean distance (normalized by the subsequence length) to their nearest non-self match is the largest. A subsequence is a non-self match with another subsequence if their subsequences do not overlap.

The approximate anomaly detection method is implemented using the rule density curve. The value at each point in the rule density curve is the number of grammar rules that span or "cover" the corresponding point in the time series. Therefore, "rule density curve intervals that contain minimal values correspond to time series anomalies" [13].

Both the exact and approximate methods can find variable length anomalies. However, if the time series being analyzed is highly irregular (very few motifs) "or the discretization parameters are far from optimal and regularities are not conveyed into the discretized space, the rule density based anomaly discovery technique may fail to output true anomalies" [13]. Therefore, using the exact approach is preferable.

Setting the Parameters The best advice is from [13]:

> Specifically, we found that the rule density curve facilitates the discovery of patterns that are much shorter than the window size, whereas the RRA algorithm naturally enables the discovery of longer patterns. Second, we observed that when the selection of discretization parameters is driven by the context, such as using the length of a heartbeat in ECG data, a weekly duration in power consumption data, or an observed phenomenon cycle length in telemetry, sensible results are usually produced.

2.3.3 Machine Learning Terms

Before getting into Representative Pattern Mining (RPM), which uses machine learning, this section provides some useful definitions to a few common machine learning terms that may be encountered in this guide:

Attribute An attribute, variable, or feature is a value that is used to describe the data point. For example, a time series is an attribute.

Class A class or label is the name used to describe a set of attributes. In classification the goal is to classify examples as belonging to a particular class or to assign a label.

Example An example is the set of attributes used to describe a single data point.

Univariate Univariate, single attribute, or single feature means that each example is represented by a single attribute. This means a single time series in time series classification is defined to be an example.

Multivariate Multivariate, multi-attribute, or a feature vector means that each example is represented by multiple attributes.

Supervised Classifier An algorithm that creates a model representation from a set of labeled examples in order to classify unlabeled examples. Some example classification algorithms

n=106	Actual positive	Actual negative	
Predicted positive	TP=1	FP=5	6
Predicted negative	FN=5	TN=95	100
	6	100	

Table 1: Example Confusion Matrix. TN = True Negative, FP = False Positive, FN = False Negative, TP = True Positive

are, Support Vector Machine, Random Forest, Logistic Regression, AdaBoost, or Naive Bayes.

Training Data The set of labeled examples used by the supervised classifier to create a model representation in order to make predictions.

Test Data The set of labeled examples that are independent from the training data and that are used to assess the performance of the classifier.

Validation Data The data that is held out and usually used to tune parameters.

In addition there are a number of means for describing the performance of the classifiers. The most useful statistics **TSAT** provides include:

Accuracy The accuracy of a classifier is the number correctly classified examples divided by the total number of examples.

Confusion Matrix A confusion matrix is useful in describing fully the performance of the classifier and one is illustrated in table 1. Here there are two classes, negative and positive, and the confusion matrix describes the result of the classifier. For example, if the class is actually positive and is predicted negative then the value of FN would increase by one. Also, the values in the confusion matrix are used to calculate the F1 Measure and the Matthews correlation coefficient.

F1 Measure Or F1 score is defined as $2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$. Where

$$\text{precision} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}}$$

and

$$\text{recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

Where in a binary class true positives mean that a classifier correctly labeled an example, false positive means that a classifier labeled an example as this class when it was the other class, and false negative means it was this class when labeled the other class. Therefore, a value of 1 is perfect precision and recall and 0 is the worst.

Matthews correlation coefficient MCC This value takes into account true and false positives and negatives and is generally regarded as a balanced measure which can be used even if the classes are of very different sizes [14]. It is defined as:

$$\frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

The values are between -1 and +1. +1 represents a perfect prediction, 0 no better than random prediction, and 1 indicates total disagreement between prediction and observation.

Area Under the Curve AUC The AUC is the area under the Receiver Operator Characteristic (ROC) curve. The ROC curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings. When using normalized units, the area under the curve is equal to the probability that a classifier will rank a randomly chosen positive instance higher than a randomly chosen negative one (assuming 'positive' ranks higher than 'negative'). The AUC varies between 0 and 1 where .5 indicates an accuracy no better than chance and 1 meaning perfect accuracy [15].

Kappa statistic This value measures the agreement between two raters who each classify N items into C mutually exclusive categories. It takes on values between 1 and negative infinity. A value of 1 indicates perfect accuracy, 0 is no better than random, and negative value is worse than random [16].

Error So, there are a number of measures for error in the prediction. Such values include: Mean absolute error, Root mean squared error, Relative absolute error, and Root relative squared error. Since RPM is performing classification on labeled categories, then the distance error measurements that are reported in TSAT should not be used. This is because the distances are not relevant as the distances between the classes are not meaningful.

Weighted F1 Measure This value is the F1 scores for each of the classes weighted by the number of examples within each of the classes divided by the total number of examples.

Weighted MCC This value is the MCC values for each of the classes weighted by the proportion of how many elements are in each class.

Weighted AUC This value is the AUC values for each of the classes weighted by the proportion of how many elements are in each class.

Accuracy is a flawed method for describing the performance of a classifier. For example, consider the confusion matrix in table 1 where the accuracy is 90%. However, the F1 score is 0.1666 which indicates that the precision and recall is not performing well at all. This is due to the imbalance of the datasets and rebalancing using a technique such as up-sampling should be used to balance the number of examples in each class.

2.3.4 Representative Pattern Mining - RPM

Univariate multiclass supervised time series classification is implemented in **TSAT** with **Representative Pattern Mining** or **RPM** [17]. RPM works by identifying an optimal sliding window size, PAA size, and alphabet length for each class and it identifies the motifs that match

the class it belongs to more so than the other classes. RPM then refines the set of motifs to the most representative and uses them to perform time series classification.

The way RPM works is to identify the motifs that are most representative of each class and use them to classify new time series. In order for RPM to identify representative patterns it first identifies the most frequent patterns or motifs within each class. It does so by concatenating the time series that are within the same class and performing motif discovery as discussed in Section 2.3.1. RPM takes care to avoid motifs that span concatenated time series by ignoring these subsequences.

However, because these are the most frequent motifs does not mean they are the most representative or class discriminative. Therefore, RPM first reduces the number of motifs by removing similar patterns. Then it selects the most representative patterns from this set of candidate patterns. For example, the most representative patterns in the ECGFiveDays dataset are seen in Figure 5.

To identify the most representative patterns RPM uses a correlation-based feature selection algorithm. To perform feature selection RPM first creates feature vectors for each example time series. The features for each class are calculated as the distance a given time series example is to each of the candidate patterns. TSAT implements both Euclidean distance and Dynamic Time Warping (DTW) distance algorithms (DTW is discussed in Section 5.8.1). The two dimensional feature vectors for the ECGFiveDays dataset are plotted in Figure 6.

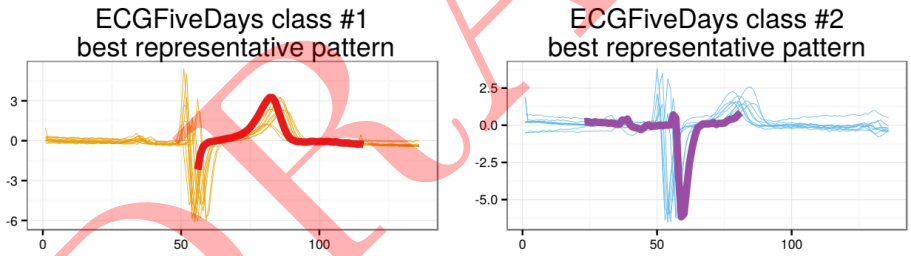

Figure 5: "Two classes from the ECGFiveDays dataset and the best representative patterns" [17]. Image taken from [17].

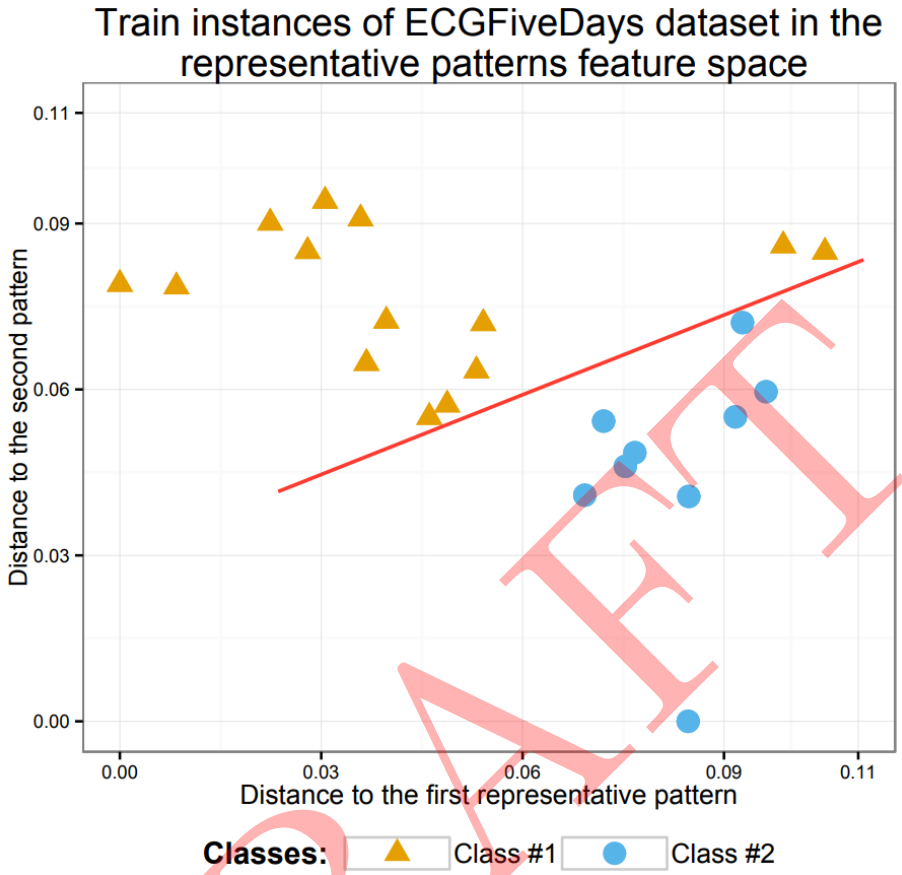

Figure 6: "Transformed data of train data from ECGFiveDays" [17]. Image taken from [17].

This distance feature vector is then used as the training data to a supervised classifier. TSAT uses Random Forest as the supervised classifier.

In order to set the optimal sliding window size, PAA size, and alphabet length (the SAX Parameter Combinations SPCs) for each class, RPM implements the DIRECT (Dividing Rectangles) parameter optimization algorithm. The error function is one minus the F1 measure from a five fold cross validation on the validation data. DIRECT will output the best SPCs so far therefore leaving it to the user to decide the number of iterations of the algorithm to perform. Leaving the number of iterations to the user is useful as running DIRECT it time intensive as it must perform the motif discovery and training the classifier for each new SPC. The number of iterations is discussed in more detail in Section 5.8.2.

2.4 Multiattribute Time Series

A multiattribute or a multivariate time series is one where a single example consists of multiple time series, all of the same length. **TSAT** implements Topological Data Analysis (TDA) in order to convert multivariate time series to univariate time series [18, 19, 20]. Then, with the created univariate time series, motif discovery, anomaly detection, or classification can be performed depending on the situation.

To run Topological Data Analysis (TDA) on a time series there are a number of parameters that need to be set. The *window size* or *window width*, this is the subsequence that TDA will compute persistence on, will take up to this number of samples. The value *dt* which is the

number of samples to skip between points. The value p integer (type of L^p norm to compute). Finally, there is *maxrad* which is the max distance between pairwise points to consider for the Rips complex.

Notes These are possibly useful notes when handling multiattribute time series.

If the features that represent anomalies are sparse and too far apart, then the window might not be long enough to capture the non-zero features together in the same window long enough to detect the voids between them as bubbles centered where they are grow outwards, in the rips complex, so maybe try increasing the size of the window.

3600 points seems like it might be enough, but enough variations in the distances between points in a window must appear in order to display interesting variation in the score function/-landscape L^p norms.

This method should work better on multivariate time series that have complex, weak correlations between them, but Gidea & Katz also showed that theoretically it should also be able to detect when independent time series drawn as random samples from the gamma distribution will show interesting scores indicating when the "heat" is turned up on the gamma shape parameter (flattening it).

One way to think about the features that the L^p norm is scoring that makes sense to me is a physical analogy: the more constrained the orbits of various particles in a dynamical system which are orbiting some attractor at a fixed speed, the larger the probability of their interaction, and so more triangles, etc should form, leading to a higher L^p norm in dimension 1. If the speed is turned up, and the sampling rate remains the same, the probability those interactions happen in the same window decreases, so we should see a dip in the L^p norm. If the system has orbits that are free to move in many directions and have fixed rates of interactions which cause a fairly steady value of L^p norm, is suddenly compressed onto a tighter orbital structure, like a torus or something, interactions will increase leading to a higher norm. The origin of using this idea came from analyzing dynamical systems, so I suspect organic-like data might be a good milieu for the technique.

With sparse data, it makes sense the landscape L^p norm is close to 0 a lot, perhaps except in a window including the non-sparse points. Say you have 3 time series which are all (0,0,0) in a window of 50 (so a 50x3 matrix of 0s). The rips complex is just going to be a bubble growing around the single point (0,0,0) for that window, and the 1st homology group H_1 is always going to be 0. If one of the points is (1,0,0) and the rest are (0,0,0), then a line segment will appear in the data between (1,0,0) and (0,0,0) at some point, but that is still equivalent to a point.

If you have (1,0,0), (0,1,0), (0,0,0) in the set, then now a triangle will appear and TDA will detect 1 tent function/landscape function with a positive area underneath it to integrate. Then, we should see a little spike in the score function or the output univariate time series at that point.

2.5 Overview

The rest of the user manual will go over in detail how to perform motif discovery (Section 3), anomaly detection (Section 4), and time series classification (Section 5) using TSAT's GUI.

3 Motif Discovery

Here the manual will go over the steps on how to format the time series and perform motif discovery in TSAT. How motif discovery works in TSAT is discussed in Section 2.3.1 in detail.

3.1 File format

In order to perform motif discovery in TSAT the time series must be formatted in a way that TSAT can read. TSAT requires that the time series be stored in a file where each line in the file contains one entry coresponding to a single time step in the time series. There must not be any missing lines. For example a correct file might look like:

```
-5.3
2.3
4
42
230
10
34
53
19
42
```

Multiattribute Time Series Format **TSAT** also handles multiattribute time series data however the format is in JSON (JavaScript Object Notation). The format is

```
[
{
        "timeSeries":[
        {
                "data":[
                2.0,
                100.3,
                10.4,
                11.4
                ]
        },
        {
                "data":[
                12.0,
                90.3,
                70.4,
                31.4
                ]
        }
        ]
```

```
}
]
```

This is a two attribute time series. Each time series must have the same length and is contained in the array labeled "data". Also, "json" must be present in the filename, e.g. "awesomeMTS.json".

3.2 Guide to Motif Discovery

Step 1 (Figure 7) Click "Browse" and browse for the time series data file and click "OK" when file is selected or "Cancel" if you wish to quit browsing. Then click the "Load data" button in the data source section and the time series will be displayed in the Data display section.

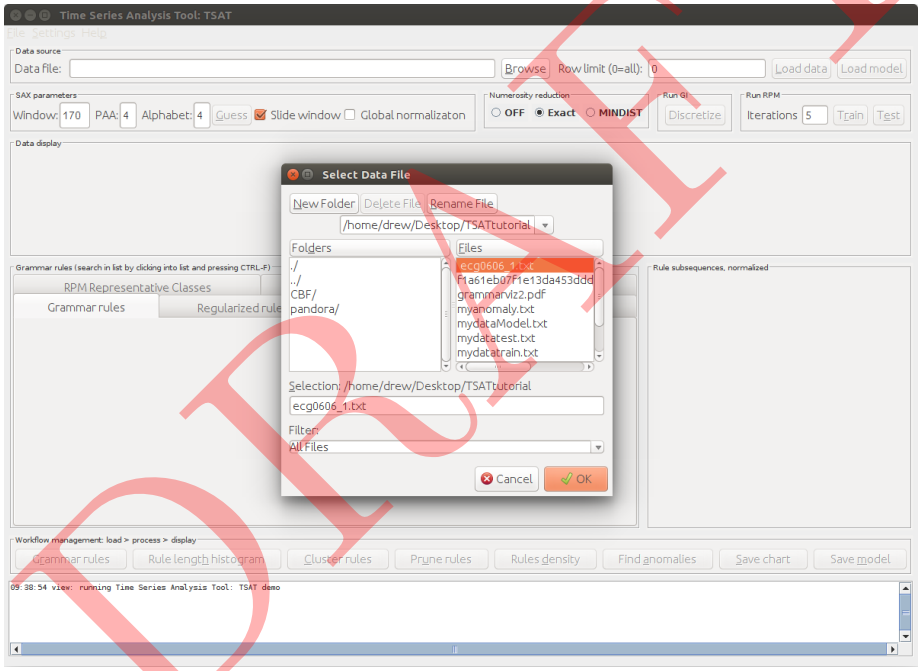

Figure 7: Click "Browse" → select file and click "OK" → click "Load data".

Step 1.1 Multiattribute If the training data is a multiattribute time series (with "json" appearing in the filename), then when the button labeled "Load Data" is clicked the Topological Data Analysis dialog will appear. This is illustrated in figure 8.

Once the values are set in the dialog and "Ok" is clicked, a new file dialog box will appear. This box requests the desired location and filename for the univariate time series data that will be generated from the multivariate time series data. This is shown in figure 9.

Once the file has been selected TDA will be run and the univariate time series will be created and the new univariate time series will be loaded into TSAT and displayed. Now that the multivariate time series has been converted to a univariate time series , continue on with the following paragraph, "Step 2".

Step 2 (Figure 10) Set the SAX parameters, "Window", "PAA", and "Alphabet" manually in the "SAX Parameters" section and then click "Discretize" in the "Run GI" section to produce

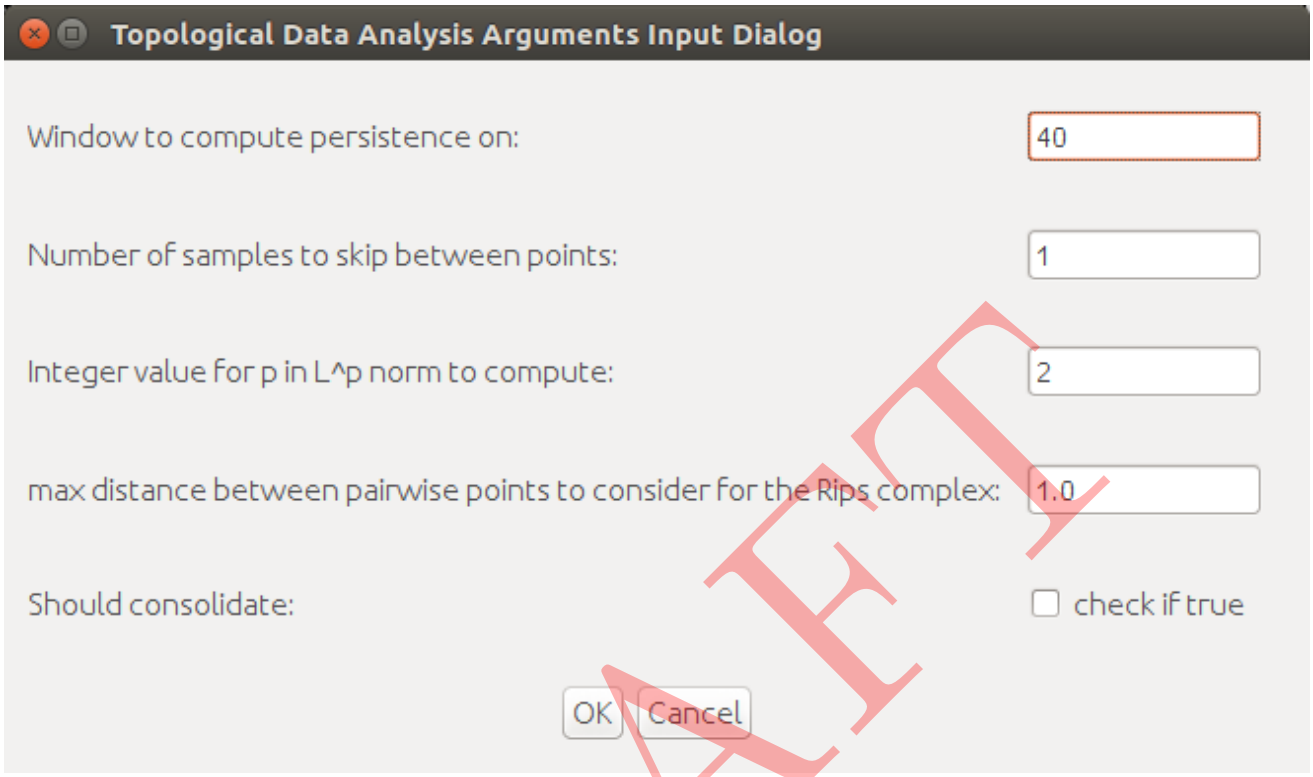

Figure 8: Topological Data Analysis dialog.

the motifs. Also, note that you can adjust the numerosity reduction strategy from Exact to either Off or MINDIST and toggling the sliding window and global normalization. How to set the SAX parameters is discussed in Section 2.3.1.

Figure 10: Set SAX Parameters and click "Discretize".

Step 3 (Figure 11) Evaluate results by clicking on the grammar rules in the "Grammar Rules" tab and seeing the subsequences highlighted in the "Data Display" section and graphed in the "Rule Subsequence" section. Each grammar rule row has nine column values: R#, Level,

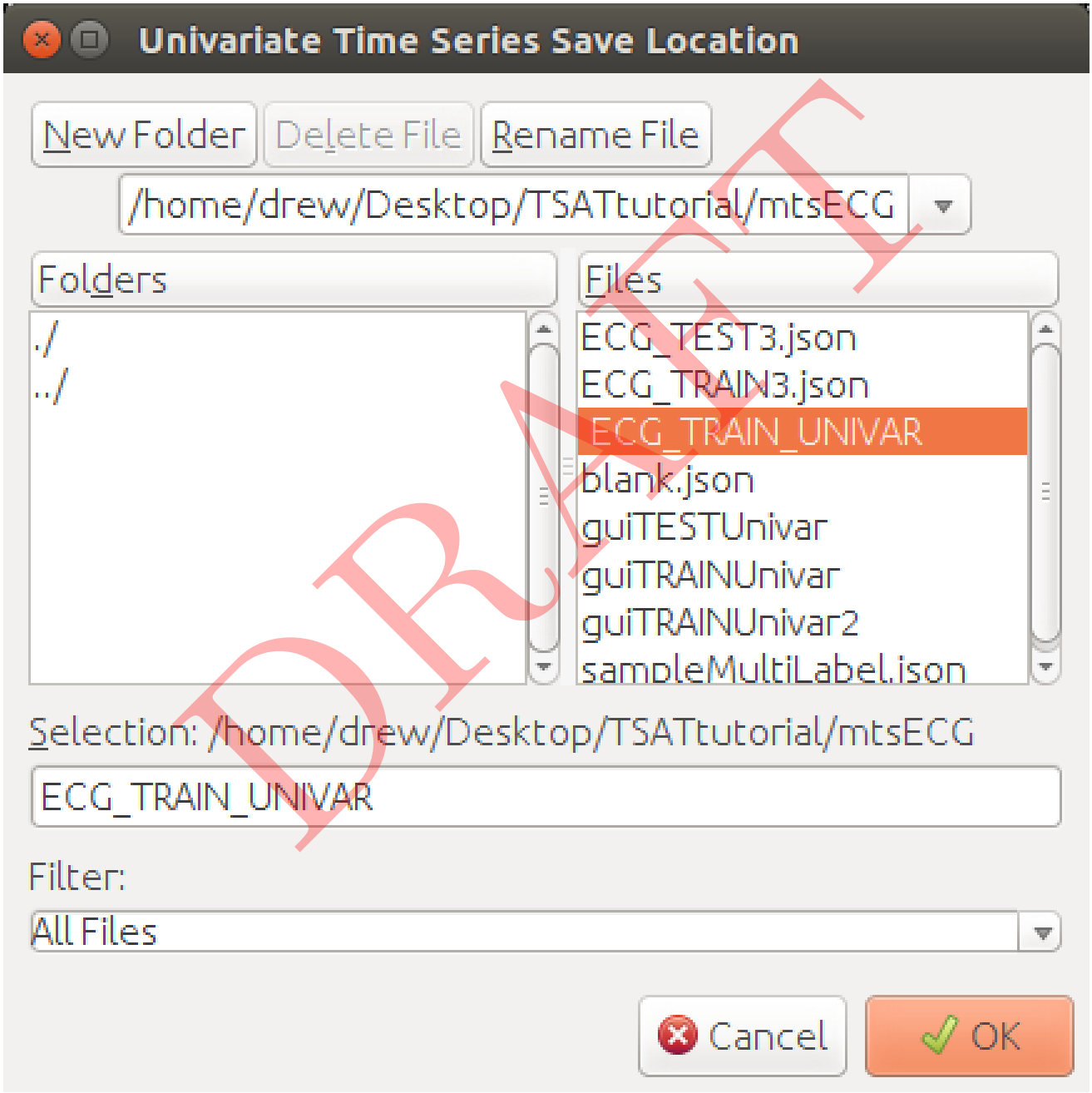

Figure 9: Location and filename for the univariate time series that TDA creates.

Frequency in R0, Rule, Expanded Rule, Used, Mean Length, and Min-max length. These values correspond to:

R# The rule number where rule number 0 is the root grammar rule.

Level The grammar level or the distance from the root rule.

Frequency in R0 The number of times this rule is used in the root rule.

Rule The actual grammar rule.

Expanded Rule The grammar rule that has its non-terminal symbols replaced with the terminal symbols.

Used The number of times that rule is used by other rules.

Mean Length The average length of the subsequence that this rule spans.

Min-max Length The minimum and the maximum length that the rule spans in the time series.

A selected rule or motif found by **TSAT** is shown in Figure 11. Note that multiple rules can be selected by holding the ctrl key and clicking on the rules or holding shift and using the arrow keys to move up and down.

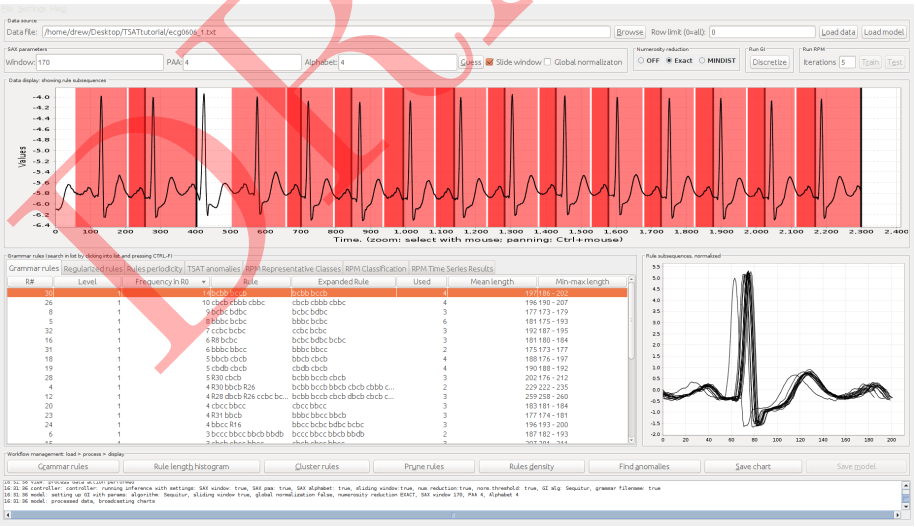

Figure 11: List of motifs found by TSAT in the "Grammar Rules" tab. Click on each rule to highlight them in the Data display and graph the subsequence in Rule Subsequence section. Multiple rules can be selected by holding the ctrl key and clicking on the rules or holding shift and using the arrow keys to move up and down.

Additionally, note that by clicking "Save chart" in the "Workflow management" section **TSAT** will save a png image of the data display area to a file located in the same directory as the jar file with a file name corresponding to yyyyMMddhhmmssSS.png.

Also, by clicking the "Rule length histogram" button in the "Workflow management" section **TSAT** will display a histogram of the rule lengths in Data Display area.

3.2.1 Guess SAX Parameters

Rather than manually trying different SAX parameters, TSAT has built in functionality to guess what it perceives as optimal parameters based on user defined range. This method uses the Re-Pair grammar induction algorithm and the entire process is beyond the scope of this manual but is described in detail in [21].

Step 1 (Figure 12) After loading the dataset from Step 1 in the previous section and instead of setting the SAX parameters manually, click "Guess." This will change the "Data Display" to read "Select the time series interval for guessing." Next, Press and hold the left mouse button and drag the mouse across the time series until the desired subsequence of the time series is highlighted and then release the mouse button. Note that the selected subsequence should be free of anomalies and noise otherwise the guess may produce biases in the results.

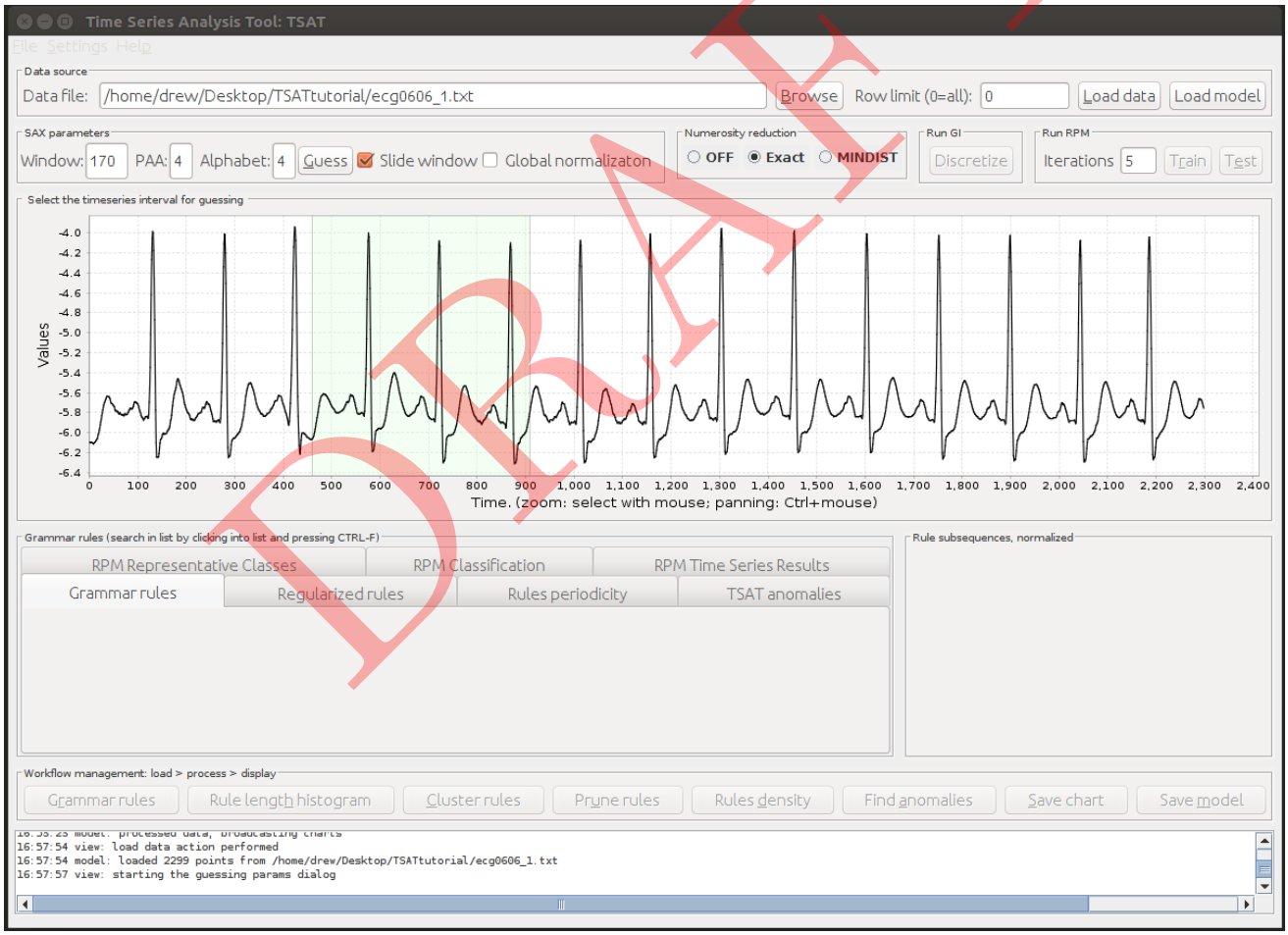

Figure 12: Click "Guess" and then select time series subsequence by clicking and dragging without anomalies or noise. The green highlighted region is the selected region.

Step 2 (Figure 13) After releasing the mouse button a dialog will appear with the title "Sampler interval and parameter ranges verification." Here you may adjust the values as appropriate and then click "OK" otherwise if you wish to cancel press "Cancel."

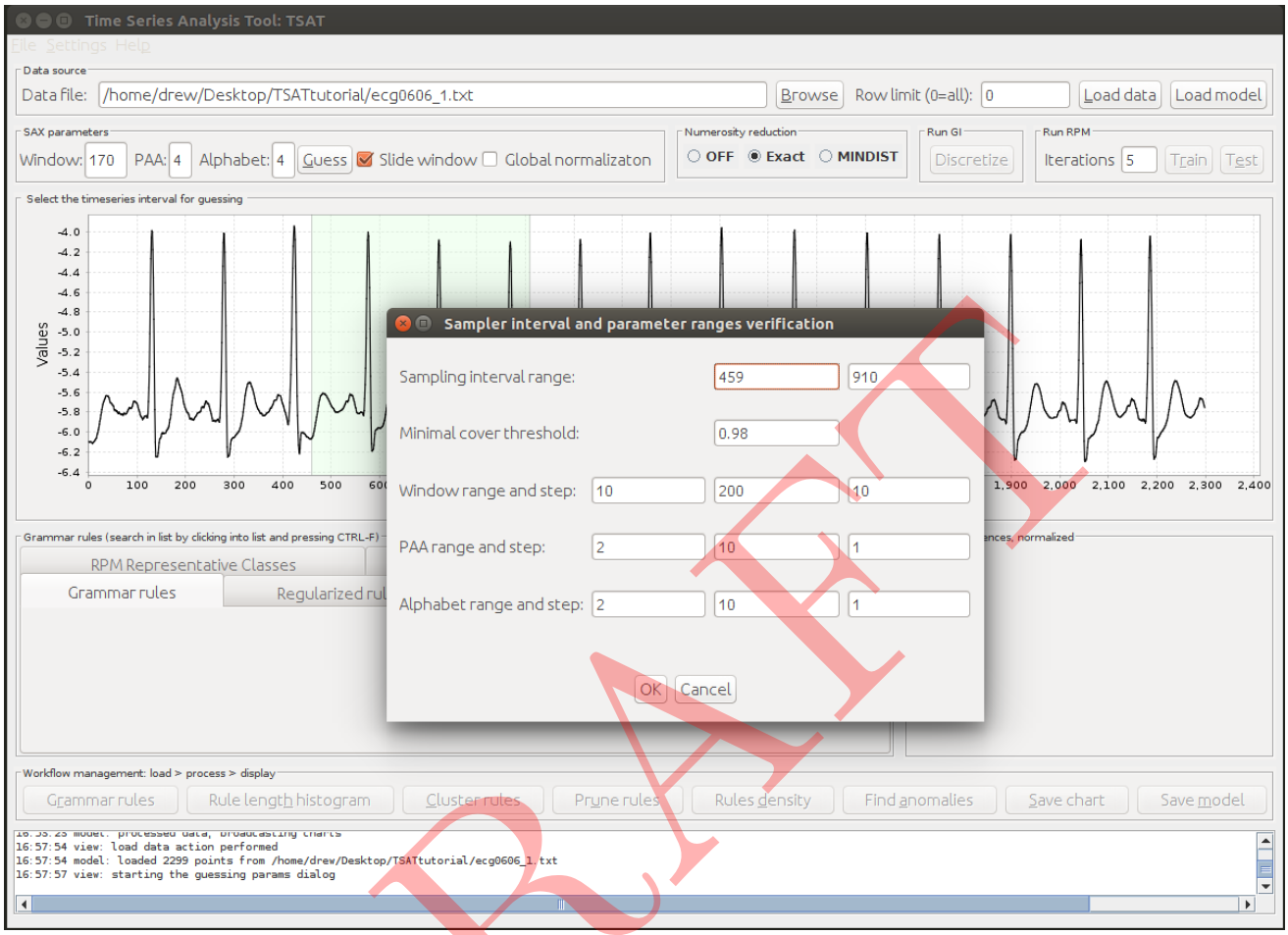

Figure 13: Adjust the range values as appropriate to your time series. Note that the larger the range the longer it will take to search.

Step 4 (Figure 14) If you have pressed "OK" then the process will beginning and if you wish to stop it press "stop!" and TSAT will stop searching for the SAX parameters. Otherwise after a period of time the values in the "SAX Parameters" section will be replaced with the parameters that guessing mechanism found.

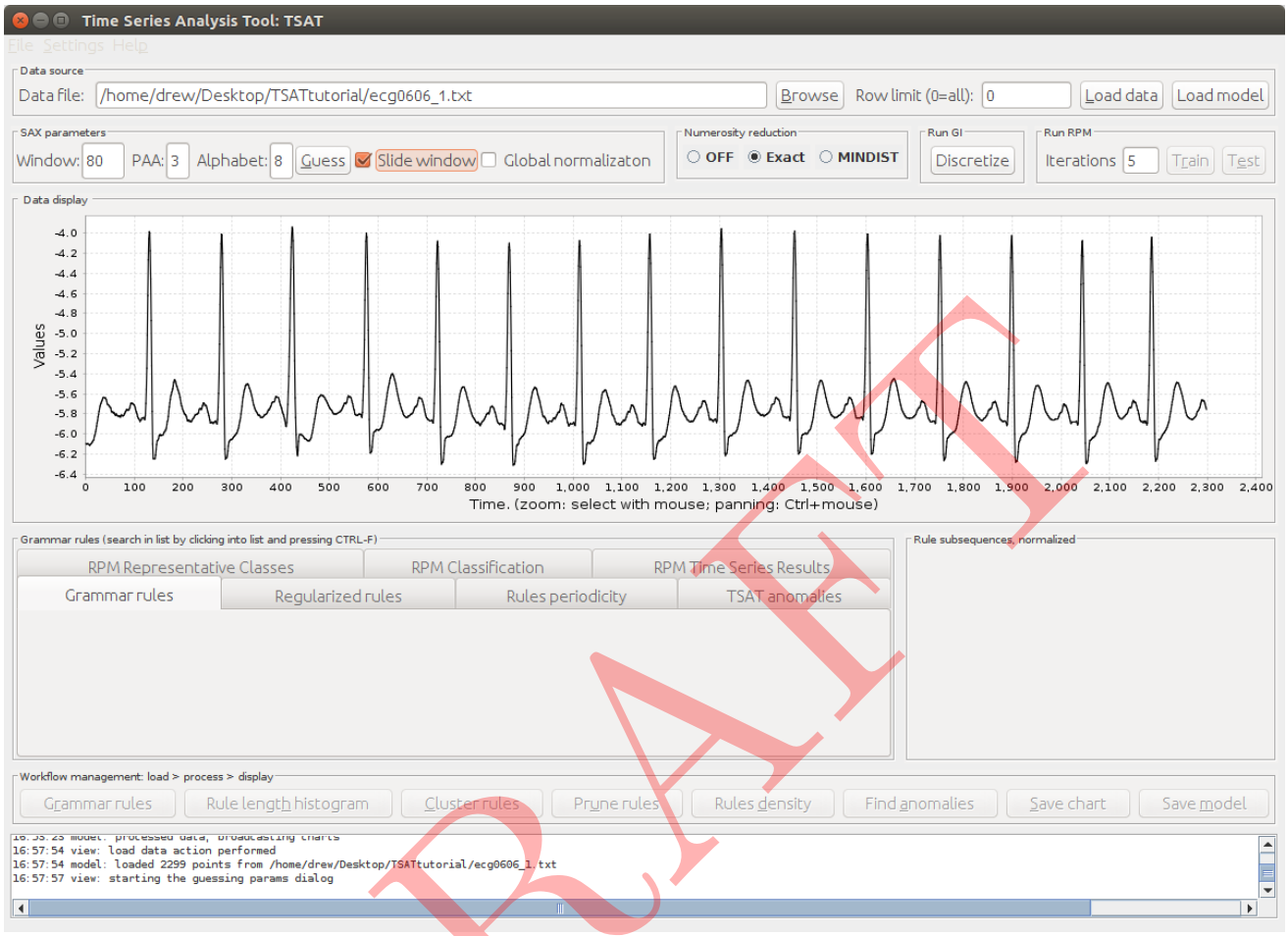

Figure 14: The inferred SAX parameters are filled in the "SAX parameters" section.

3.2.2 Options

The settings for motif discovery are is accessed by clicking the menu item "Settings" and then under that click "TSAT options" in order to open the settings. There are a number of settings including choice of grammar induction, normalization threshold, and location for the output.

Grammar Induction The default grammar induction algorithm is Sequitur. However, Re-PAIR is also an option.

Then click on the "GI Implementation" tab and the option for RePAIR is presented. Click on the desired radio button and then click "save" to keep your selection or click "cancel" to quit.

Normalization Threshold The normalization threshold is default value of 0.05 is usually appropriate but is adjustable under the "Options" tab.

Output Currently the Output tab does not affect the location of output files other than for the Rule density curve filename.

3.3 Rule Pruning

TSAT provides two means for pruning the motifs or the rules: clustering and a greedy pruning algorithm. Pruning is useful in order to gain better visual insight into the motifs of the time series that are of importance. This section assumes that the desired time series has been loaded and discretization has been performed following Section 2.3.1.

3.3.1 Clustering Technique

The clustering method prunes the motifs by classifying the subsequences by length and removing the overlapping in the same length range. This technique is accessed by clicking on the "Cluster rules" button in the Workflow management section of the GUI.

Once clicked you are asked for "threshold for length" and a "threshold for overlap." The threshold for overlap is not actually currently used. However the value for the threshold length is used. This value is a value between zero and one and states how similar the lengths of the sub=sequences need to be in order to be within the same class. So, if the threshold is 0 then they must be exactly the same length and if one then they have to be at least as long as the sub-sequence being compared to. So, a value of .1 (a distance of 10 for a length of 100 sub-sequence) is a pretty good standard. Then within the classes if two sub-sequences are overlapping and the difference in the overlap is above some threshold then one of the sub-sequences will be pruned.

Once finished the results are presented in the "Regularized rules" tab in "Grammar Rules" section.

3.3.2 Greedy Rule Pruning

The other approach to rule pruning uses a greedy algorithm following a greedy solution of minimum-cardinality set cover problem (an NP-hard problem). This is an attempt to find the smallest set of rules which cover the most of the input time series in a greedy fashion. According to [22]:

> The intuition behind this algorithm is simple – since our task in hand is to find maximally repeated and minimally-overlapping subsequences (which we consider the most informative), at each iteration, as the best candidate we select the rule which covers the most of the uncovered-so-far time series span thus naturally provides the most information about its structure.

This functionality is experimental still. However, it can be used after discretization by clicking "Prune rules" in the "Workflow management" section of the GUI. This will update the "Grammar rules" tab in the "Workflow management" section with the pruned rules.

3.4 Python Interface

The python interface for motif discovery is accessed via the buildMotifs function that has the function signature:

```
tsail.buildMotifs(pathToTimeseries, outputFile, window_size=30, word_size=6,
    alphabet_size=4, strategy="EXACT", threshold=0.01, numworkers=2)
```

The parameters are as follows:

pathToTimeseries The path as a string to the location of the time series data

outputFile The name of the file that the motifs will be written to.

window_size The SAX parameter for the window size with a default value of 30.

word_size This is the same as the PAA value and indicates the length of the SAX words. Default word length is 6.

alphabet_size This is the same as Alphabet and indicates the number of characters or symbols that can be chosen. Default value is 4.

strategy This is the numerosity reduction strategy to be used. The default value of EXACT should be used in most cases. Other valid values are NONE and MINDIST.

threshold SAX normalization threshold meaning that if the input time series' standard deviation is less than this value the z-normalization will not be applied. Default value of 0.01.

numworkers This is the number of threads to use when running SAX. The default value is 2.

For buildMotifs it returns a dictionary where you can index the motifs (or rules generated by Sequitur).

`<returned_dict>` is the GrammarRules object that was returned as a result of the call to buildMotifs and has a rules map indexed by the integer rule number starting at `'0'`.

https://github.com/jMotif/GI/blob/master/src/main/java/net/seninp/gi/logic/GrammarRules.java

`<returned_dict>['rules']['rule_number']` will give you a GrammarRuleRecord for a particular `rule_number`:

https://github.com/jMotif/GI/blob/master/src/main/java/net/seninp/gi/logic/GrammarRuleRecord.java

The list of members accessible in each GrammarRuleRecord follows:

```
/* The rule number in Sequitur grammar. */
private int ruleNumber;


/* The rule string, this may contain non-terminal symbols. */
private String ruleString;

/* The expanded rule string, this contains only terminal symbols. */
private String expandedRuleString;


/* The indexes at which the rule occurs in the discretized time series. */
private ArrayList<Integer> timeSeriesOccurrenceIndexes = new ArrayList<Integer>();


/* This rule intervals on the original time series. */
private ArrayList<RuleInterval> ruleIntervals;
```

```
/* The rule use frequency - how many time that rule is used by other rules. */
private int ruleUsageFrequency;

/* The rule level in the hierarchy */
private int ruleLevel;

/* The rule's minimal length. */
private int minLength;

/* The rule's maximal length. */
private int maxLength;

/* The rule mean length - i.e. mean value of all subsequences corresponding to the
    rule. */
private Integer meanLength;

/* The rule mean period - i.e. the mean length of intra-rule intervals. */
private double period;

/* The rule period error. */
private double periodError;

/* The rule yield - how many terminals it produces in extended form. */
private int ruleYield;
```

For example, `<returned_dict>['rules']['1']` will give you the grammarRuleRecord for rule 1. The values within the GrammarRuleRecord can then be accessed as you do a python dictionary.

Also, note that ruleIntervals is an array of:

https://github.com/jMotif/GI/blob/32f58578f5a0b184fc836f9d397aa0bfc8e68ee6/src/main/java/net/seninp/gi/logic/RuleInterval.java

To access this you just do:

`<returned_dict>['rules']['1']['ruleIntervals'][<'rule_interval_index'>]`

For example, `<returned_dict>['rules']['1']['ruleIntervals'][0]`

Each RuleInterval has the following properties:

```
public int id; // the corresponding rule id
public int startPos; // interval start
public int endPos; // interval stop
public double coverage; // coverage or any other sorting criterion
```

4 Anomaly Detection

4.1 Guide to Anomaly Detection

Anomaly detection is used to identify surprising patterns which correspond to rare grammar rules or motifs. This section assumes that you have already performed the steps from Section 3 and have loaded a time series and have produced the motifs. This section will cover both the approximate and exact

forms of anomaly detection using rule density and the Rare Rule Anomaly detection algorithm.

Rules Density After following the steps in Section 3 of loading the dataset and identifying the motifs click the "Rules Density" button in the "Workflow management" section of the GUI. Once clicked the results will be displayed in the Data display. White corresponds to a zero density indicating an anomaly and the darker the color blue the less likely an anomaly exists in that subsequence. The rule density is displayed in Figure 15.

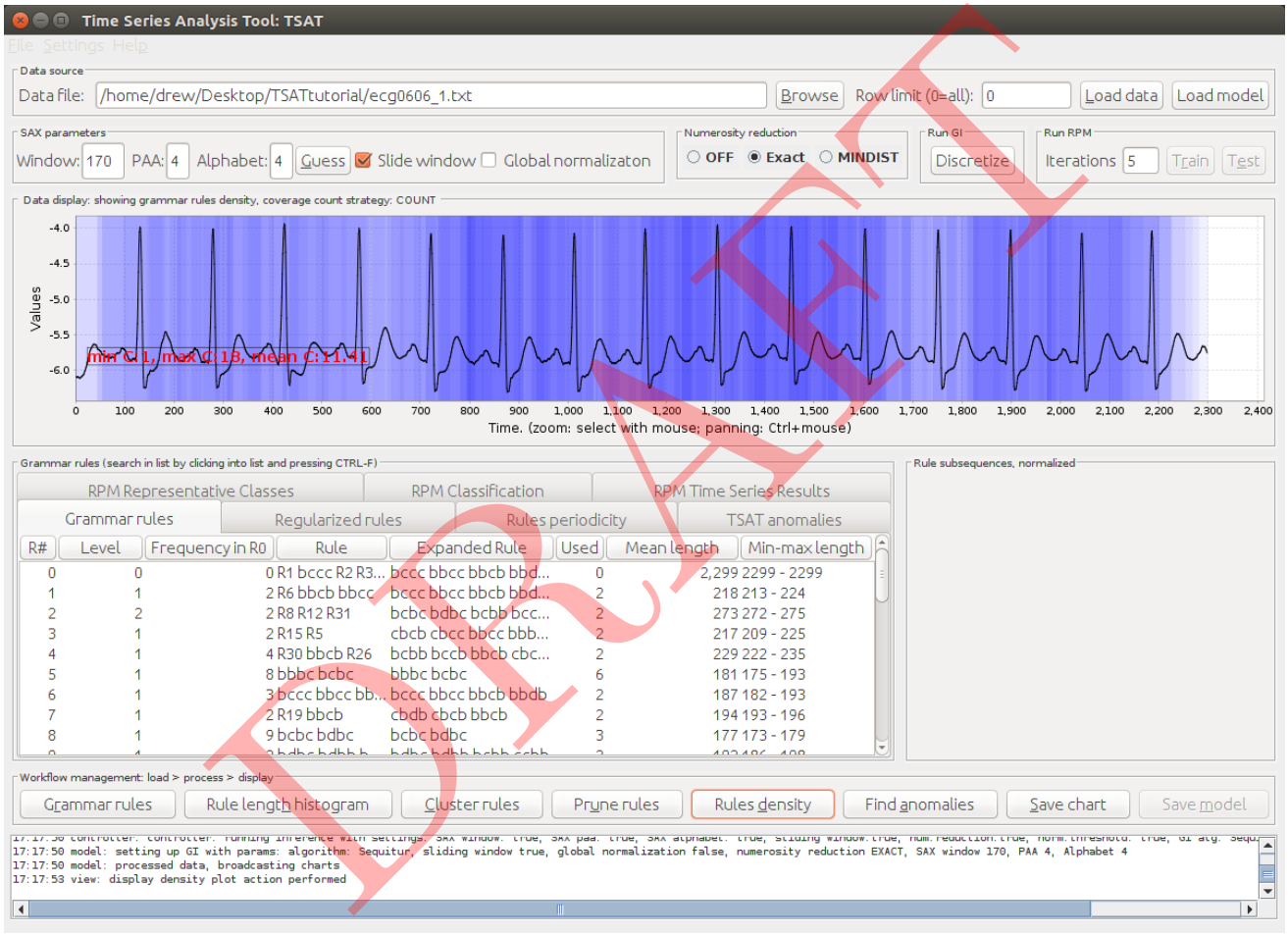

Figure 15: Rule density plot in the Data display area.

By clicking on "Settings" and then "TSAT options" menu items and a dialog window will appear and you can choose a coverage strategy in the "Coverage Strategy" tab (shown in Figure 16). Changing the coverage strategy will change how the rule density is calculated. The default is rule count which corresponds to the number of grammar rules that include that particular timestamp. This is how rule density is usually calculated. Other options include rule level increment, this corresponds to summing up the rule levels that each rule is at in the grammar at each timestamp. Rule occurrence, corresponds to summing up the number of times this rule is applied in the time series. Rule yield, corresponds to summing up the number of terminals that the particular grammar rule yields. The product of level and occurrence is just the product of these two summed for each of the grammar rules at each timestamp.

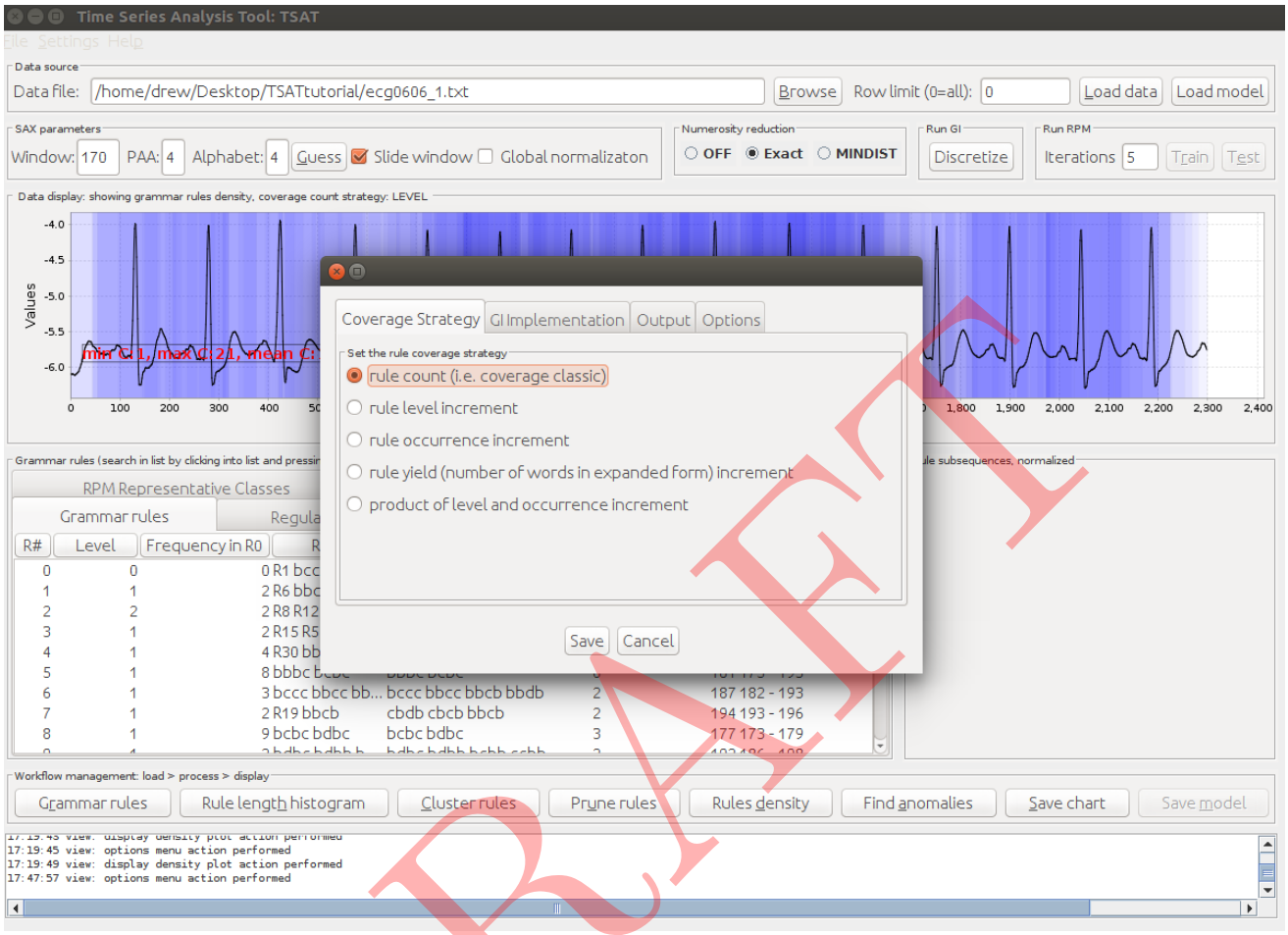

Figure 16: Changing the coverage strategy.

Rare Rule Anomaly Detection The exact strategy for finding anomalies in **TSAT** is by using the Rare Rule Anomaly detection algorithm. This is done by clicking "Find anomalies" in the "Workflow management" section and then clicking on the "TSAT anomalies" tab in the "Grammar rules" section. The top 10 anomalies will be listed in the table in the "TSAT anomalies" tab. Rank, Position, Length, NN Distance, and Grammar Rule are the columns. Also, when an anomaly is clicked on, the subsequence is highlighted in the Data display and shown in the Rule subsequences section. Multiple anomalies can be selected by holding Ctrl and then clicking. The anomaly in the data is shown in Figure 17.

Rank The smaller the value the more anomalous the subsequence.

Position The start location of the anomaly.

Length The length of the subsequence containing the anomaly.

NN Distance The distance to the closest subsequence. The larger this value is the smaller the value Rank is.

Grammar Rule The grammar rule that corresponds to the anomaly. Can return to the Grammar Rules tab and find the rule based on this number.

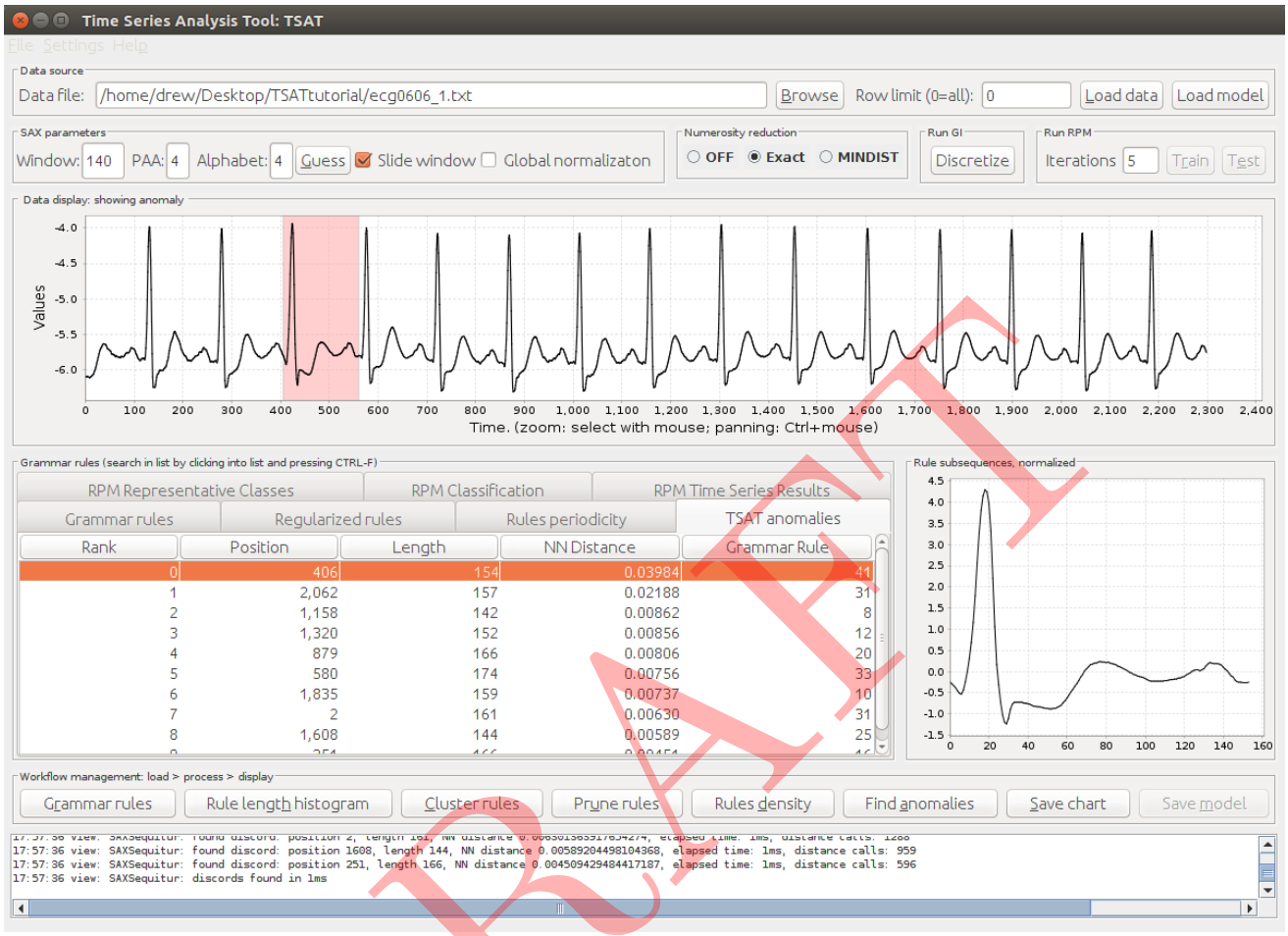

Figure 17: Rare Rule Anomalies listed in "TSAT anomalies" tab.

4.2 Python Interface

The function signature for calling anomaly detection using the Rare Rule Anomaly algorithm is:

```
RRA(pathToTimeseries, outputFile, window_size=30, word_size=6, alphabet_size=4,
    threshold=0.01, discords_num=5)
```

With the following parameter definitions: The parameters are as follows:

pathToTimeseries The path as a string to the location of the time series data

outputFile The name of the file that the motifs will be written to.

window_size The SAX parameter for the window size with a default value of 30.

word_size This is the same as the PAA value and indicates the length of the SAX words. Default word length is 6.

alphabet_size This is the same as Alphabet and indicates the number of characters or symbols that can be chosen. Default value is 4.

threshold SAX normalization threshold meaning that if the input time series' standard deviation is less than this value the z-normalization will not be applied. Default value of 0.01.

discords_num This is the number of discords to report. The default value is 5.

Which returns a dictionary holding the discord record.
`returned_dict = RRA(...)`
When calling RRA you are returned a dictionary representation of DiscordRecords:
https://github.com/jMotif/SAX/blob/27607baa823df21a10d10e80827ffdd15090cbd9/src/main/java/net/seninp/jmotif/sax/discord/DiscordRecords.java
Which is a list of DiscordRecord based on:
https://github.com/jMotif/SAX/blob/660e837edf1c8058eac6ef05185c7f83e12f3689/src/main/java/net/seninp/jmotif/sax/discord/DiscordRecord.java
So, a DiscordRecord can be accessed by doing `<returned_dict>['discords'][0]`
Each DiscordRecord has the following properties:

```
/** The discord id (used when wrapped by RRA). */
private int ruleId;

/** The discord position. */
private int position;

/** The discord length. */
private int length;

/** The NN distance. */
private double nnDistance;

/** The payload - auxiliary variable. */
private String payload;

/** The info string - auxiliary variable. */
private String info;
```

For example, length of the DiscordRecord 0 can be accessed as: `<returned_dict>['discords'][0]['length']`

5 Time Series Classification using RPM

TSAT implements Representative Pattern Mining or RPM (see Section 2.3.4 for more details) to perform time series classification. In order to perform time series classification you will need a training and a test dataset containing time series data.

The standard method to train a supervised learning classifier is to take the labeled dataset and split it into two datasets, training and testing data. One common way to split the data is to have 80% training and 20% testing.

Training Data Training data is the primary data and will be used to create a model that can identify similar patterns in new, unlabeled, data. This data must have a label for each time series so that RPM can learn what the labels can look like. For RPM to work there must be more than one training label and there must be more than one example for each of the different training labels. This is where the bulk of the data should be set aside for as RPM will need many samples to find representative patterns.

Testing Data Testing data is a small subset of the data usually from the same source as the training data, but not found in the training data. This set of data will be used to test the model that RPM made for accuracy or some other statistic or to predict labels for unlabeled test data.

Splitting data into a training and a test set is beyond the scope of this manual and is not done by TSAT. The goal of this section is to first detail the proper file formats for training and testing data in Section 5.1. Then the proper procedures to train (Section 5.2) and test (Section 5.3)are presented step by step along then with a number of other useful features.

5.1 File formats

File formatting is very important in TSAT and especially when using RPM. If the file is not in the correct format TSAT will not be able to read the file and may produce unexpected results or error messages. The data may be formatted by column, row, or following the ARFF file format. Additionally, the labels for the time series may be any string excluding white space and "?" as this is reserved for unknown values in test data.

Figure 18: Examples of RPM Data

```
# 1 1 1 2 2 2 2 2 2
0 0 0 122880 122880 122880 122880 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 122880 122880 0 0 0
```

(a) Example 1

```
# 1.0000000e+000 1.0000000e+000 1.0000000e+000
-4.6427649e-001 -8.9697208e-001 -4.6469596e-001
-5.5504787e-001 -6.8568553e-001 -5.6773891e-001
-8.4284310e-001 -1.3513818e+000 -3.2022764e-002
-8.6589548e-001 -1.4586668e+000 -6.3504562e-001
-9.3639631e-001 -1.1653456e+000 -6.0282554e-001
-8.1726995e-001 -1.4039293e+000 -2.6685628e-001
-2.6361216e-001 -1.8217996e+000 -2.6706128e-001
-1.2580483e+000 -8.3160109e-001 -9.3104230e-001
-1.2503934e+000 -1.0163124e+000 -4.4938186e-001
-9.1830825e-001 -8.0353040e-001 -7.2134200e-001
-9.2210226e-001 -1.2595048e+000 -3.9727192e-001
-9.8448828e-001 -1.1392341e+000 -9.6212589e-001
-1.2880511e+000 -8.7865203e-001 -1.4206669e+000
```

(b) Example 2

Column Formatted Data The data files are simple text files that store the time series data with one entry per column, with a space delimiter, with each row representing a time step in the time series data. With RPM compatible data the first row in the file starts with a "#" with rest of the row containing the label for each time series rather then the time series values. If the file is missing this row RPM will not be enabled in TSAT. Examples of column formatted RPM compatible data can be seen in figure 18. Another thing to keep in mind is that in this format the time series must all be the same length.

Row Formatted Data Another acceptable format is the row format. This format is especially useful when the time series are not all the same length as each row or time series may have its own length. In this format the first line of the file is a "#" followed by a new line. Starting on the second line, each line starts with the label followed by the corresponding time series (each value separated by a space). There should be no empty lines. For example,

```
#
1 -5.3 -23 5 ...
1 23 1 5 3 1 ...
two 23 3 4 200 ...
```

```
two 42 3 4 102 ...
...
```

In this example the labels are "1" and "two" and the time series follow after the labels.

ARFF Formatted data A standard format for many public time series datasets is the ARFF file format. This file format is accepted in **TSAT** though the file must contain "arff" in the filename, e.g. "awesomeTS.arff". Also, http://timeseriesclassification.com/dataset.php has a number of time series in ARFF format that can be used in TSAT. ARFF files are more complicated than both the column and row formats, but is more widely used outside TSAT. Here is an abbreviated example ARFF file:

```
@relation Adiac

@attribute att0 numeric
@attribute att1 numeric
...
@attribute target 1,2,...

@data
1.3749,1.2894,1.2043,1.1194,1.0347, ...  1
1.7257,1.7001,1.6611,1.6089,1.5319, ...  2
...
```

The ARFF file begins with the name of the dataset (above the name is Adiac) by using the ARFF formatting by putting it after the `@relation` element. After the name of the dataset each timestep is listed as an attribute `@attribute <timestepName> numeric` where you can choose what to name each timestep. After listing the timesteps as attributes the labels are listed as the target attribute `@attribute target {1, 2, ...}` where these are the labels for the time series. Finally, the time series data is in comma separated value (CSV) format following the `@data` line. Each value in a time series is separated by a comma on a single line and the last value on the line is the label for the time series.

Multiattribute Time Series Data For a multiattribute time series the data is formatted in a JSON (Javascript Object Notation) file. Like arff files, json must be present in the filename, e.g. "awesomeMTS.json". There is at the point of this manual no recognized standard for multivariate time series data. ARFF files are a bit cumbersome to use when dealing with multiattribute time series. Therefore, **TSAT** uses JSON format which is much easier to use.

An example of the format is:

```
[
{
        "timeSeries":[
        {
                "data":[
                2.0,
                100.3,
                10.4,
                11.4
                ]
        },
        {
```

```
                "data":[
                12.0,
                90.3,
                70.4,
                31.4
                ]
        }
        ],
        "label":"1"
},
{
        "timeSeries":[
        {
                "data":[
                20.0,
                10.5,
                109.4,
                14.6
                ]
        },
        {
                "data":[
                112.8,
                23.1,
                64.1,
                32.7
                ]
        }
        ],
        "label":"1"
}
]
```


The above can be viewed as an array of MultivariateTimeSeries objects. Where the Java class definition is below.

```
        private class MultiVariateTimeSeries {
                public TimeSeries[] timeSeries;
                public String label = null;
        }

        private class TimeSeries {
                protected double[] data = null;
        }
```

Each multiattribute or multivariate time series consists of an array of the time series and the associated label. As always there must be more than one class label (therefore the above is not valid for RPM since "1" is the only label) and there must be more than one example for each class label.

Unknown Test Data In column, row, or ARFF format when predicting unlabeled test data, the test data must be labeled as "?" (note that there must only be test data that is labeled with a "?").

For example, a row formatted test dataset might be:

```
#
?  -5.3 -23 5 ...
?  23 1 5 3 1 ...
?  23 3 4 200 ...
...
```

As can be seen the label is "?" and the time series follows after the label. When training there must always be more than one example from each class label and there must be more than one label.

5.2 Training the Model

Once you have the data in the proper format, training RPM can begin.

Step 1 First click on the "Browse" button under the "Data Sources" section of the window, as seen in figure 19.

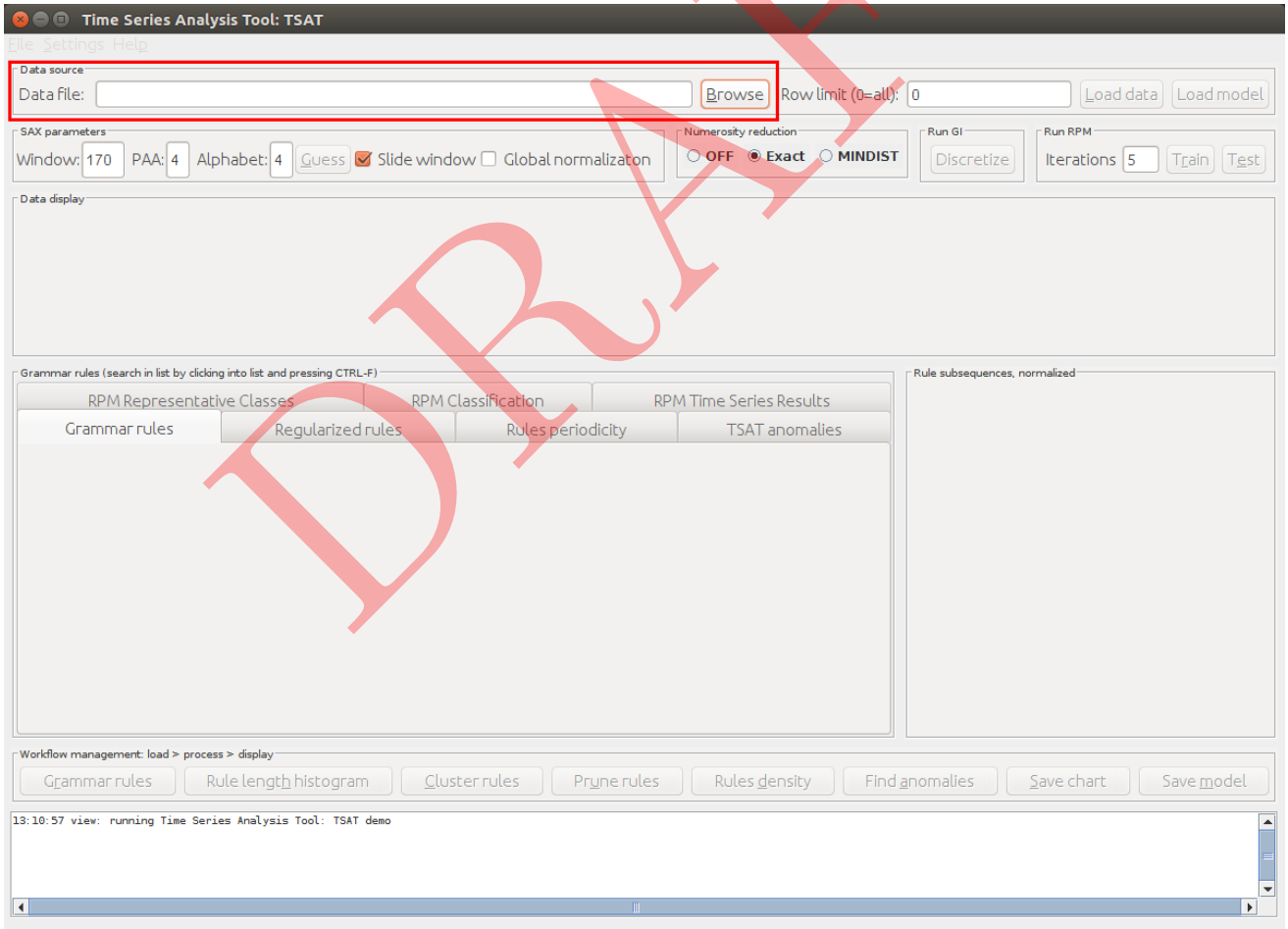

Figure 19: Open TSAT

Step 2 This should bring up the file browser prompt in figure 20. Using this prompt select the file containing the training set in the RPM compatible format, figure 21.

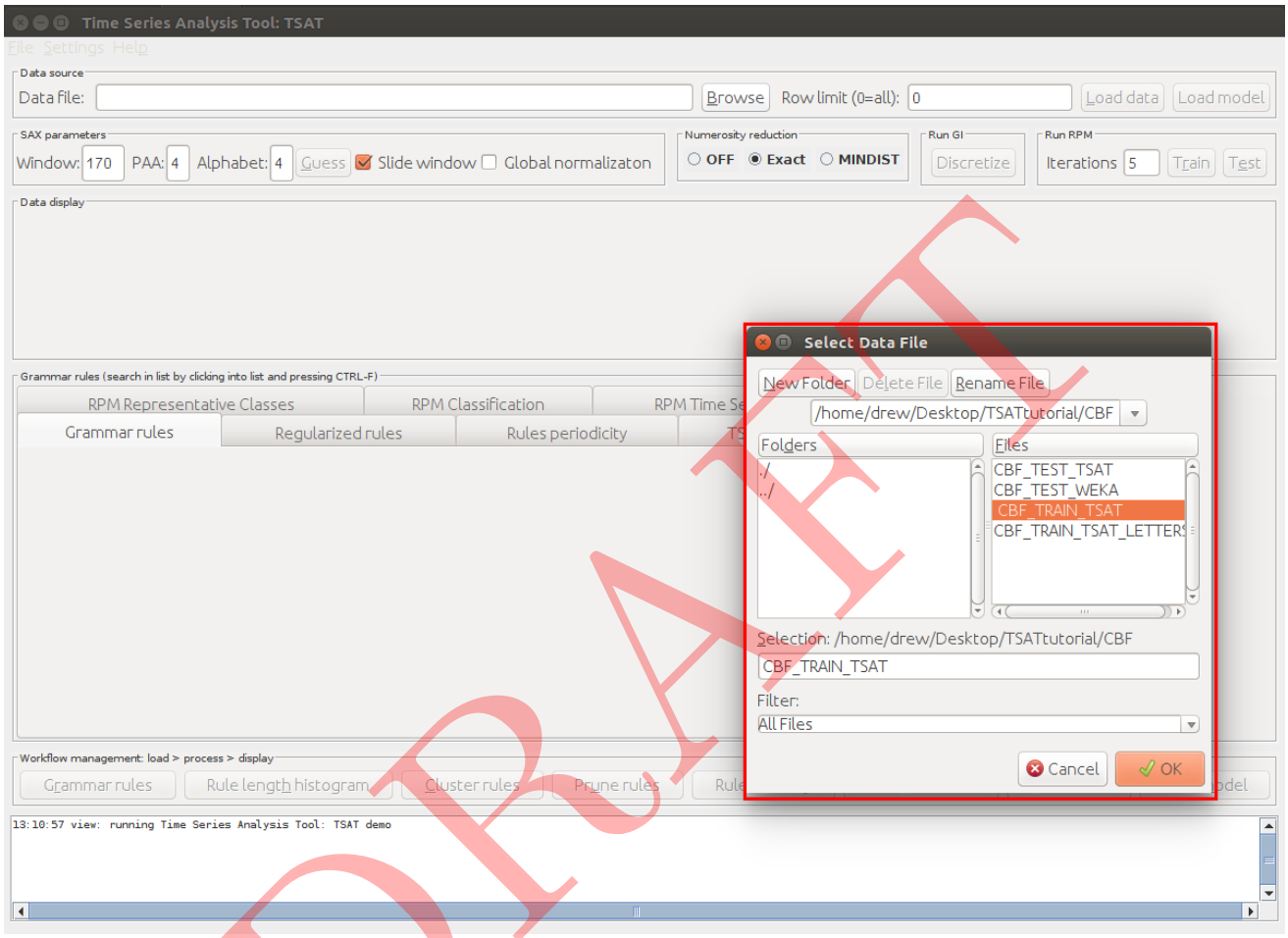

Figure 20: Open the file browser prompt

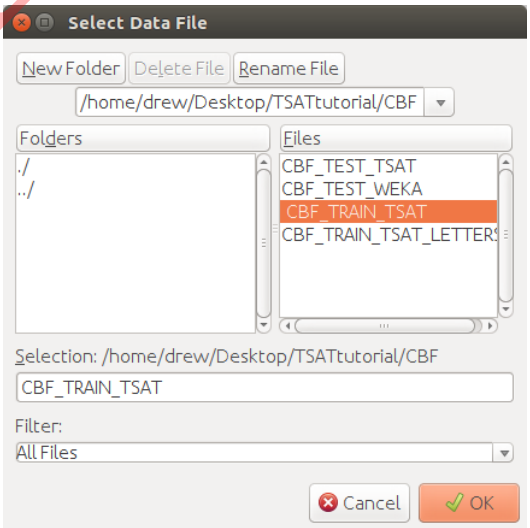

Figure 21: Browser prompt

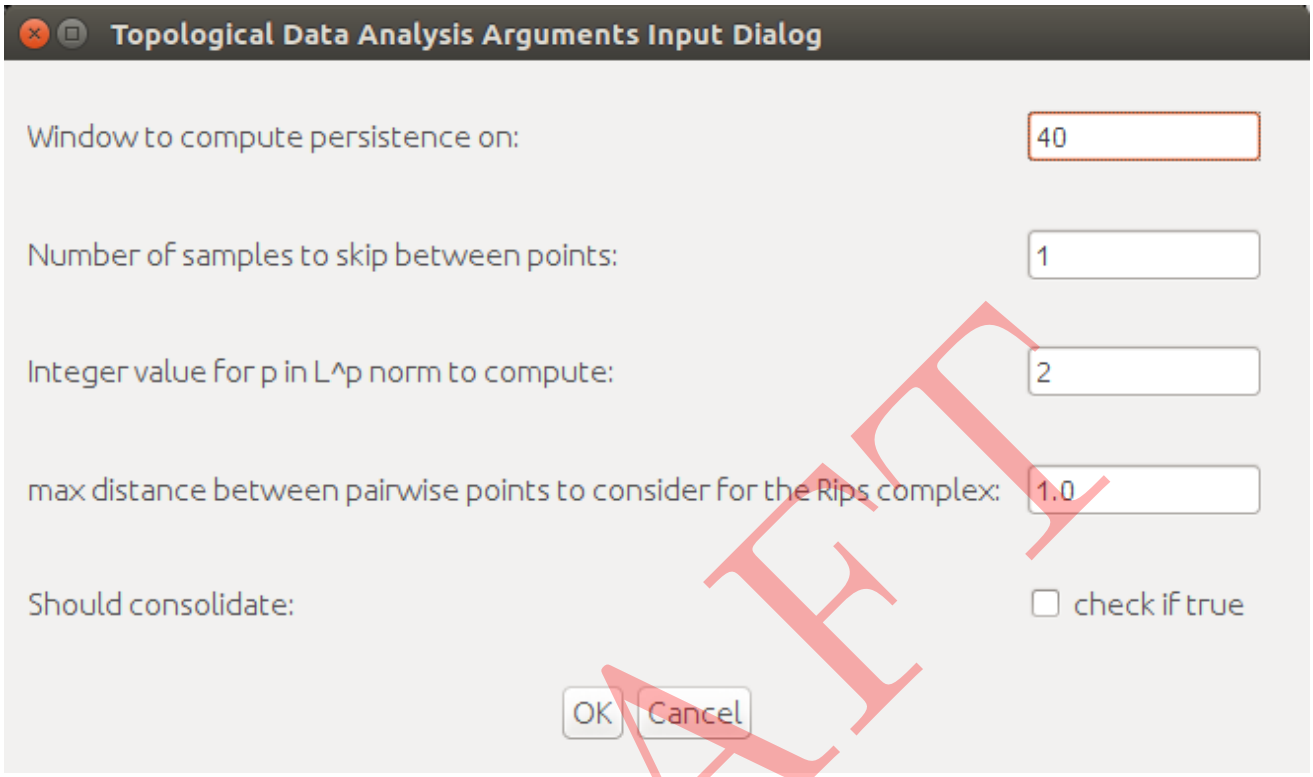

Figure 22: Topological Data Analysis dialog.

Step 3 After selecting the file press the button labeled "Load Data" and TSAT will load the data and the graphs will be populated, and if the data is found to be RPM compatible data then the "Train" button should become available. The text field labeled "Row Limit" allows the user to limit the number of rows that are read in from file, for example if the file contains 100 rows the user could limit it to the first 50.

Step 3.1 Multiattribute If the training data is a multiattribute time series (with "json" appearing in the filename), then when the button labeled "Load Data" is clicked the Topological Data Analysis dialog will appear. This is illustrated in figure 22.

Once the values are set in the dialog and "Ok" is clicked, a new file dialog box will appear. This box requests the desired location and filename for the univariate time series data that will be generated from the multivariate time series data. This is shown in figure 23.

Once the file has been selected TDA will be run and the univariate time series will be created and the new univariate time series will be loaded into TSAT and displayed. Now that the multivariate time series has been converted to a univariate time series, continue on with the following paragraph, "Step 3 continued".

Step 3 continued If the time series data is successfully loaded then all of the time series will be graphed in the "data display" area. The data can be zoomed in to by left clicking and dragging the mouse over the area you wish to zoom into. In order to zoom back out, right click on the graph area, hover over "Auto Range" in the menu, and then click "Both Axes." Other features available after right clicking are to save the plot, print, copy, zoom in, and zoom out, and auto range. The purpose of this display is to allow the user to gain an intuition of what the time series data looks like.

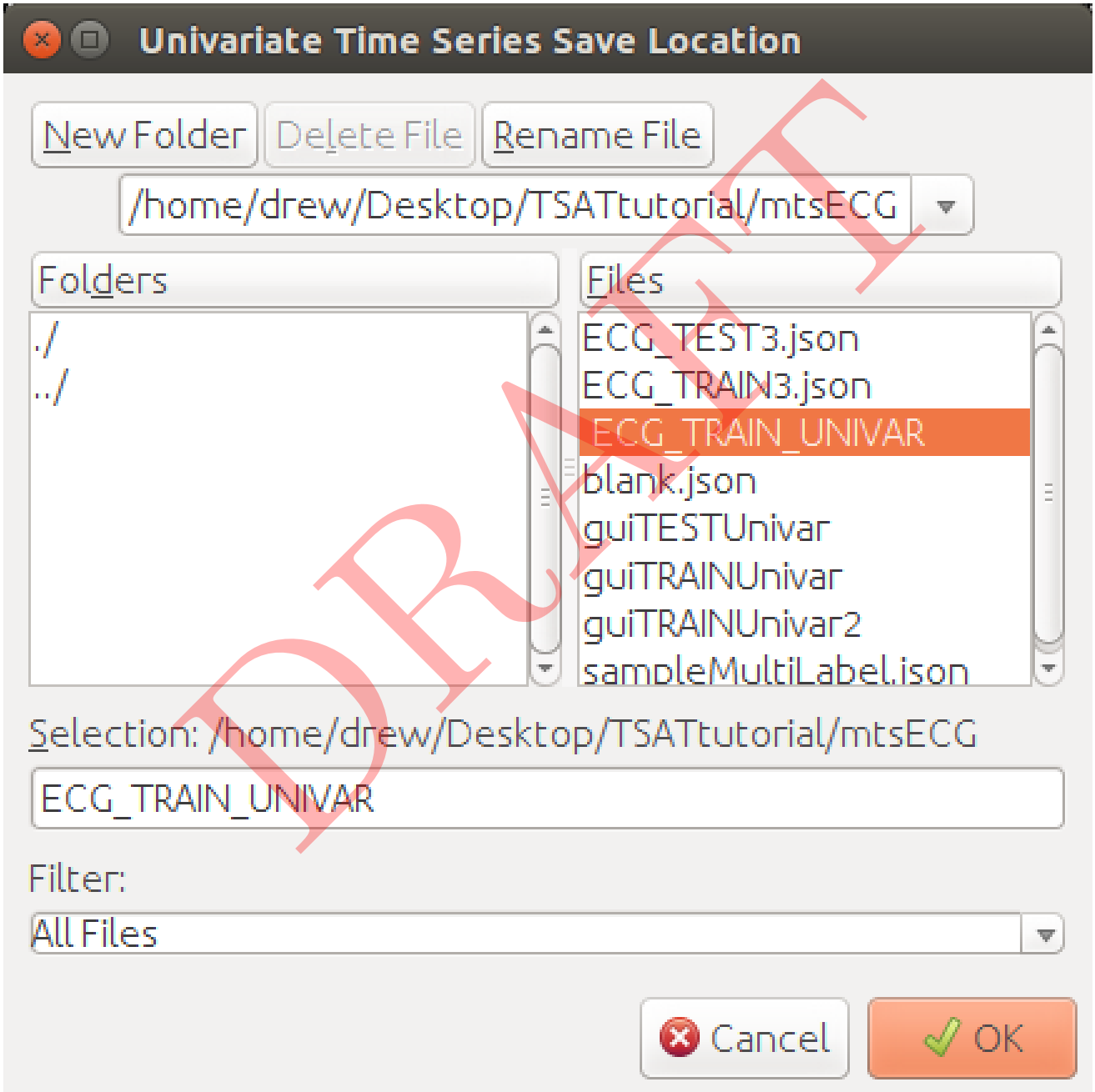

Figure 23: Location and filename for the univariate time series that TDA creates.

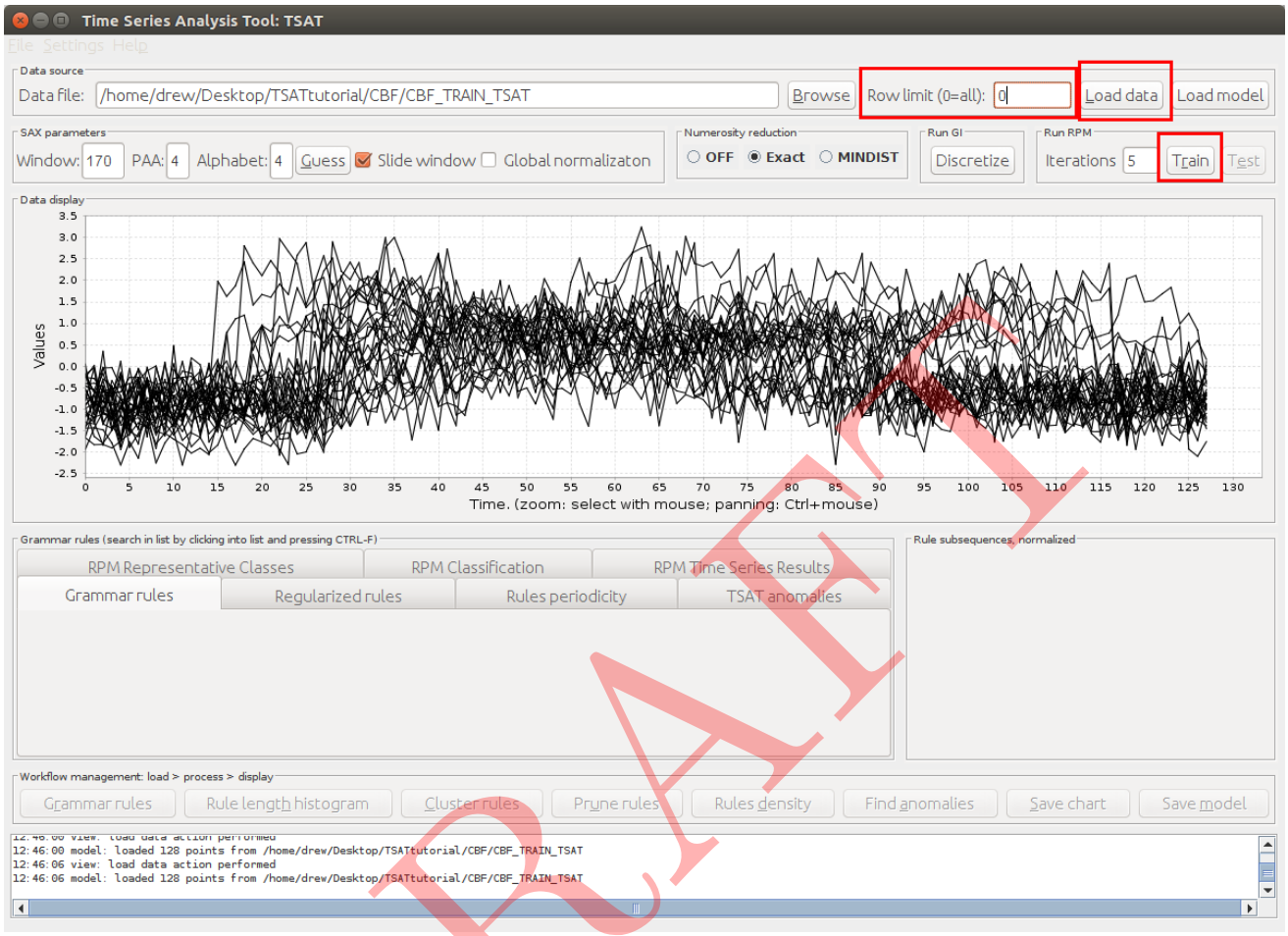

Figure 24: Loaded data

Hitting the train button will begin the training phase of RPM, this can take some time depending on the data and the number of iterations RPM will run. The text field labeled “Iterations” sets the maximum number of iterations RPM will go, this prevents RPM from running for to long trying to refine the model. Once the training is complete the tab “RPM Representative Classes” will become populated with patterns RPM thinks be represent the labels given. The fields “Window”, “PAA”, and “Alphabet” will also be populated with the values RPM believes are the best fit for the data to aid in further analysis.

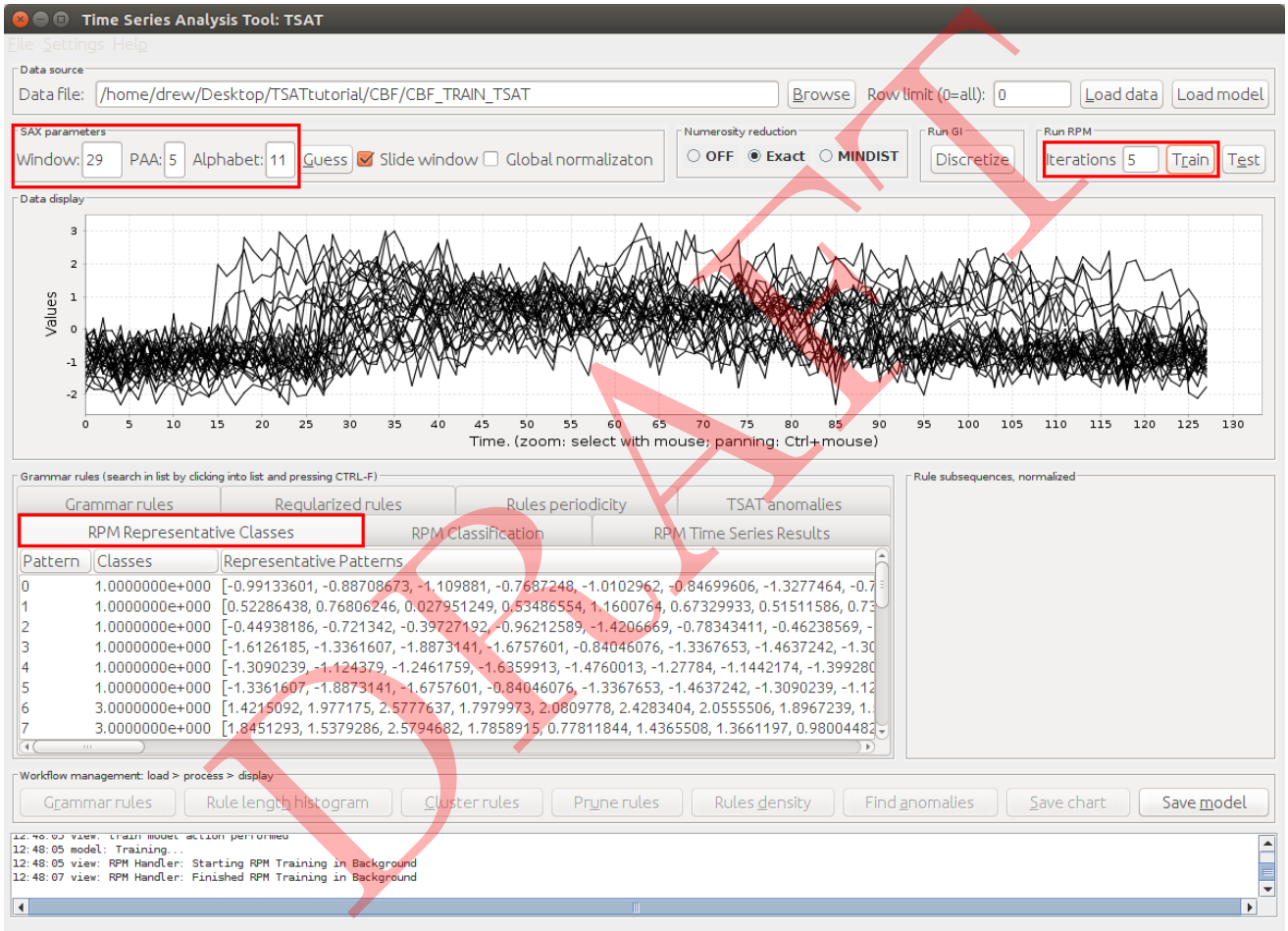

Figure 25: Representative Classes after Training

Selecting the patterns will display their graph on the right hand side of the window, multiple patterns can be selected.

The time series data that is displayed in the "Rule Subsequence" area corresponds to the selected patterns. These subsequences can be zoomed in to by left clicking and dragging the mouse over the area you wish to zoom into. In order to zoom back out, right click on the graph area, hover over "Auto Range" in the menu, and then click "Both Axes." Other features available after right clicking are to save the plot, print, copy, zoom in, and zoom out, and auto range. The purpose of this display is to allow the user to gain an intuition of what the time series data looks like.

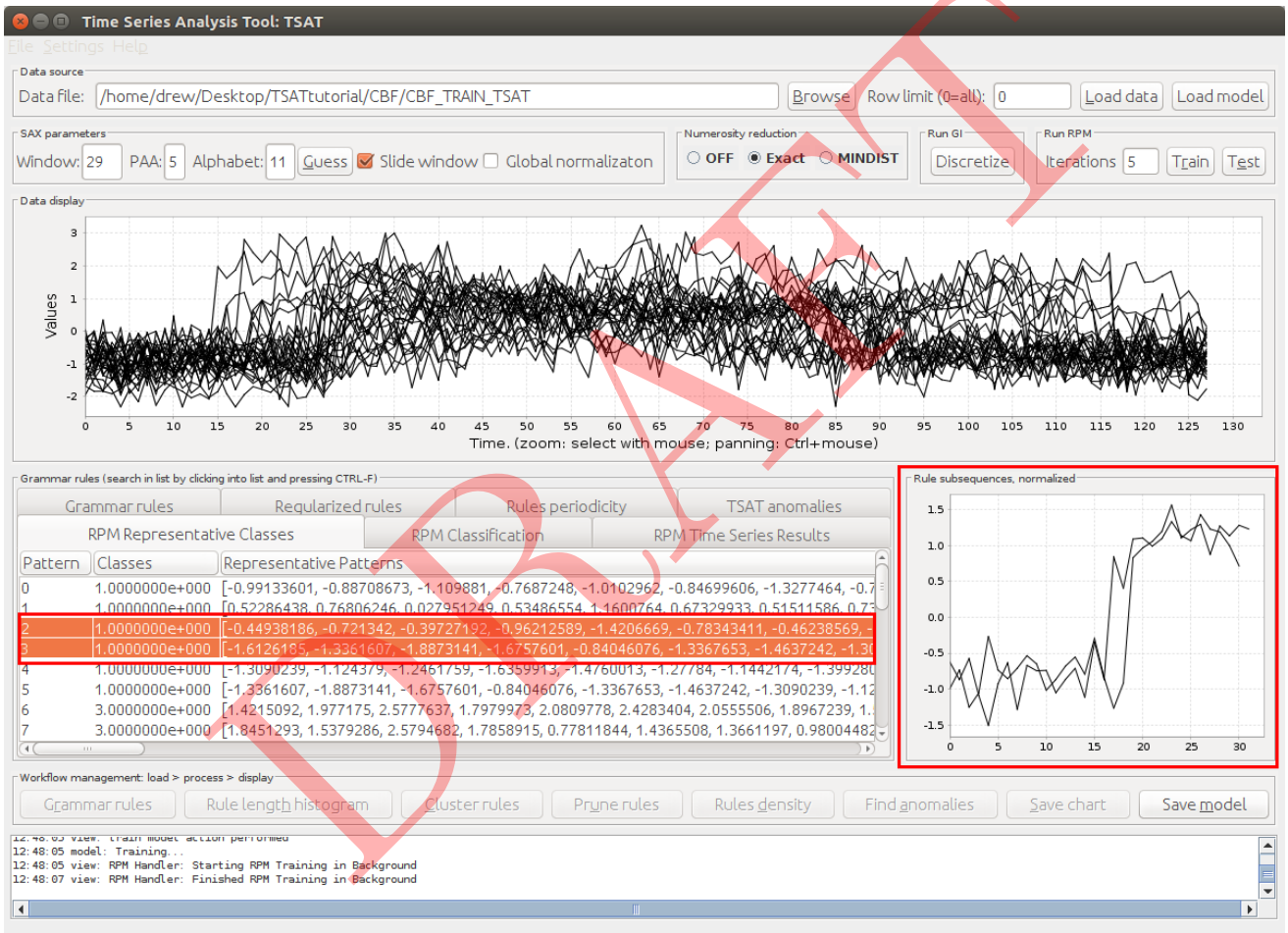

Figure 26: Representative pattern preview

5.3 Testing

Once the model has be trained it should be tested for accuracy, this will use a smaller dataset in the RPM compatible format to measure how well the model does.

Step 1 Click the "Test" button and a file browser prompt will appear, depending on how large the dataset is this may take a moment.

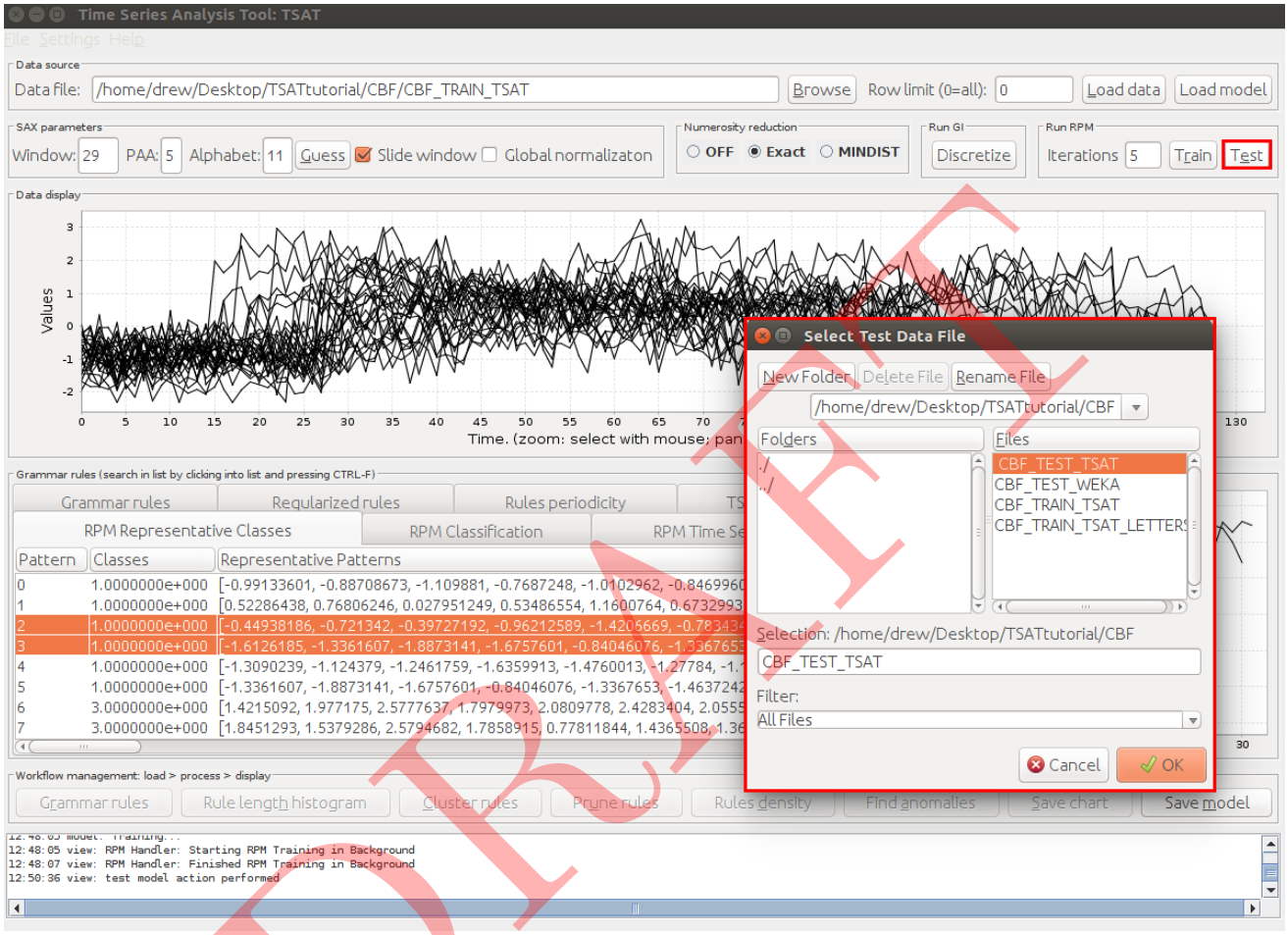

Figure 27: Testing the RPM model

Step 1 Multiattribute If the testing data is a multiattribute time series (with "json" appearing in the filename), then when the button labeled "Test" is clicked the Topological Data Analysis dialog will appear. This is illustrated in figure 28.

Once the values are set in the dialog and "Ok" is clicked, a new file dialog box will appear. This box requests the desired location and filename for the univariate time series data that will be generated from the multivariate time series data. This is shown in figure 29.

Once the file has been selected TDA will be run and the univariate time series will be created and the new univariate time series will be loaded into TSAT and displayed. Now that the multivariate time series has been converted to a univariate time series **TSAT** will proceed to perform the testing procedure.

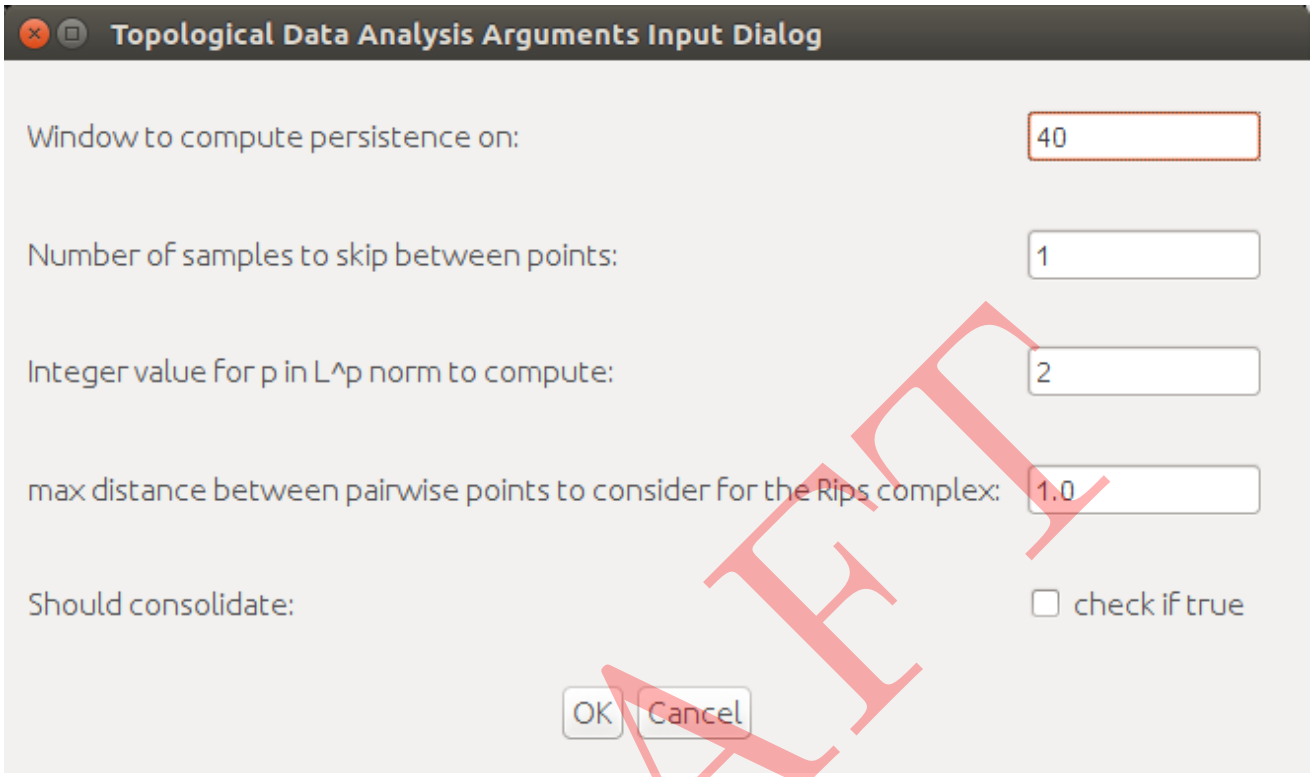

Figure 28: Topological Data Analysis dialog.

Once the testing is complete the tab labeled "RPM Classification" will be populated. This provides statistics on the effectiveness of the model by reporting the number of samples that were incorrectly labeled by the model.

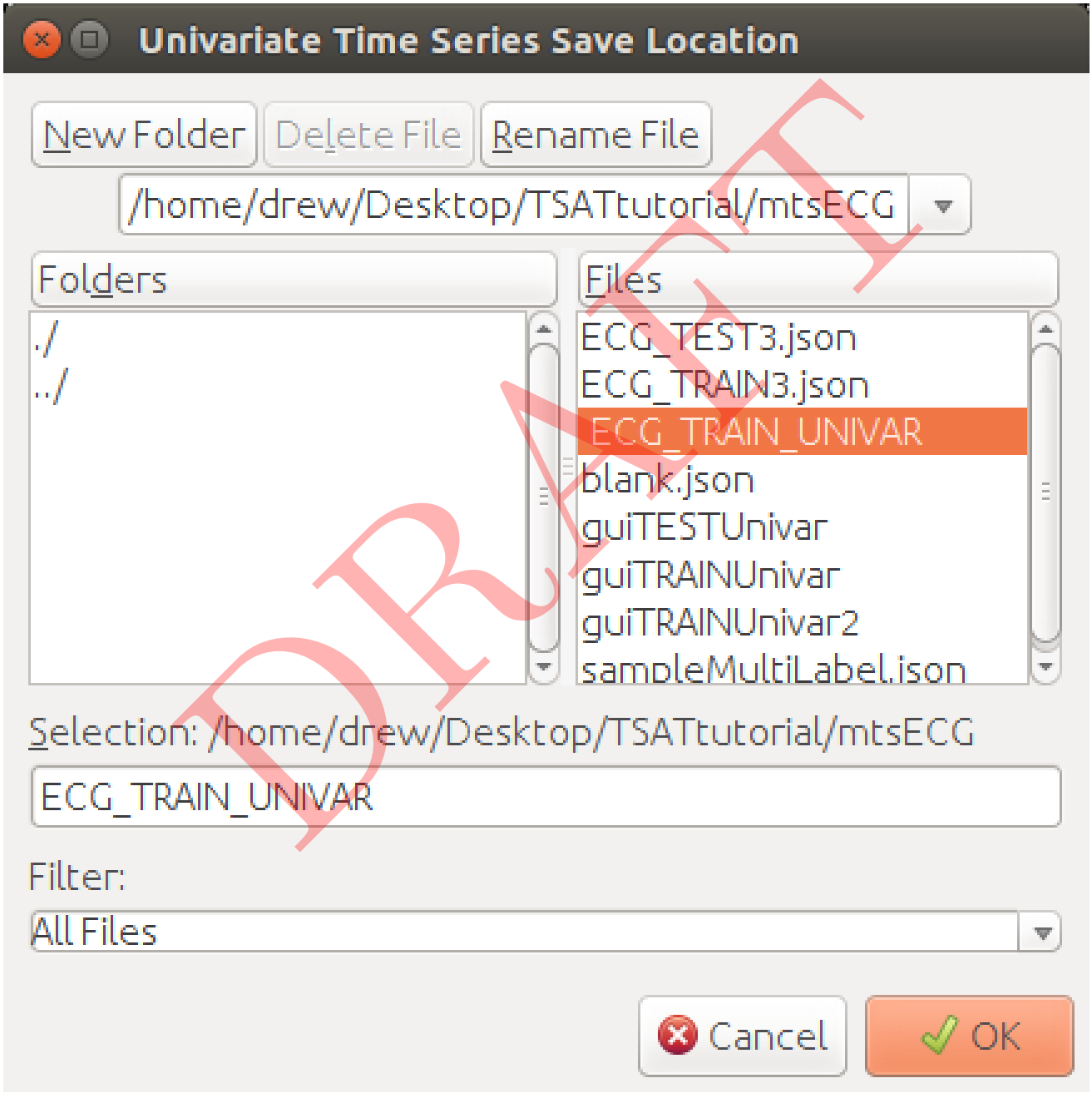

Figure 29: Location and filename for the univariate time series that TDA creates.

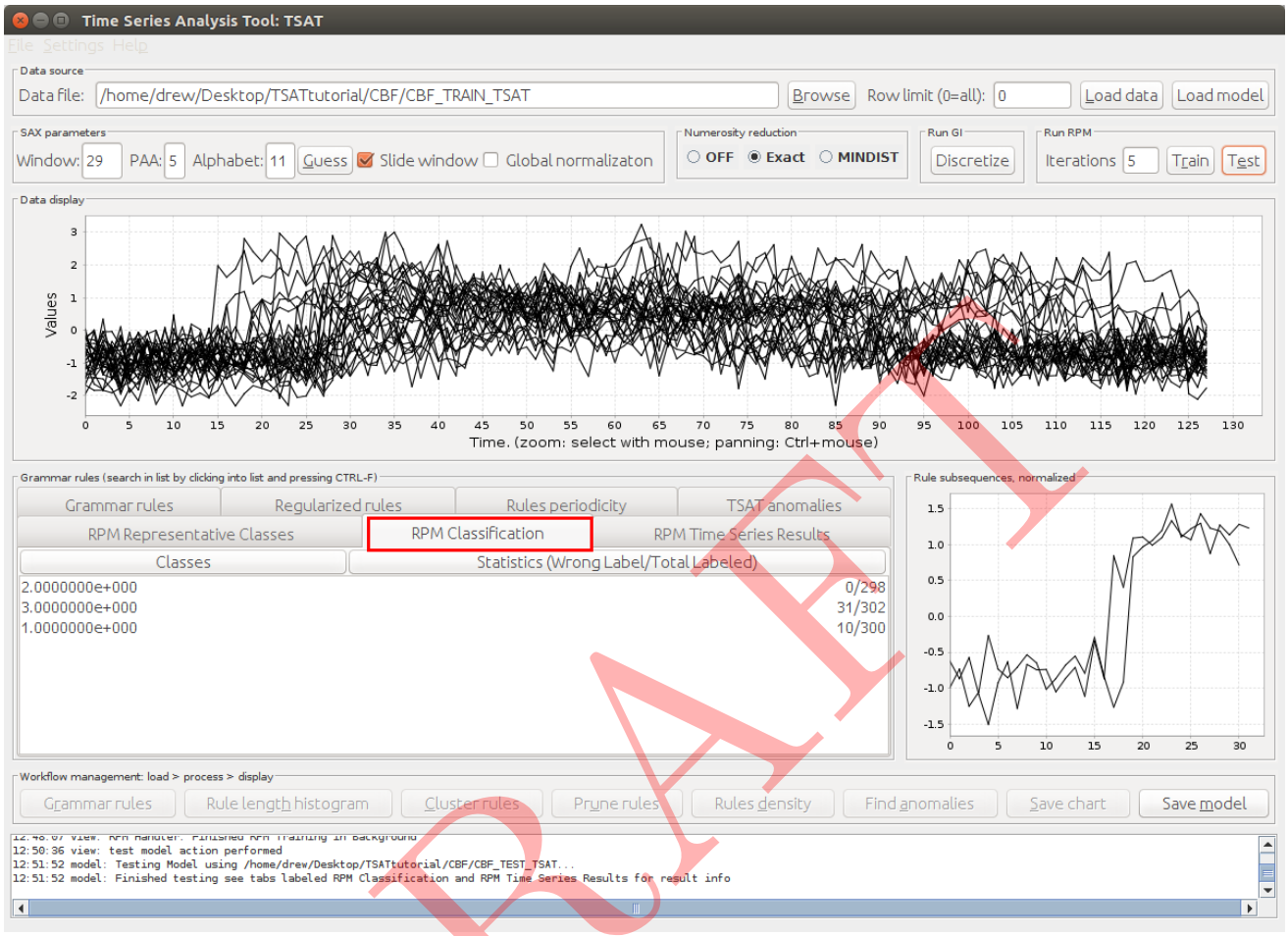

Figure 30: The results from the testing

Additionally, under the "RPM Time Series Results" tab shows the list of time series that were predicted incorrectly along with the the actual class the time series belongs, the predicted class, time series ID, and the time series. Shown in Figure 31.

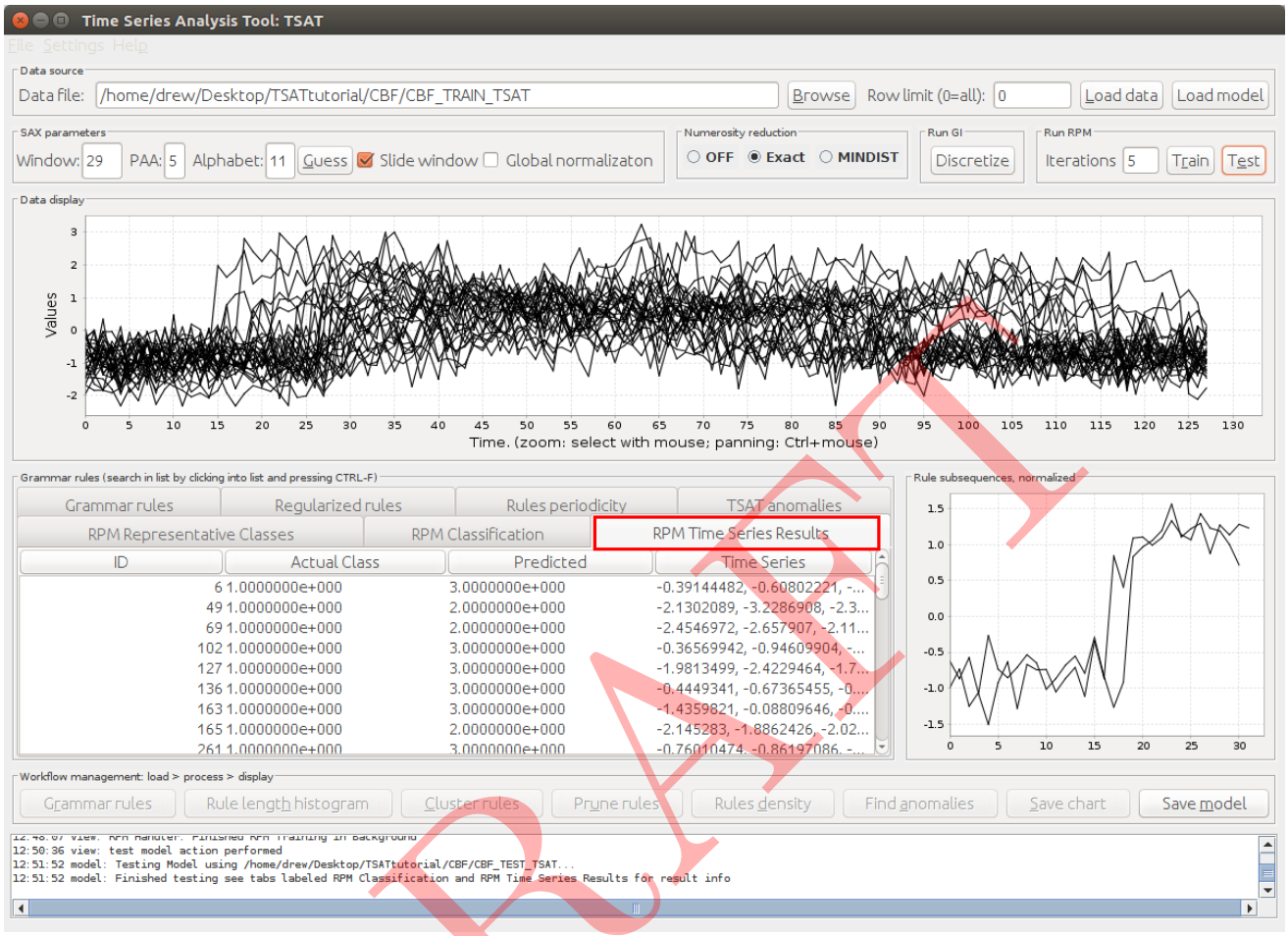

Figure 31: The results from the testing

5.4 RPM Statistics

Once testing is completed comprehensive statistics are provided under the "RPM Statistics" tab shown in Figure 32.

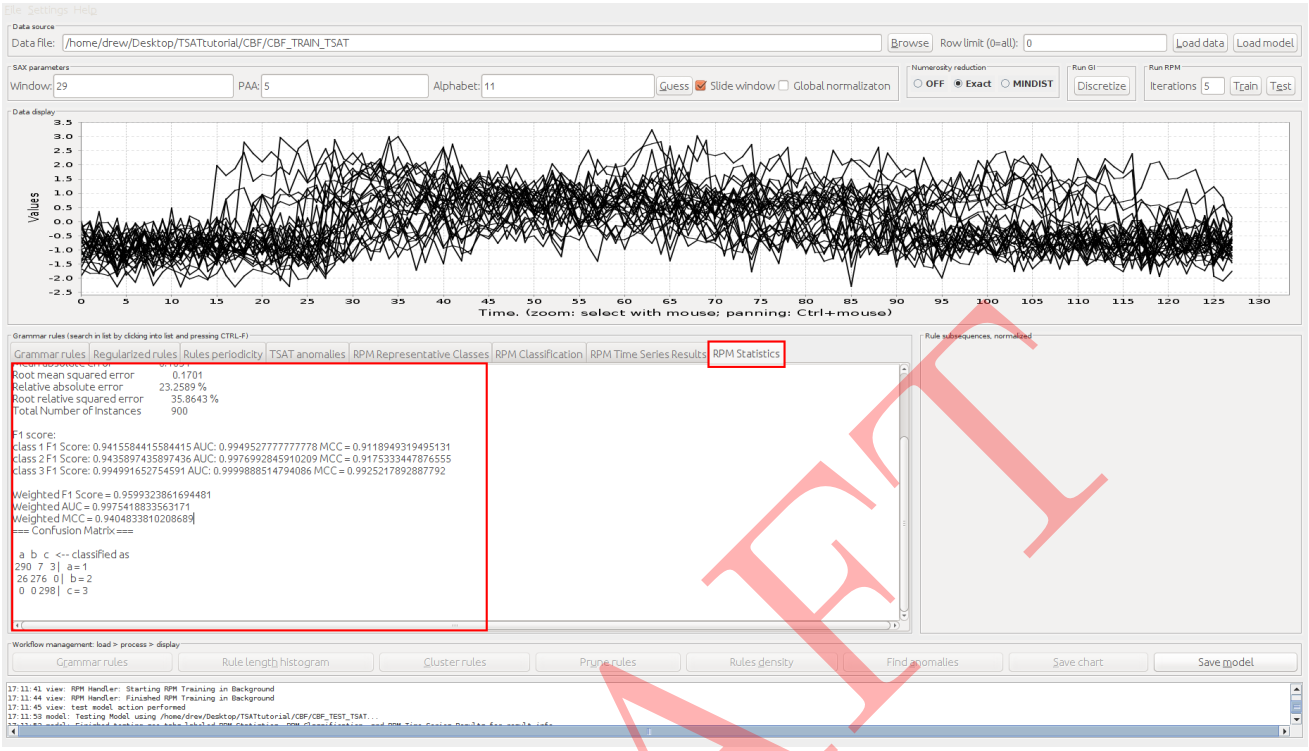

Figure 32: Statistics are shown under the "RPM Statistics" tab in the text area.

5.5 Testing Unlabeled Data

Using the same method for loading the test data when the data is labeled we can see the results for unlabeled data. Here the test data labels are all question marks so the results will consist of the probability that the test example is in each of the different training classes and the predicted label. For example, in figure 33 the solid box has the label probability for each class and dashed box has the predicted class label.

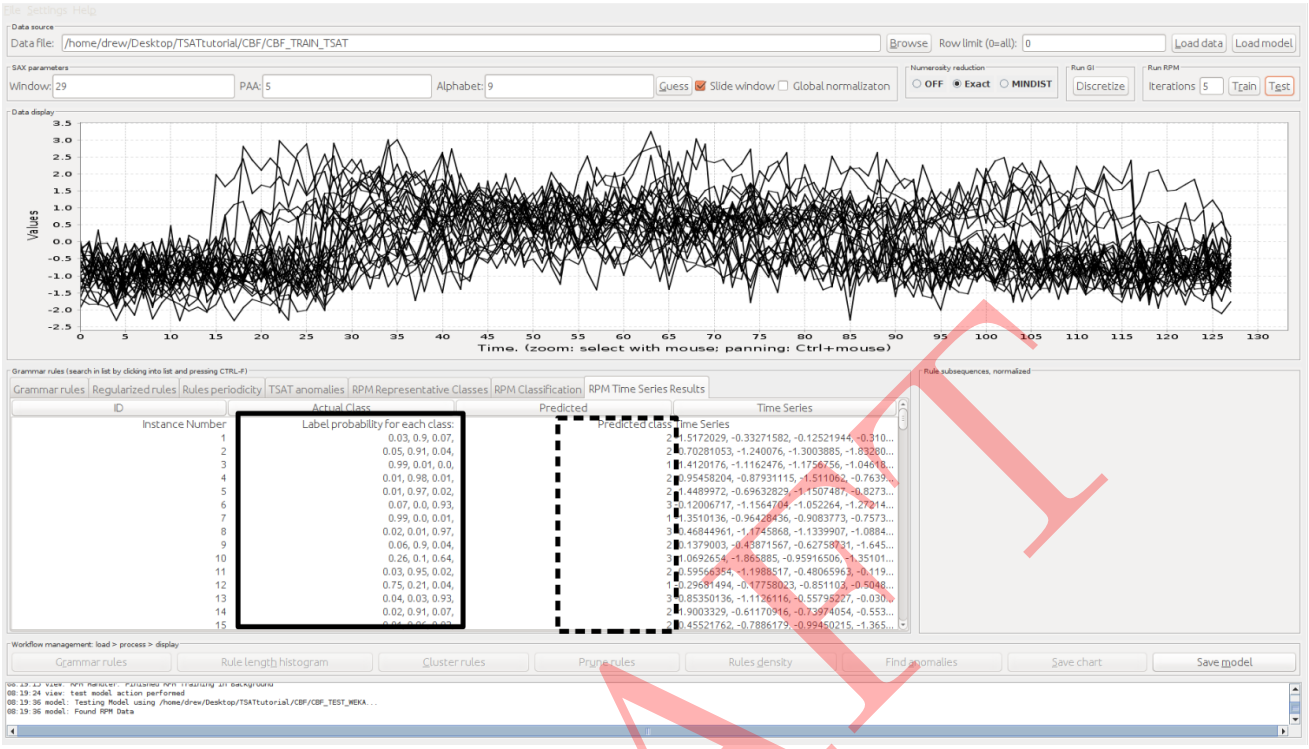

Figure 33: Solid box highlights the probability that the time series was in each of the different class labels and the dashed box highlights the predicted label. For example, instance 1 has label probability values of 0.03, 0.9, and 0.07. This means that the classifier claims that this instance is label 1 with probability 0.03, label 2 with probability of 0.9, and label 3 with probability 0.07. The predicted class is 2. This can be interpreted as instance 1 being given label 2 since it has a probability of 0.9 (the highest probability) of being the correct label (according to the "Label probability for each class" column).

5.6 Saving a Trained RPM Model

Creating a model can take some time and there for being able to save the model for later uses is a useful feature. Saving the RPM model will generate a file that can be loaded in later for further testing. One thing to note is that the saved model does not contain the training data however the training data is still needed when doing testing there for a copy of the training data must be retained.

Step 1 Once a model has been trained, click the save model button, as in figure 34, a file browser prompt will appear.

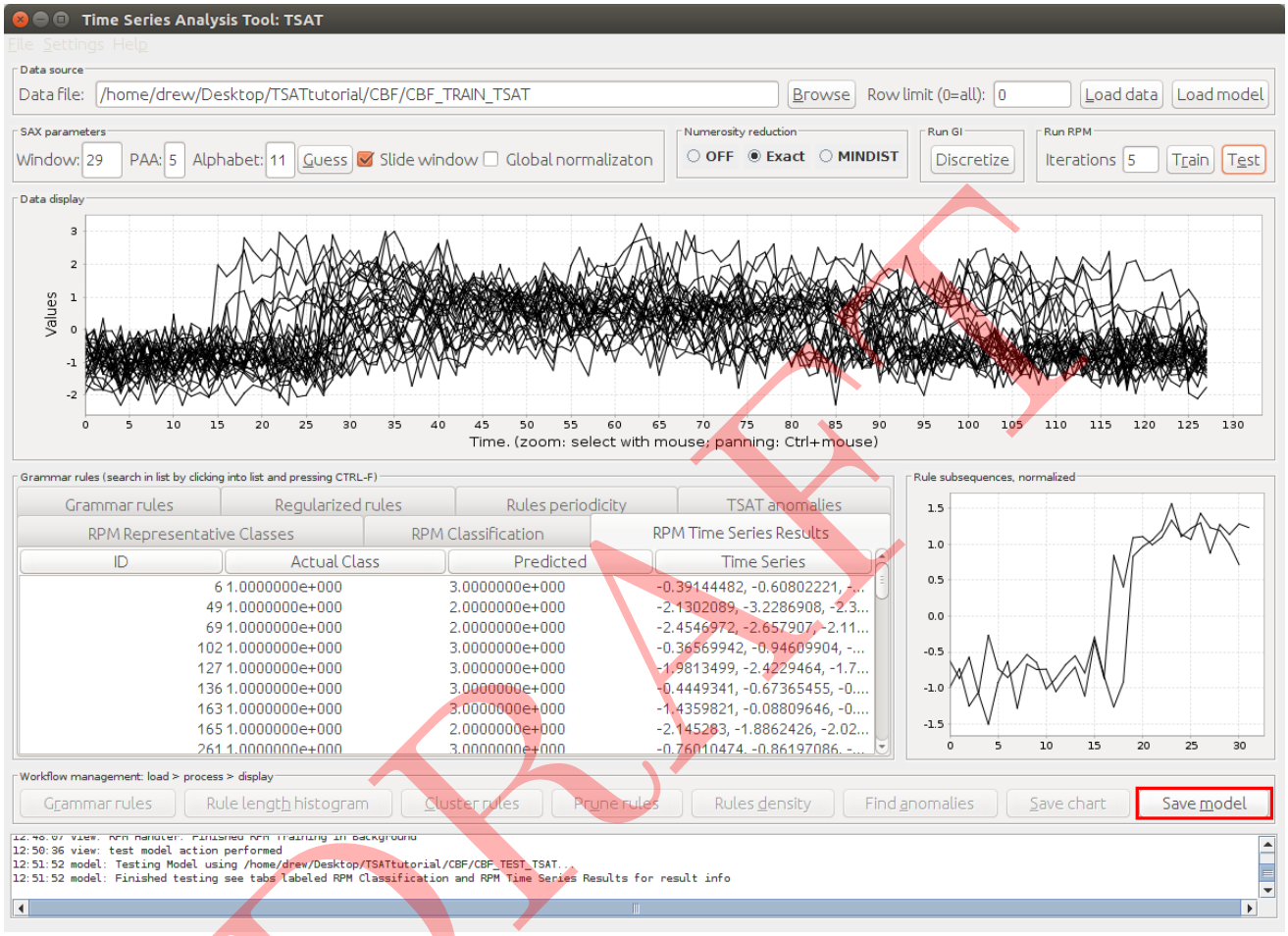

Figure 34: Saving the RPM model

Step 2 With the file browser prompt select a location to save the model and give it a name (in figure 35 the model name is CBF_Model), then click the "OK" button to save the model. Here, CBF is the name of the dataset that was trained. The files that are displayed in this folder are additional training and test time series files. It is recommended to keep the model file in the same directory as the train and test datasets.

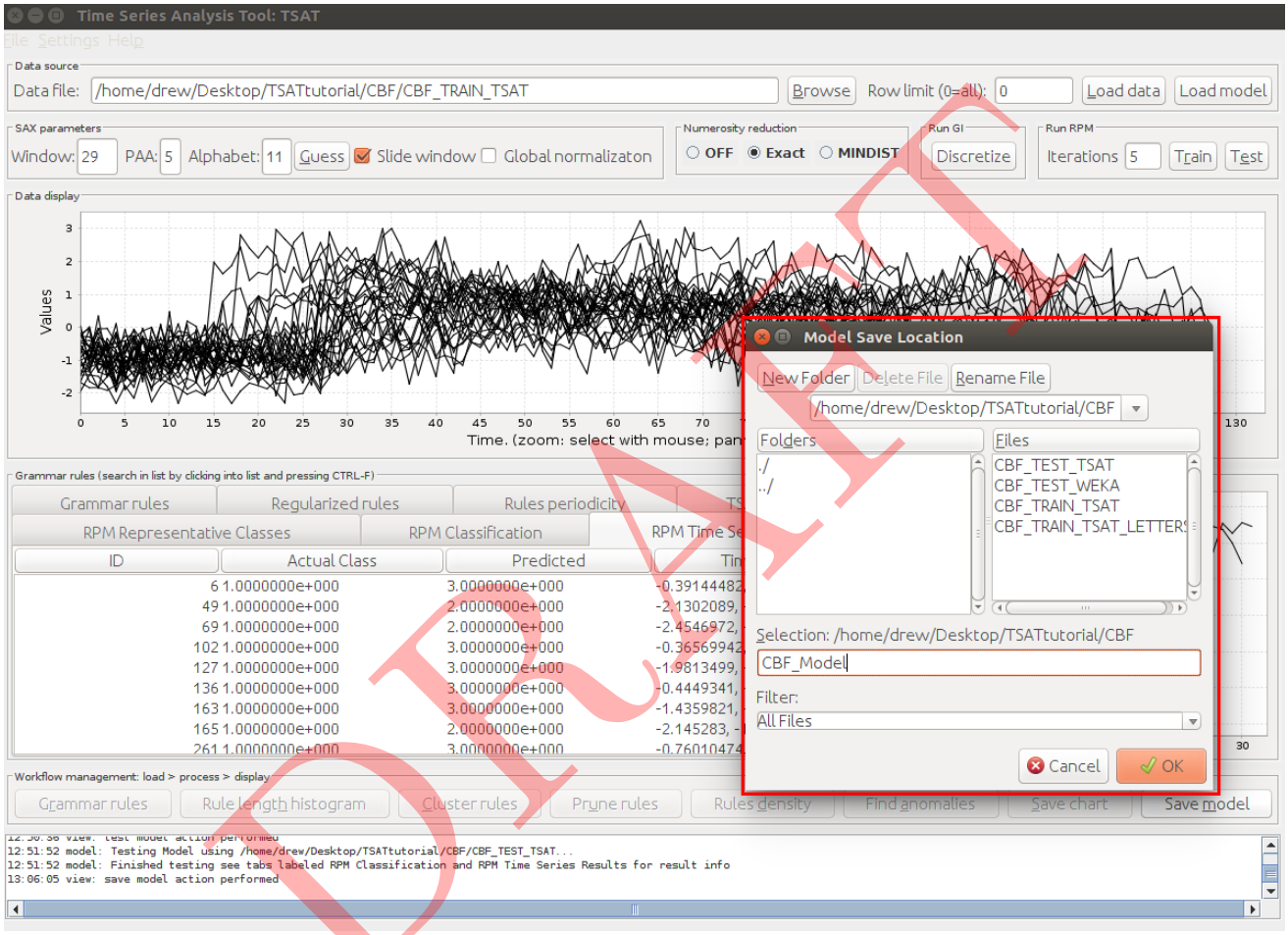

Figure 35: Saving the RPM model to file

5.7 Loading an RPM Model

When a model has already been saved, simply loading the will allow for further testing. When loading a model the software will look for the original training data from where it was when it was originally trained. If the data is not there then the software will ask for the location of the data.

Step 1 First click on the "Browse" button under the "Data Sources" section of the window, as seen in figure 36.

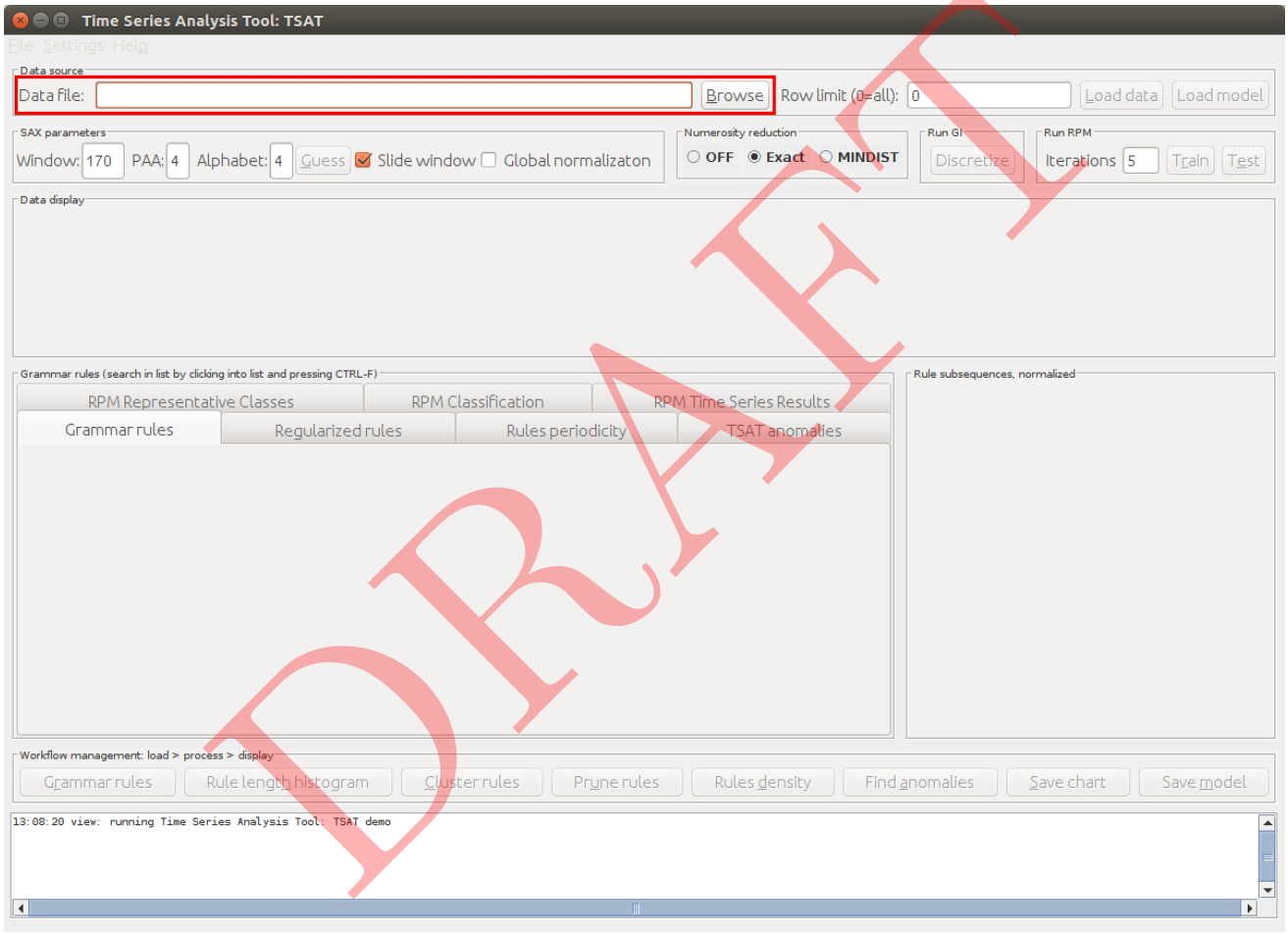

Figure 36: Loading a model

Step 2 This should being up the file browser prompt in figure 37. Using this prompt select the previously saved model.

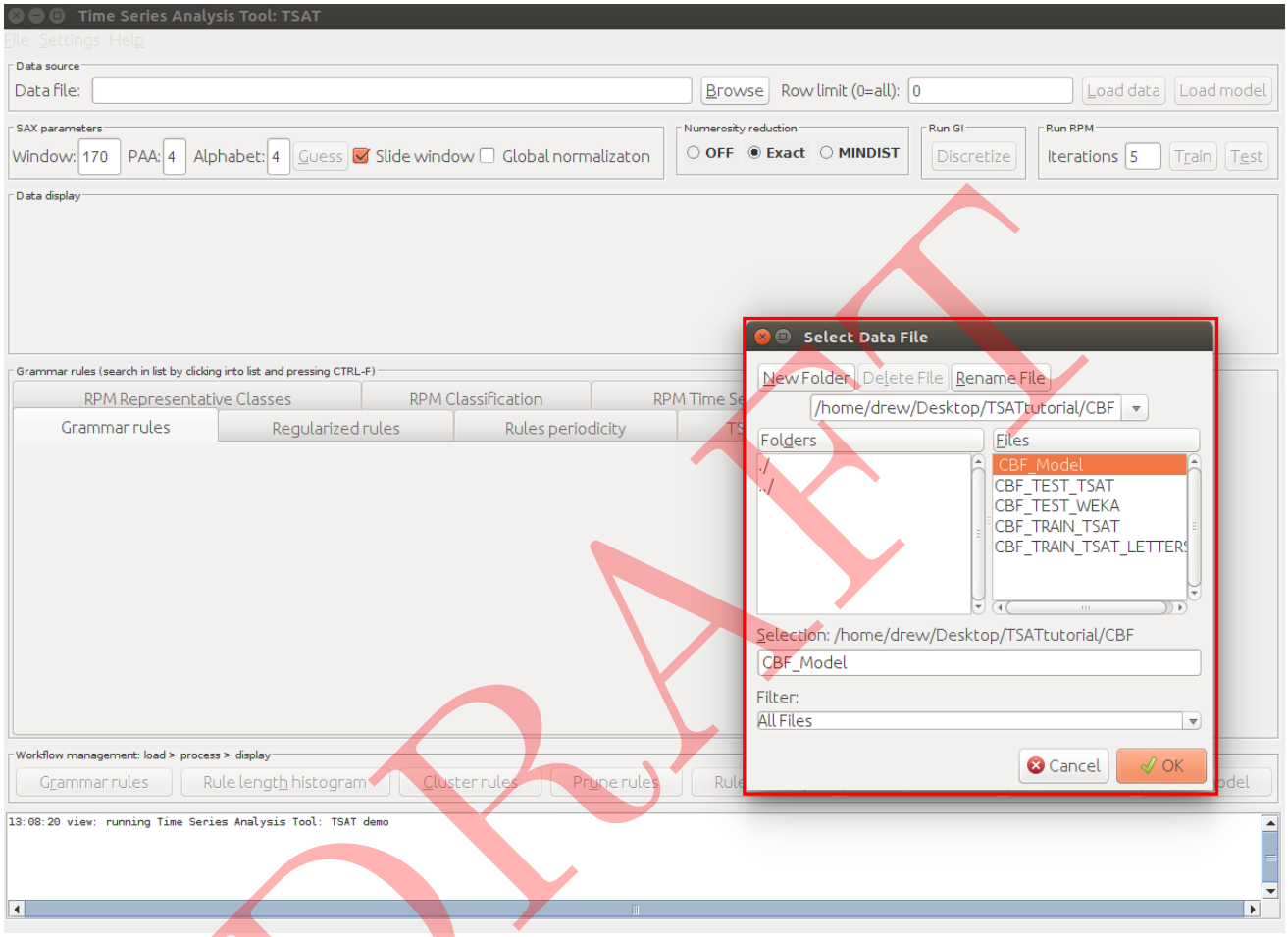

Figure 37: Open the file browser prompt

Step 3 Once the model has been selected, click the "Load Model" button and the model will be loaded into TSAT. If the data is not found during the loading step TSAT will ask for the location of the data using a file browser prompt, like in figure 39, simple provide the data and the model will finish loading.

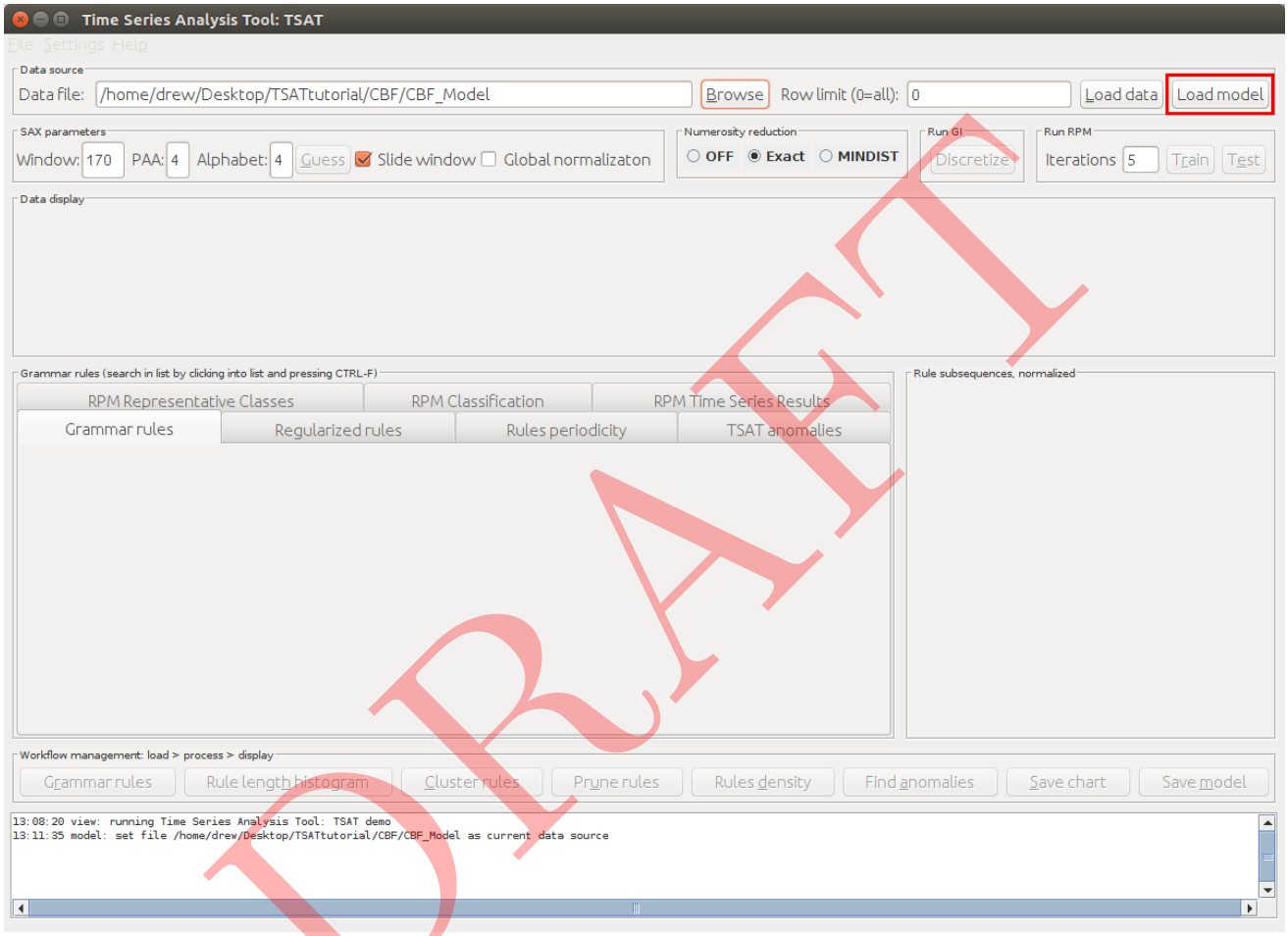

Figure 38: Model loaded

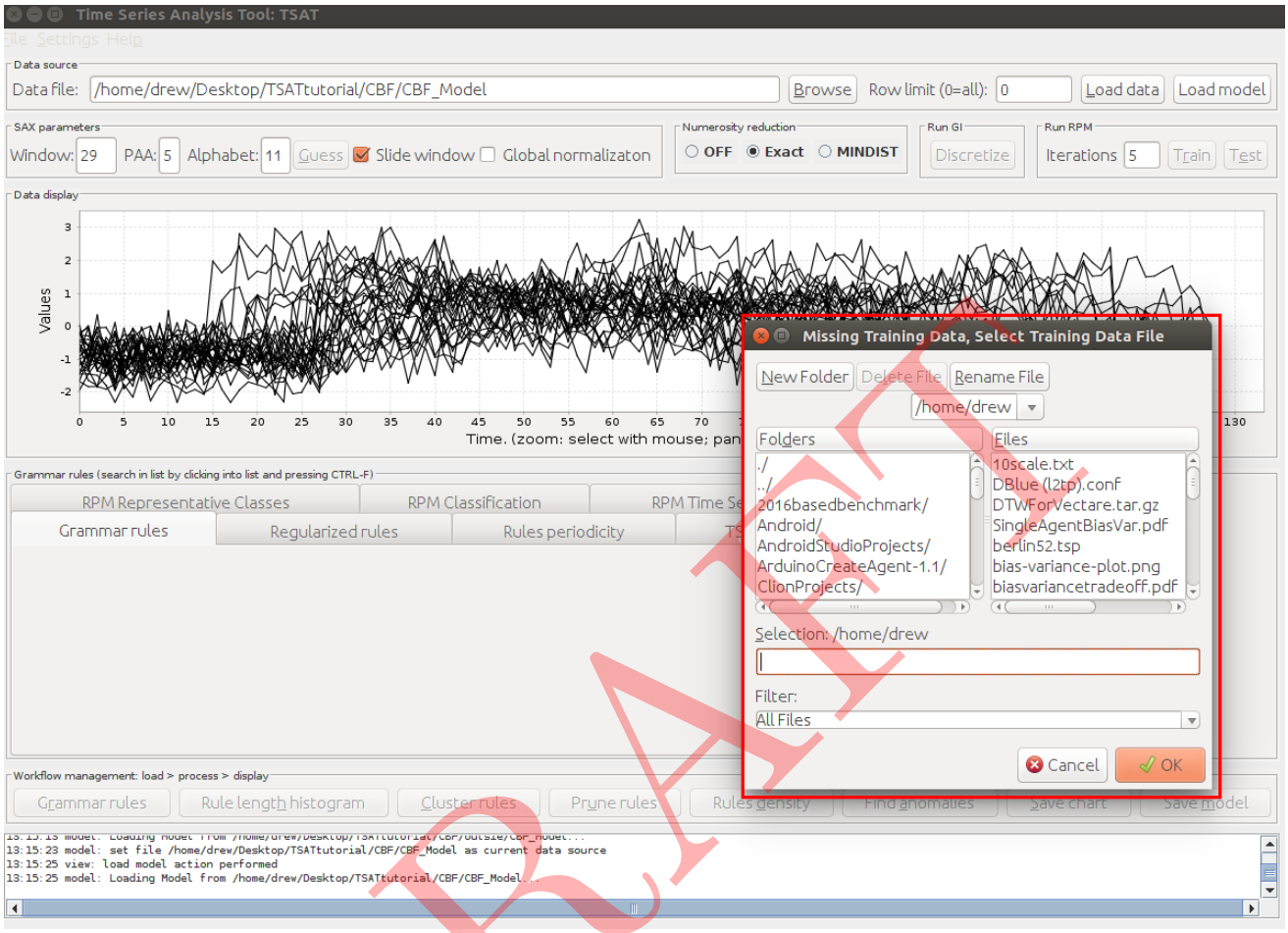

Figure 39: Missing training data file browser prompt

5.8 Settings

There are a few options that can be changed when using RPM in TSAT, some of them have already been mentioned and will be covered again.

5.8.1 Dynamic Time Warping

Dynamic Time Warping, or DTW, is a method of measuring distance between two time series, this means how similar or different they are to each other. By default RPM uses Euclidean distance which is a simple and fast measurement, however it does not do well when the similar patterns between time series occur at different positions. This is where DTW comes in, it can handle temporal shifts in patterns and, depending on the data, can vastly improve the accuracy of the model. There is a cost however, DTW is a much slower operation and is very expensive to run so it is left as an option for the user.

DTW also has another parameter called “Window” which can have dramatic effects on DTW both in how long it takes to run and its accuracy. The window size basically limits how far DTW will go to try to accurately try to match the two time series. A smaller window will stop DTW from trying to over match them and will take less time to compute. A larger window will take much longer to compute but can allow DTW to match patterns that are father apart. Choosing a good window size can be highly dependent on the data and what is being compared, and therefore some experimentation may be needed to find a good window size. However, there are a few good rules when choosing a window size. First,

a window size greater than 10 will usually give bad results so 10 is considered a good starting point. Also, for more common types of data, a 3-5 window size can be a much better option with significant speed ups. Note, DTW's window should not be confused with the Window size in the SAX parameters section of the main window, these are two different and distinct uses of the word window.

Step 1 To change between Euclidean distance and DTW first open the settings menu: "Settings"→"TSAT options" or press Ctrl+p. This will bring up the settings menu in figure 40.

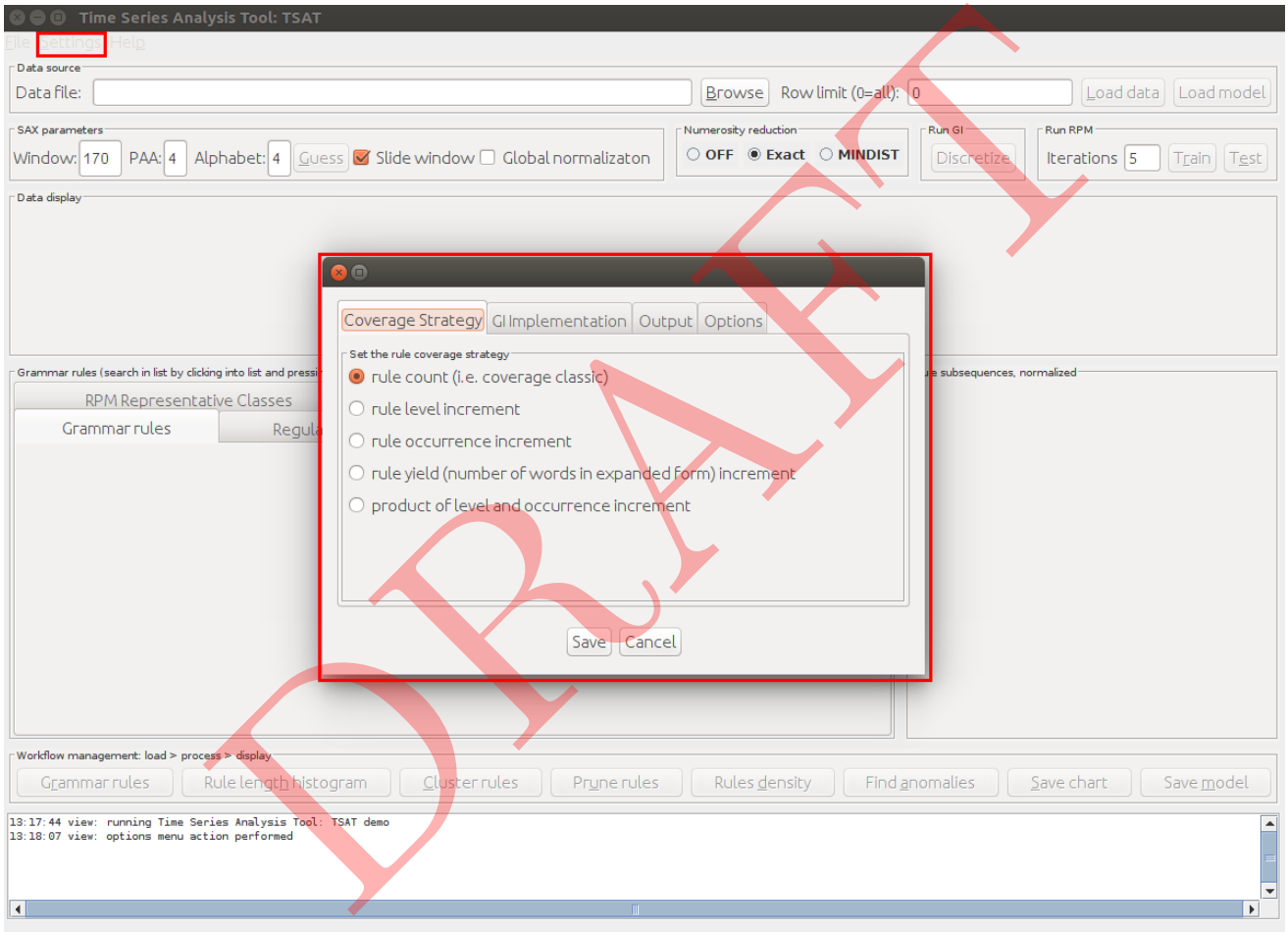

Figure 40: TSAT Settings Dialog

Step 2 Now click on the "Options" tab.

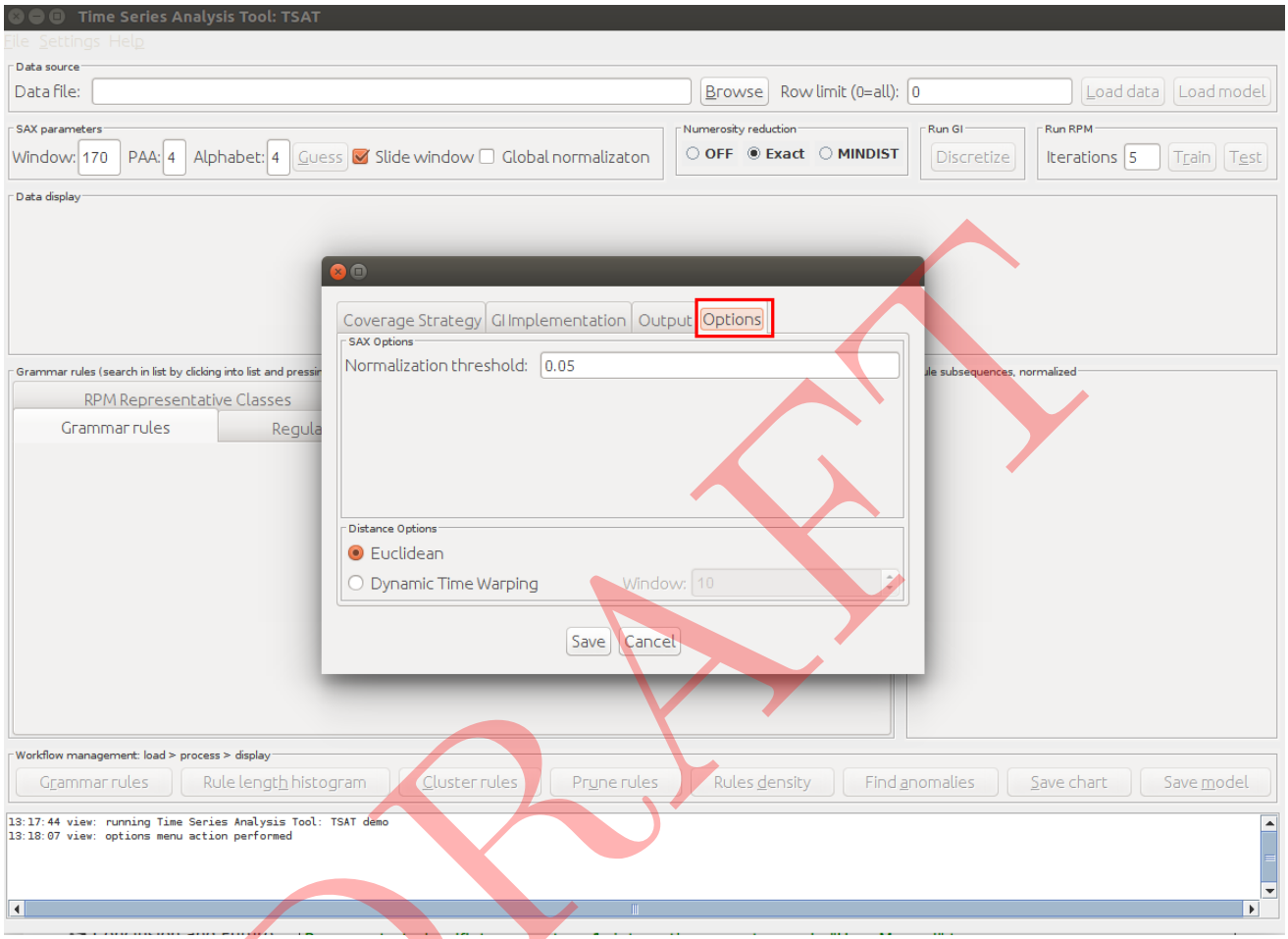

Figure 41: TSAT Settings Dialog Options

Step 3 Now select the "Dynamic Time Warping" option and the desired "Window" then click save.

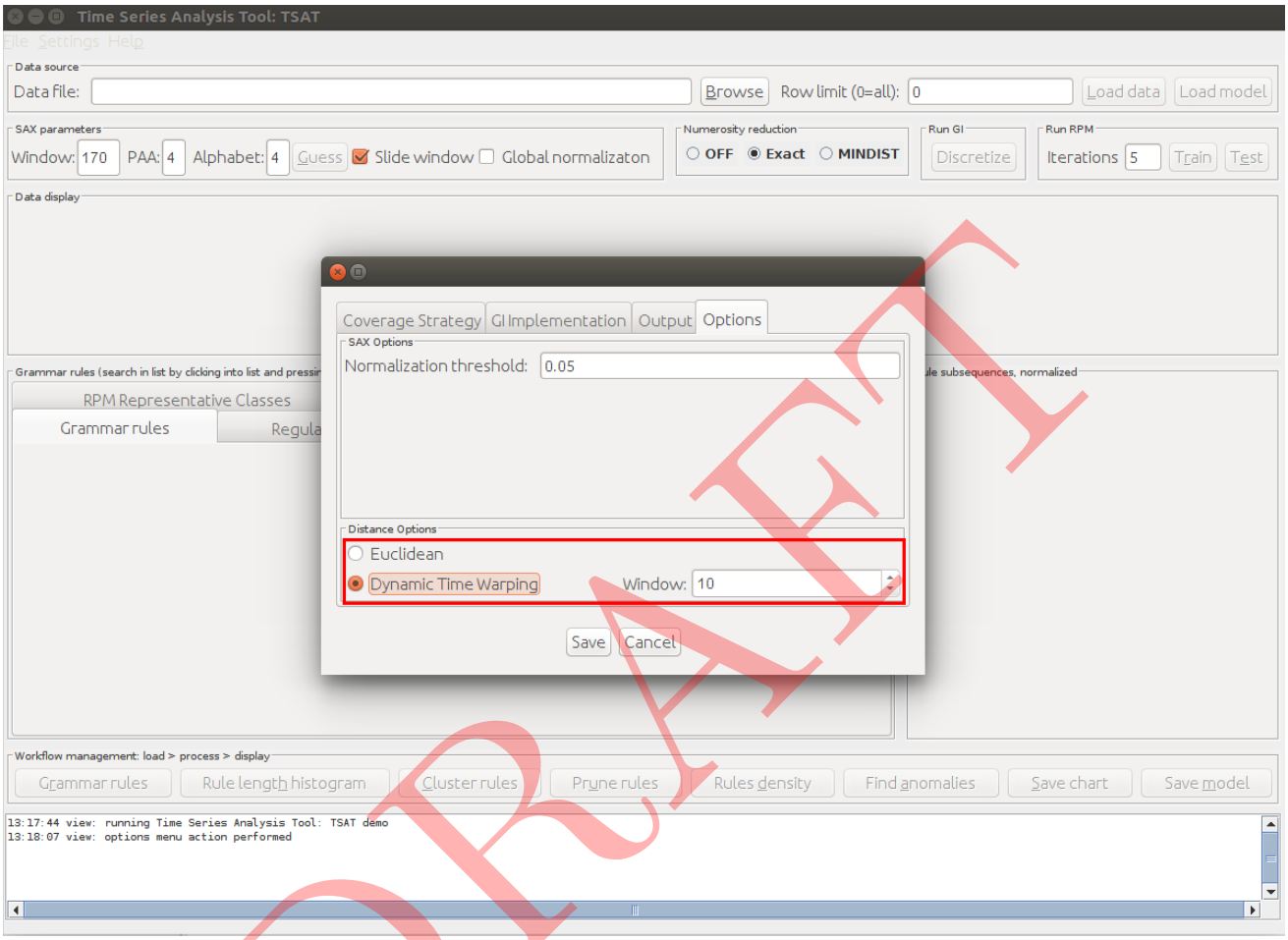

Figure 42: TSAT Settings Dialog Options DTW

5.8.2 Iterations

During the operation of RPM it goes though a step that gets repeated many times. This step only stops under two conditions, a minimum threshold is met or if the maximum number of iterations are reached. The iterations setting found under the "Run RPM" section of the main window in TSAT is how the user can control the maximum number of iterations, figure 43. The number of iterations can have an effect on how accurate the model can get, however the more iterations RPM runs through the longer it will take to complete. This becomes a balance between the quality of the model and the how long the training phase will take. It should also should be noted that RPM can stop before the maximum number of iterations is met if the model has reached an ideal state. However, this does not mean that all models will or even can reach an ideal state before the maximum number of iterations is reached, indeed some data sets may never return a model that meets the requirements. As RPM runs through the iterations the model should get better but the amount it gets better by can be come increasingly insignificant and therefore adding another 10 iterations may not add any significant results to the model. The only way to know if adding more iterations will improve the model is by experimentation which would involve training multiple times, increasing the maximum number of iterations every run until the testing results return no significant improvements.

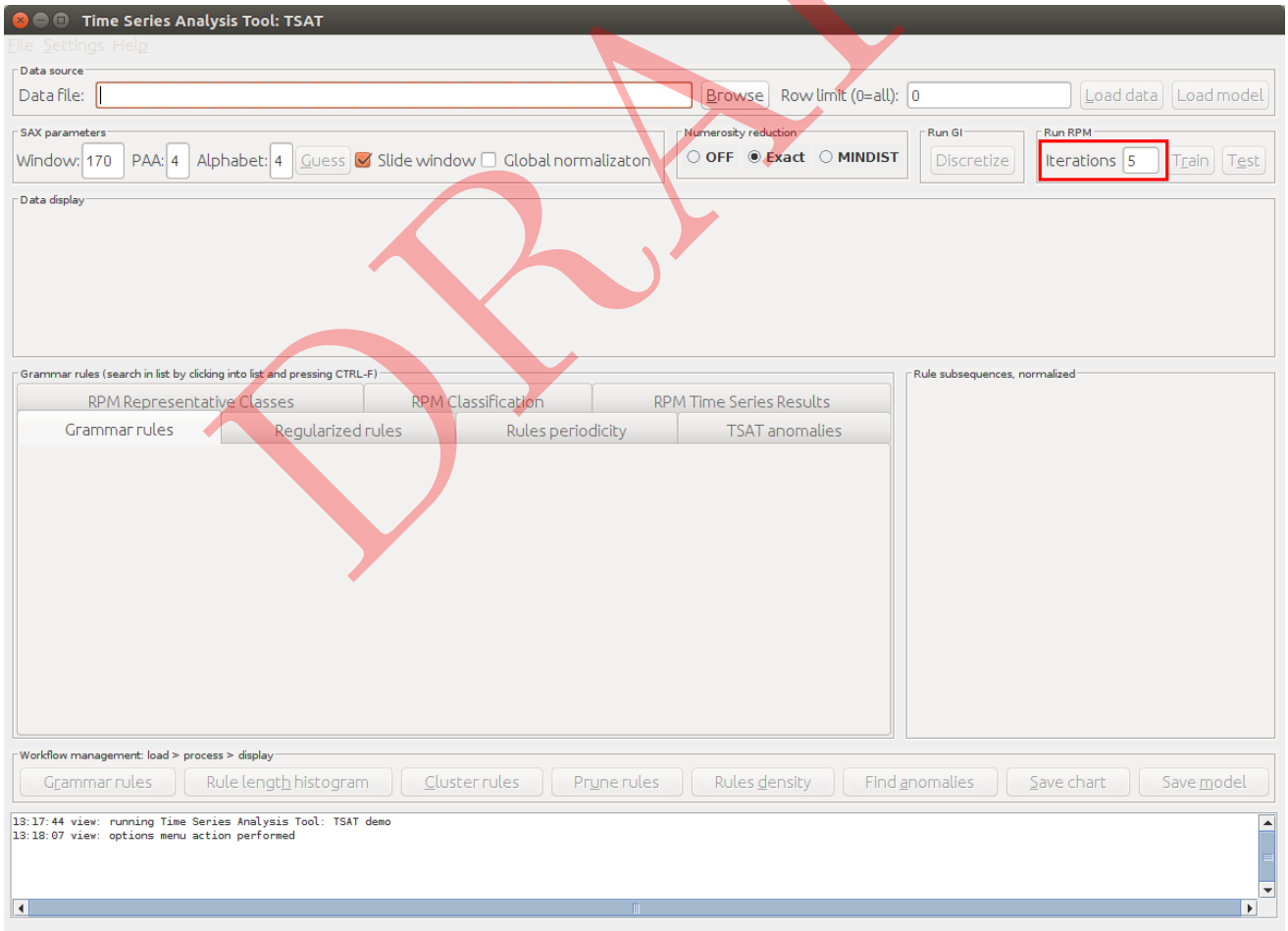

Figure 43: TSAT RPM Iteration Setting

5.9 Python Interface

There are three functions: RPMTrainTest, RPMTrain, and RPMTest and they can be called external from tsail.py as:

```
tsail.RPMTrainTest(pathToTraining, pathToTest, outputFile, num_iters)
tsail.RPMTrain(pathToTraining, outputFile, num_iters)
tsail.RPMTest(pathToTest, modelFile, num_iters)
```

With parameters:

pathToTraining The path to the file containing the training time series data.

pathToTest The path to the file containing the testing time series data.

outputFile The name of the file that will be used for writing the returned json object and the serialized model file which can be used as input to RPMTest.

num_iters The number of RPM iterations to run.

modelFile The filename of the serialized model file. Either saved from the GUI or generated by RPMTrainTest or RPMTrain.

For RPMTrainTest both training and testing will be output whereas in RPMTrain and RPMTest they each do exactly either training or testing.

The training output is an array of:

https://github.com/dwicke/TSAT/blob/72498ab66795e221eb6e26fbc65b0f156169ca66/src/main/java/edu/gmu/grammar/patterns/TSPattern.java

TSPattern has the following properties that are accessible via the returned json object:

```
private int frequency; // The number of times this pattern is used
private double[] patternTS; // the time series pattern
private double error = 0; // the pattern's error rate.
private String label; // The class label for the pattern
private int fromTS; // the index for the time series.
private int startP; // the patterns start position in the concatenated data.
```

The test output is a 2D array of strings for each instance we have the value corresponding to `[['inst#', 'actual class', 'predicted class', 'timeSeries']]`

The test output when the data input is unlabeled is a 2D array of strings where for each instance we have:

`[['inst#', 'comma separated list of the probabilities of being in the particular class', 'predicted class', 'timeSeries']]`

When running RPMTrainTest you will generate three files as output instead of 1.

`<outputfileName>.train <outputfileName>.test <outputfileName>`

Files with the .train and .test extensions are the json of the python dictionaries as discussed in previous sections. The last file can be imported into TSAT GUI as it is the same as the saved model in **TSAT**. Example usage of all three functions are contained within tsail.py.

6 Python Multiattribute Time Series

If your time series is multiattribute then it must be converted to a univariate time series before being used with any of the motif, anomaly detection, or classification python **TSAT** interfaces. This conversion is

implemented as a command line interface. An example call:

```
python tdaInterface.py /home/drew/Desktop/TSATtutorial/mtsECG/ECG_TRAIN3.json
ecgUnivar.train 1 50 1 2 1.0 0
```

The help documentation is displayed when used with the -h flag. The help is shown below.

```
usage: tdaInterface.py [-h] MVData UVData numL window dt p maxrad shouldConsolidate

Topological Data Analysis tool to convert multivariate time series to
univariate time series.

positional arguments:
MVData              The json file containing the multivariate time series
dataset
UVData              The name for the univariate time series data file created
by this tool.
numL                The number of examples to read from the multivariate time
series file
window              The width of the window to compute persistence on, will
take up to this number of samples
dt                  number of samples to skip between points.
p                   integer (type of L^p norm to compute)
maxrad              max distance between pairwise points to consider for the
Rips complex
shouldConsolidate   If 1 will merge values in the time series that are
less than 10 time steps apart.

optional arguments:
-h, --help  show this help message and exit
```

7 Notes

When training there must always be more than one example from each class label and there must be more than one label.

Installation This tutorial assumes that you are running Ubuntu 16.04 with Java 1.8 or greater installed.

```
git clone https://github.com/dwicke/TSAT.git
cd TSAT
mvn package -Psingle
```

This will create tsat-1.0-SNAPSHOT-jar-with-dependencies.jar in the target directory. You can execute the jar and run the GUI by double clicking on it after changing its permissions:

```
chmod +x tsat-1.0-SNAPSHOT-jar-with-dependencies.jar
```

To run the GUI from a shell you can do:

```
$ java -Xmx2g -jar target/tsat-1.0-SNAPSHOT-jar-with-dependencies.jar
```

The -Xmx2g allocates max of 2Gb of memory for the software.

7.1 Javadoc

The javadoc can be created by running
`mvn javadoc:javadoc`
in the root directory of **TSAT**. This will generate javadoc in the directory
`target/site/apidocs/` Then loading index.html in a browser will let you explore the generated javadoc.

Topological Data Analysis Installation In order to install the Topological Data Analysis tool we assume that an Anaconda environment is created with the name py36 with python 3.6 set as the python interpreter. Now within the py36 conda enviroment do the following:

```
#dependencies (Linux assumed)
#install in this order:

conda install numpy, scipy, dill, matplotlib, pandas

# sklearn from pip
pip install sklearn

# tqdm from pip (progress meter widget)
pip install tqdm

#libboost-dev
sudo apt-get install libboost-dev

#dionysus2 from pip
pip install dionysus
```

There should be a .tsat file in your home directory that has set the correct folder locations in order to call the tdaInterface.py. An example is given below that is also included with the source code in the root directory of the project.

```
# The location of the  anaconda installation
# and the location of the tdaInterface should be set
source /home/$USER/anaconda2/etc/profile.d/conda.sh
export tdaInterface="/home/${USER}/src/TSAT/src/main/resources/tdaInterface.py"
function tda() {
        conda activate py36
        echo $CONDA_DEFAULT_ENV
        python -u $tdaInterface $1 $2 $3 $4 $5 $6 $7 $8
}

export -f tda
```

With the .tsat stored in the home directory which **TSAT** will look for in order to execute TDA everything should work.

References

[1] P. Senin, J. Lin, X. Wang, T. Oates, S. Gandhi, A. P. Boedihardjo, C. Chen, and S. Frankenstein, "Time series anomaly discovery with grammar-based compression," in *Proc. EDBT (Brussels, Belgium, March 2015)*, pp. 481–492, 2015.

[2] P. Senin, J. Lin, X. Wang, T. Oates, S. Gandhi, A. P. Boedihardjo, C. Chen, S. Frankenstein, and M. Lerner, "Grammarviz 2.0: a tool for grammar-based pattern discovery in time series," in *Machine Learning and Knowledge Discovery in Databases*, pp. 468–472, Springer, 2014.

[3] P. Senin, J. Lin, X. Wang, T. Oates, S. Gandhi, A. P. Boedihardjo, C. Chen, and S. Frankenstein, "Grammarviz 3.0: Interactive discovery of variable-length time series patterns," *ACM Trans. Knowl. Discov. Data*, vol. 12, pp. 10:1–10:28, Feb. 2018.

[4] C. G. Nevill-Manning and I. H. Witten, "Identifying hierarchical structure in sequences: A linear-time algorithm," *Journal of Artificial Intelligence Research*, vol. 7, pp. 67–82, 1997.

[5] N. J. Larsson and A. Moffat, "Off-line dictionary-based compression," *Proceedings of the IEEE*, vol. 88, no. 11, pp. 1722–1732, 2000.

[6] J. Lin, E. Keogh, S. Lonardi, and P. Patel, "Finding motifs in time series," in *Proc. of the 2nd Workshop on Temporal Data Mining*, pp. 53–68, 2002.

[7] J. Lin, E. Keogh, S. Lonardi, and B. Chiu, "A symbolic representation of time series, with implications for streaming algorithms," in *Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery*, pp. 2–11, ACM, 2003.

[8] E. Keogh, J. Lin, and A. Fu, "Hot sax: Finding the most unusual time series subsequence: Algorithms and applications," in *Proc. ICDM*, pp. 440–449, 2004.

[9] J. Lin, E. Keogh, L. Wei, and S. Lonardi, "Experiencing sax: a novel symbolic representation of time series," *Data Mining and knowledge discovery*, vol. 15, no. 2, pp. 107–144, 2007.

[10] Y. Li, J. Lin, and T. Oates, "Visualizing variable-length time series motifs," in *Proceedings of the 2012 SIAM international conference on data mining*, pp. 895–906, SIAM, 2012.

[11] https://github.com/jMotif/GI/blob/master/README.md. [Accessed June 14th 2018].

[12] https://grammarviz2.github.io/grammarviz2_site/morea/motif/experience-m1.html. [Accessed June 14th 2018].

[13] P. Senin, J. Lin, X. Wang, T. Oates, S. Gandhi, A. P. Boedihardjo, C. Chen, and S. Frankenstein, "Time series anomaly discovery with grammar-based compression.," in *EDBT*, pp. 481–492, 2015.

[14] https://en.wikipedia.org/wiki/Matthews_correlation_coefficient. [Accessed July 23rd 2018].

[15] https://en.wikipedia.org/wiki/Receiver_operating_characteristic. [Accessed August 13th 2018].

[16] https://en.wikipedia.org/wiki/Cohen%27s_kappa. [Accessed August 13th 2018].

[17] X. Wang, J. Lin, P. Senin, T. Oates, S. Gandhi, A. P. Boedihardjo, C. Chen, and S. Frankenstein, "Rpm: Representative pattern mining for efficient time series classification.," in *EDBT*, pp. 185–196, 2016.

[18] M. Gidea and Y. Katz, "Topological data analysis of financial time series: Landscapes of crashes," *Physica A: Statistical Mechanics and its Applications*, vol. 491, pp. 820–834, 2018.

[19] F. Chazal and B. Michel, "An introduction to topological data analysis: fundamental and practical aspects for data scientists," *arXiv preprint arXiv:1710.04019*, 2017.

[20] L. M. Seversky, S. Davis, and M. Berger, "On time-series topological data analysis: New data and opportunities," *2016 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 1014–1022, 2016.

[21] https://grammarviz2.github.io/grammarviz2_site/morea/optimization/reading-o1.html. [Accessed June 19th 2018].

[22] https://grammarviz2.github.io/grammarviz2_site/morea/optimization/reading-o1.html. [Accessed July 20th 2018].