

User Manual: Using RPM with TSAT

June 14, 2018

Contents

1	Introduction to TSAT	2
1.1	Interface Layout	2
1.2	Tools Introduction	3
1.2.1	Motif Discovery	3
1.2.2	Anomaly Detection	6
1.2.3	Representative Pattern Mining - RPM	6
1.3	Overview	6
2	Time Series Classification using RPM	6
2.1	File formats	6
2.2	Training	8
2.3	Testing	13
2.4	Testing Unlabeled Data	15
2.5	Saving a Trained RPM Model	16
2.6	Loading an RPM Model	19
2.7	Settings	22
2.7.1	Dynamic Time Warping	22
2.7.2	Iterations	27
3	Motif Discovery	28
3.1	File format	28
3.2	Guide to Motif Discovery	28
4	Anomaly Detection	28
4.1	File format	28
4.2	Guide to Anomaly Detection	28
5	Error Messages	28
5.1	File Errors	28
5.2	RPM Errors	28
6	FAQs	28

1 Introduction to TSAT

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

1.1 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.

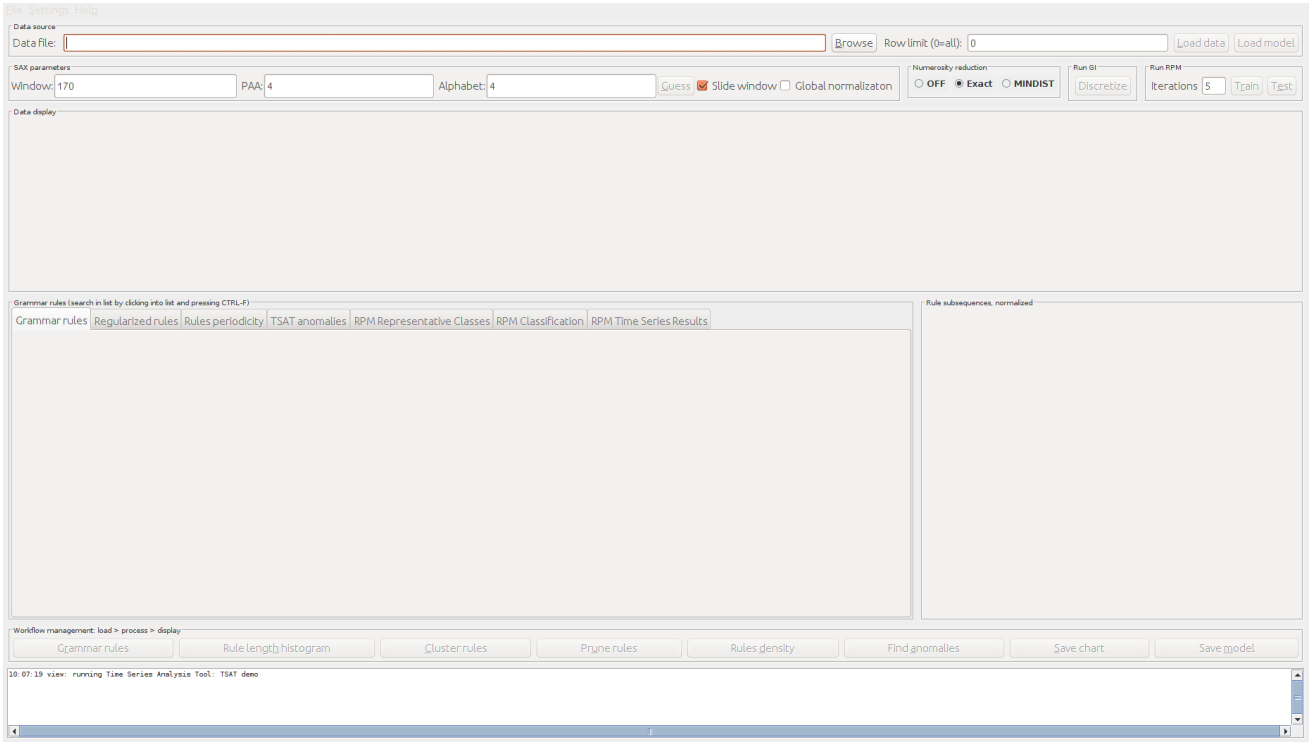


Figure 1: Initial state of the **TSAT** Graphical User Interface (GUI).

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.

1.2 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.

1.2.1 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**).

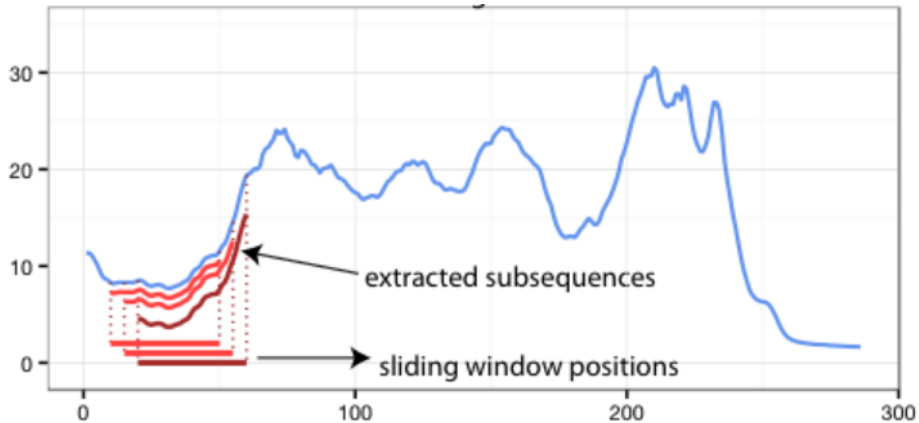


Figure 2: Subsequence extraction from a sliding window. Adopted from:
http://grammarviz2.github.io/grammarviz2_site/morea/assets/sax-error.png

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. 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].”

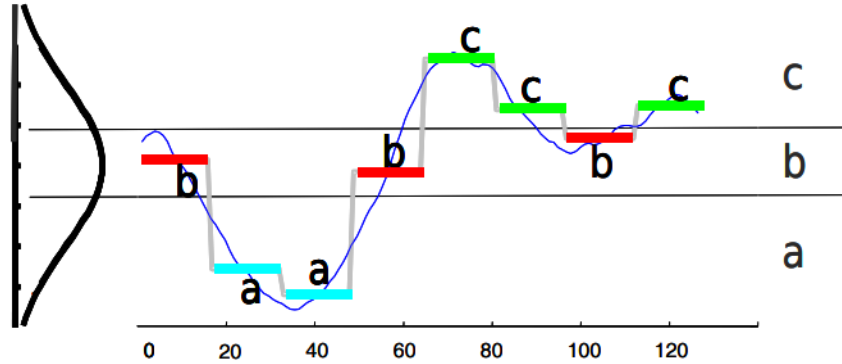


Figure 3: “A time series is discretized by first obtaining a PAA approximation and then using predetermined breakpoints to map the PAA coefficients into SAX symbols. In the example above, with $n = 128$, $w = 8$ and $a = 3$, the time series is mapped to the word baabccbc.” The figure and the included 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

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

is converted to the much smaller string using numerosity reduction:

$$S_2 = aac_1 abc_3 abb_4 acd_5 aac_6 abc_9$$

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:

$$\begin{aligned} S &\rightarrow BBA \\ A &\rightarrow B \text{ is} \\ B &\rightarrow \text{a rose} \end{aligned}$$

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].

1.2.2 Anomaly Detection

TSAT also implements anomaly detection called Rare Rule Anomaly detection due to how it identifies the anomalies. In **TSAT**, anomalies are anomalous subsequences which correspond to rare grammar rules. More specifically, anomalies are discords or the subsequences whose distance 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.

Anomaly detection is not sensitive to window size.

1.2.3 Representative Pattern Mining - RPM

1.3 Overview

2 Time Series Classification using RPM

TSAT implements Representative Pattern Mining or RPM (see Section 1.2.3 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. 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 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 2.1. Then the proper procedures to train (Section 2.2), test (Section 2.3), and review the results (Section ??) are presented step by step.

2.1 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 5: Examples of RPM Data

# 1 1 1 2 2 2 2 2 2	# 1.0000000e+000 1.0000000e+000 1.0000000e+000
0 0 0 122880 122880 122880 122880 0 0	-4.6427649e-001 -8.9697208e-001 -4.6469596e-001
0 0 0 0 0 0 0 0 0	-5.5504787e-001 -6.8568553e-001 -5.6773891e-001
0 0 0 0 0 0 0 0 0	-8.4284310e-001 -1.3513818e+000 -3.2022764e-002
0 0 0 0 0 0 0 0 0	-8.6589548e-001 -1.4586668e+000 -6.3504562e-001
0 0 0 0 0 0 0 0 0	-9.3639631e-001 -1.1653456e+000 -6.0282554e-001
0 0 0 0 0 0 0 0 0	-8.1726995e-001 -1.4039293e+000 -2.6685628e-001
0 0 0 0 0 0 0 0 0	-2.6361216e-001 -1.8217996e+000 -2.6706128e-001
0 0 0 0 0 0 0 0 0	-1.2580483e+000 -8.3160109e-001 -9.3104230e-001
0 0 0 0 0 0 0 0 0	-1.2503934e+000 -1.0163124e+000 -4.4938186e-001
0 0 0 0 0 0 0 0 0	-9.1830825e-001 -8.0353040e-001 -7.2134200e-001
0 0 0 0 0 0 0 0 0	-9.2210226e-001 -1.2595048e+000 -3.9727192e-001
0 0 0 0 122880 122880 0 0 0	-9.8448828e-001 -1.1392341e+000 -9.6212589e-001
	-1.2880511e+000 -8.7865203e-001 -1.4206669e+000

(a) Example 1

(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 than 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 5. 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. For example, <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 `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.

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.

2.2 Training

Once you have the data in the proper format and TSAT open training RPM can begin.

Step 1 First click on the “Browse” button under the “Data Sources” section of the window, as seen in figure 6.

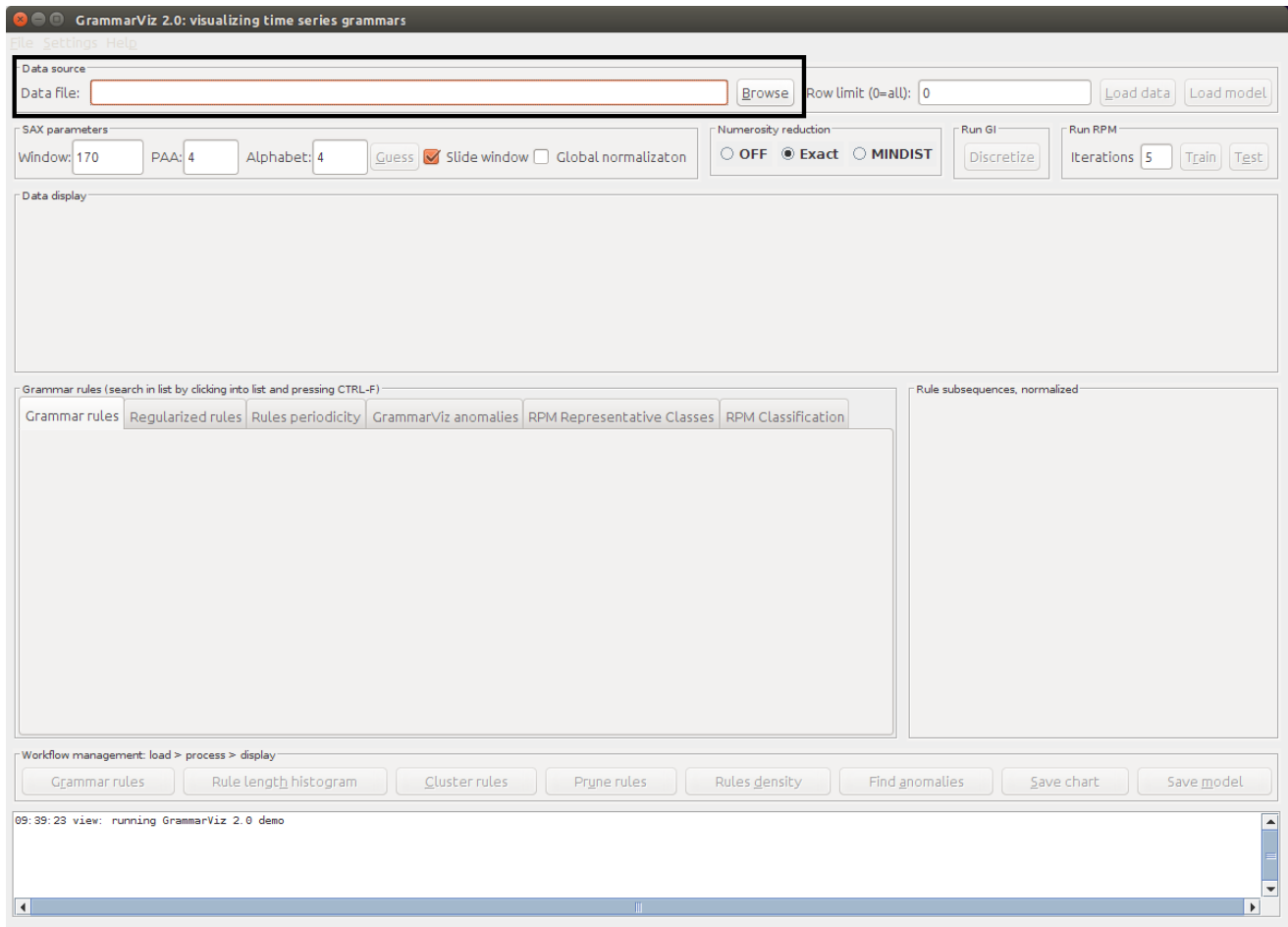


Figure 6: Open TSAT

Step 2 This should bring up the file browser prompt in figure 7. Using this prompt select the file containing the training set in the RPM compatible format, figure 8.

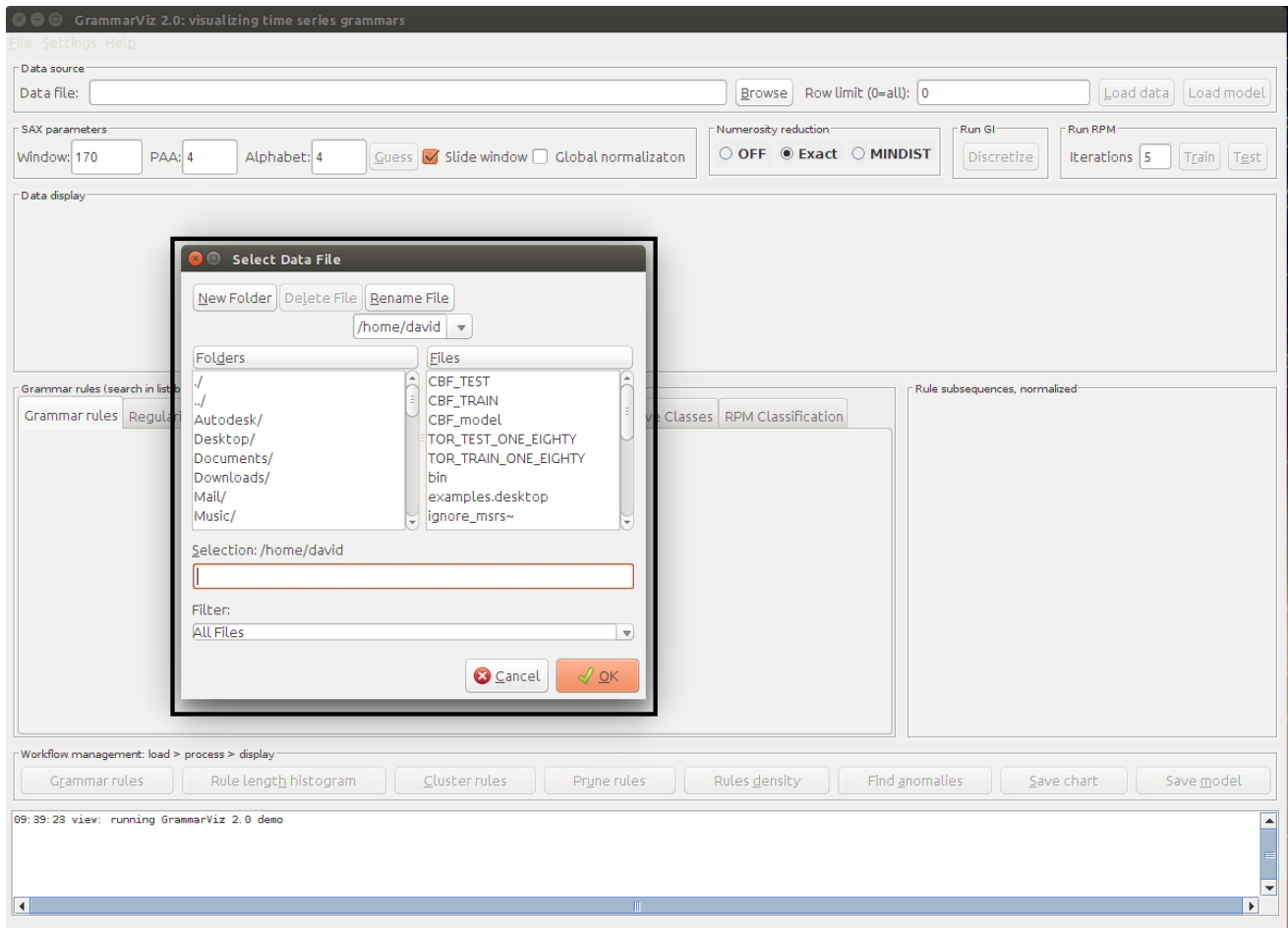


Figure 7: Open the file browser prompt

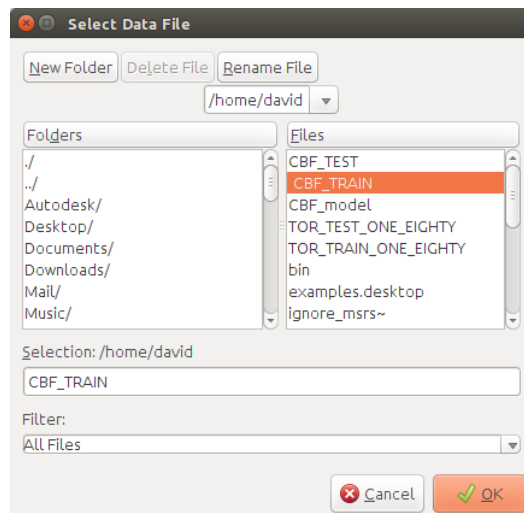


Figure 8: Browser prompt

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.

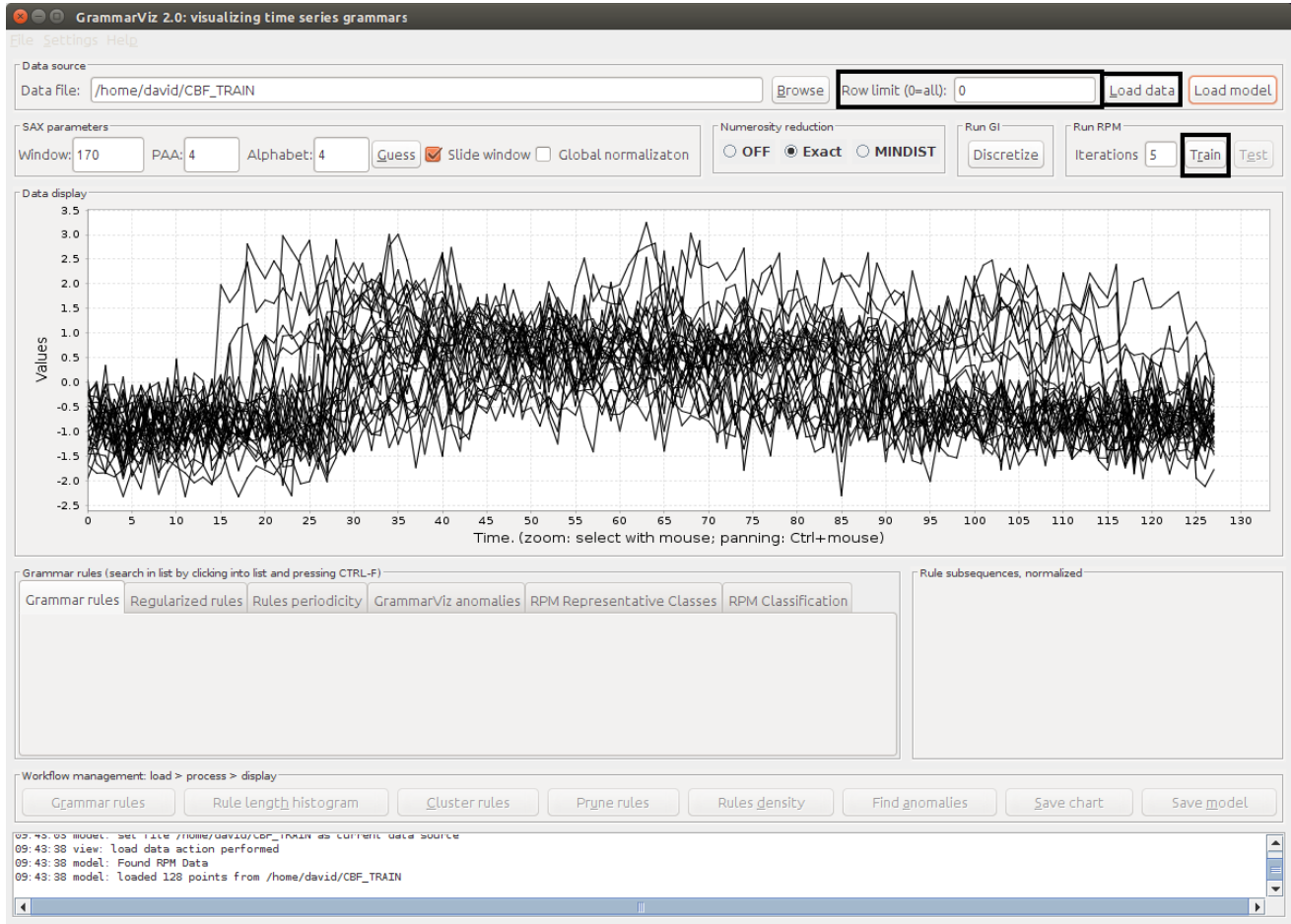


Figure 9: Loaded data

Hitting this 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 too long trying to refine the model. Once the training is complete the tab “RPM Representative Classes” will become populated with patterns RPM thinks 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 10: Representative Classes after Training

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

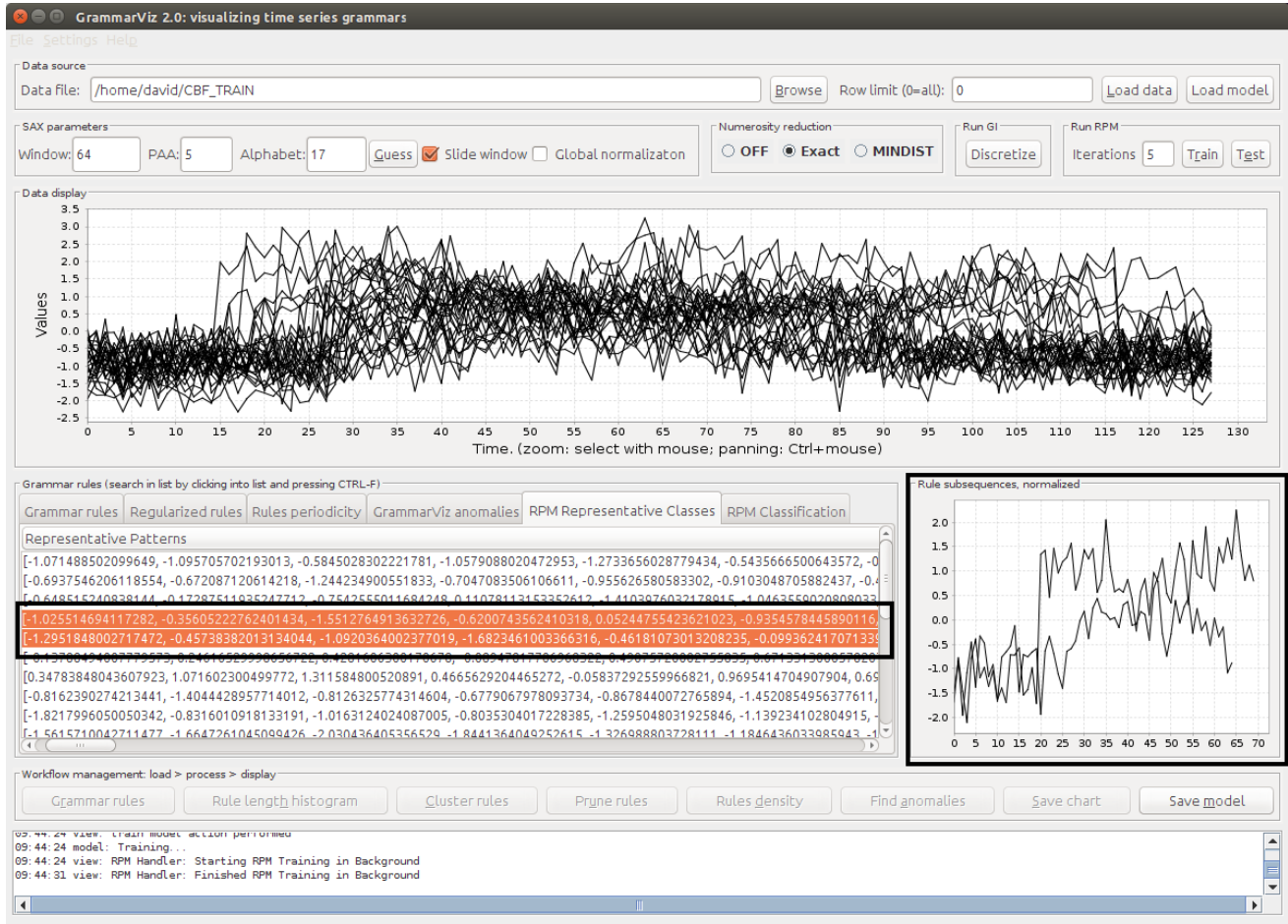


Figure 11: Representative pattern preview

2.3 Testing

Once the model has been 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.

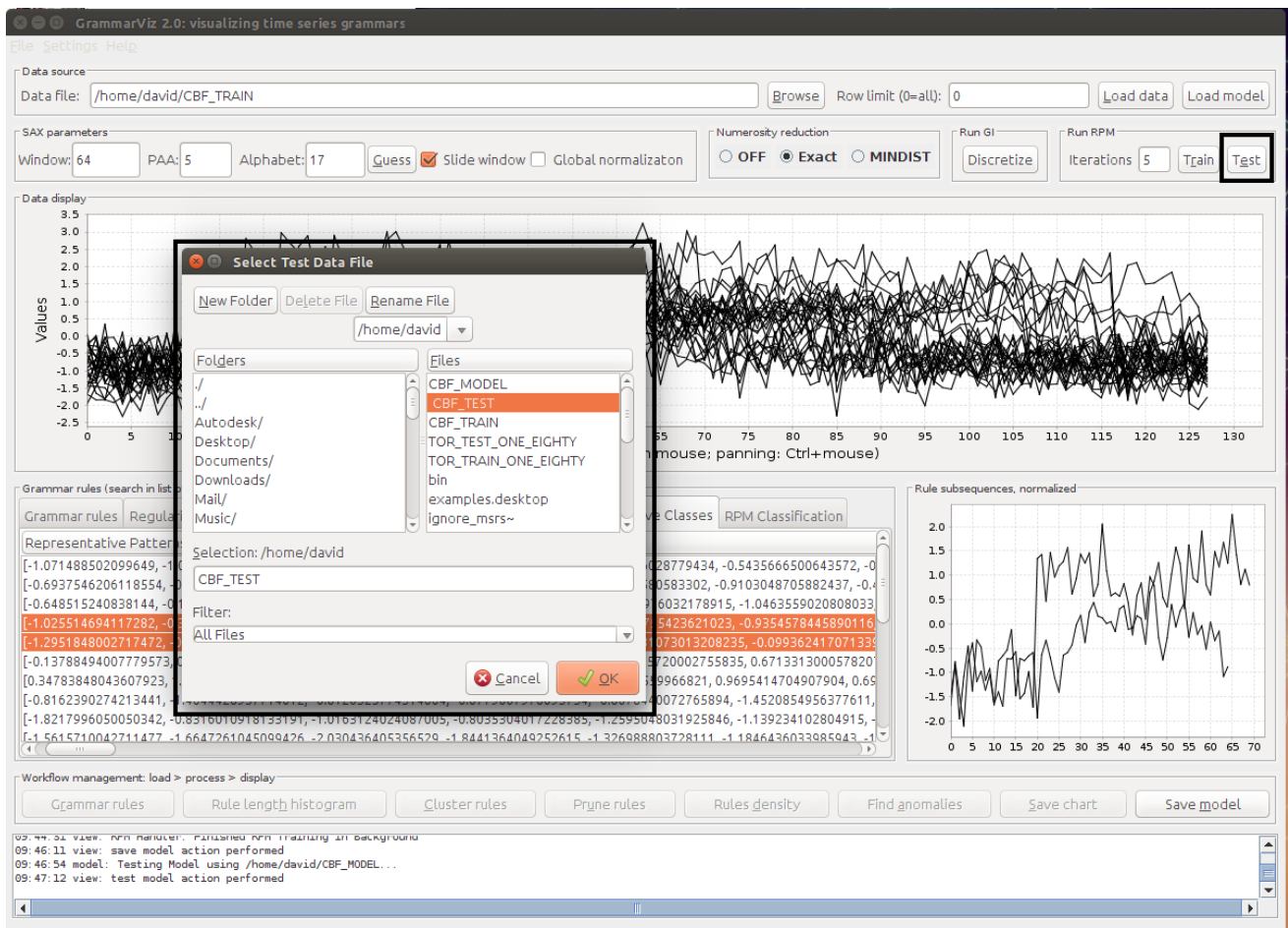


Figure 12: Testing the RPM model

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.

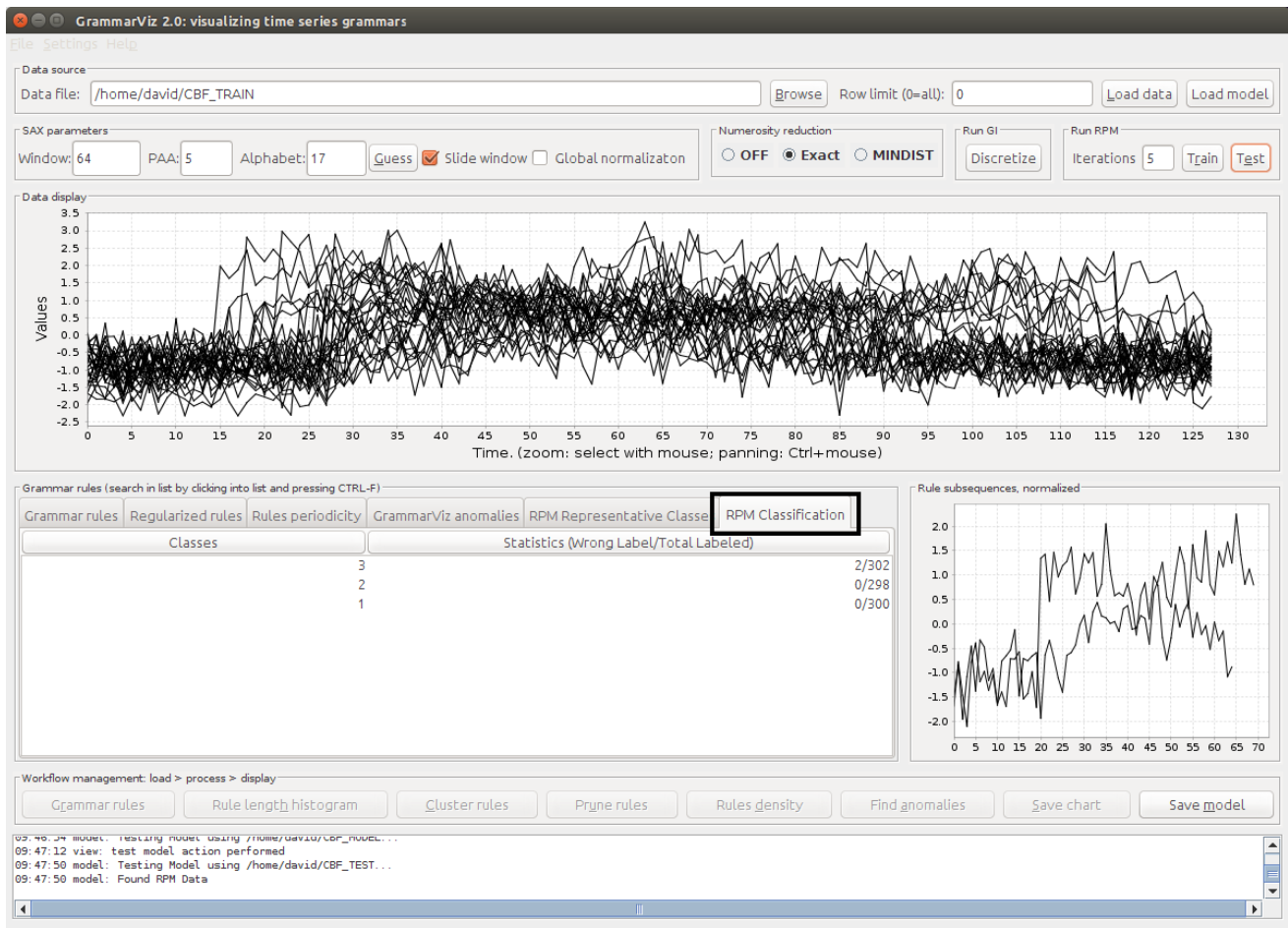


Figure 13: The results from the testing

2.4 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 14 the solid box has the label probability for each class and dashed box has the predicted class label.

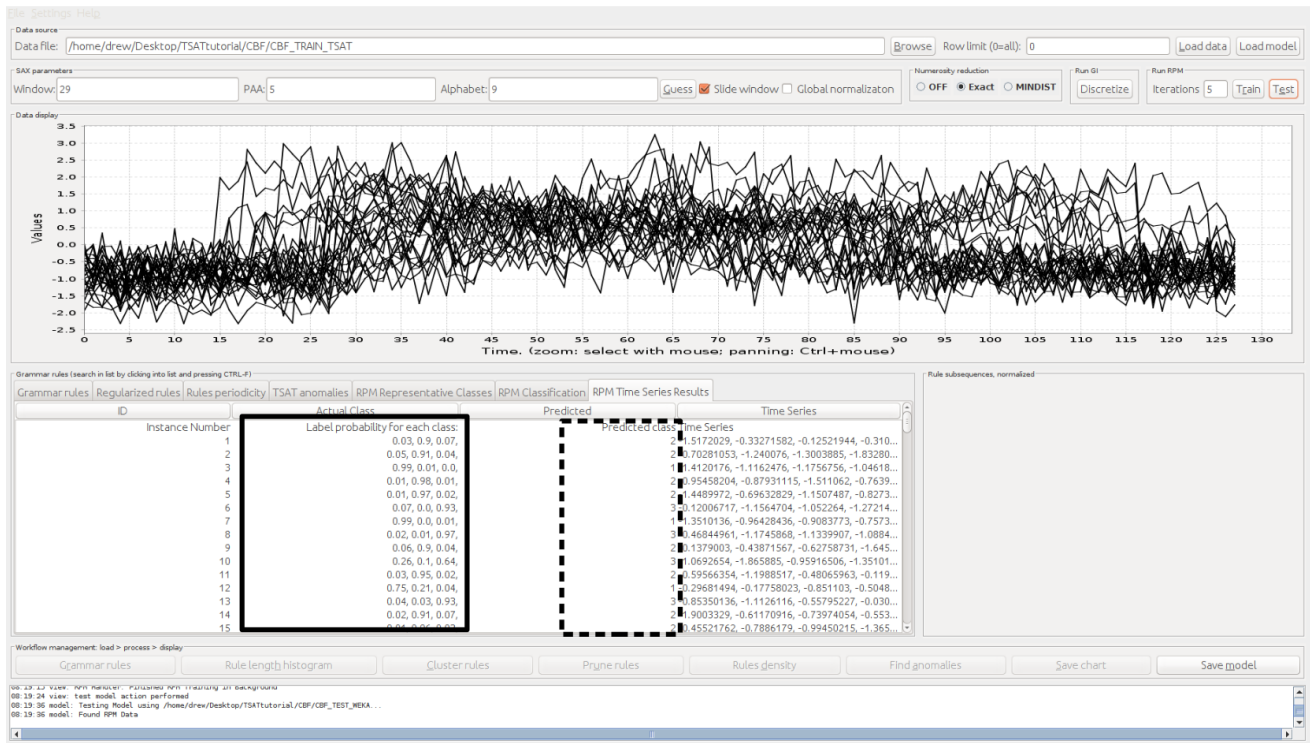


Figure 14: Solid box highlights the probability that the time series was in each of the different class labels and the dashed box highlights the predicted label.

2.5 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 up clicking the save model button, as in figure 15, a file browser prompt will appear.

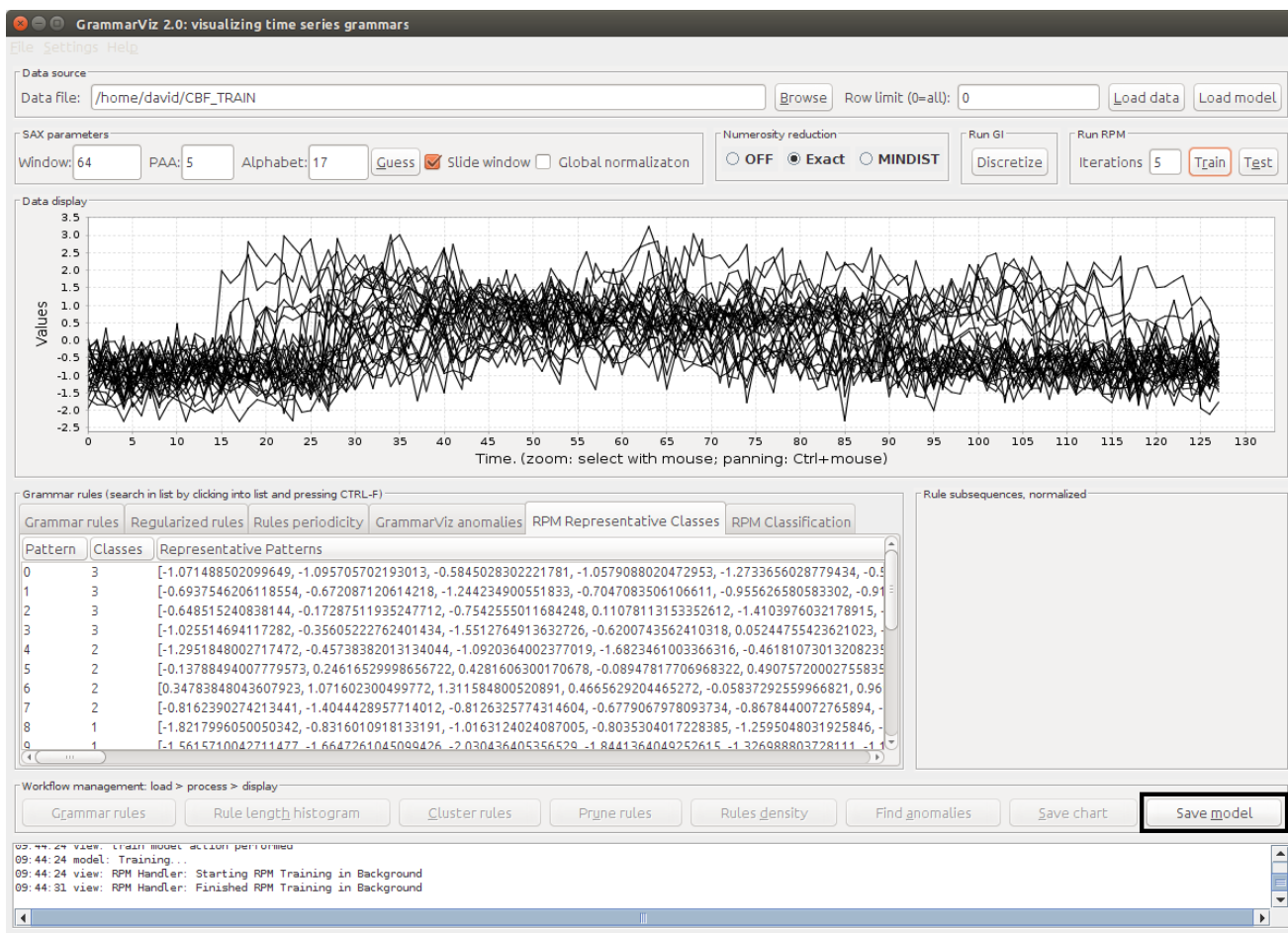


Figure 15: Saving the RPM model

Step 2 With the file browser prompt select a location to save the model and give it a name, then click the “OK” button to save the model.

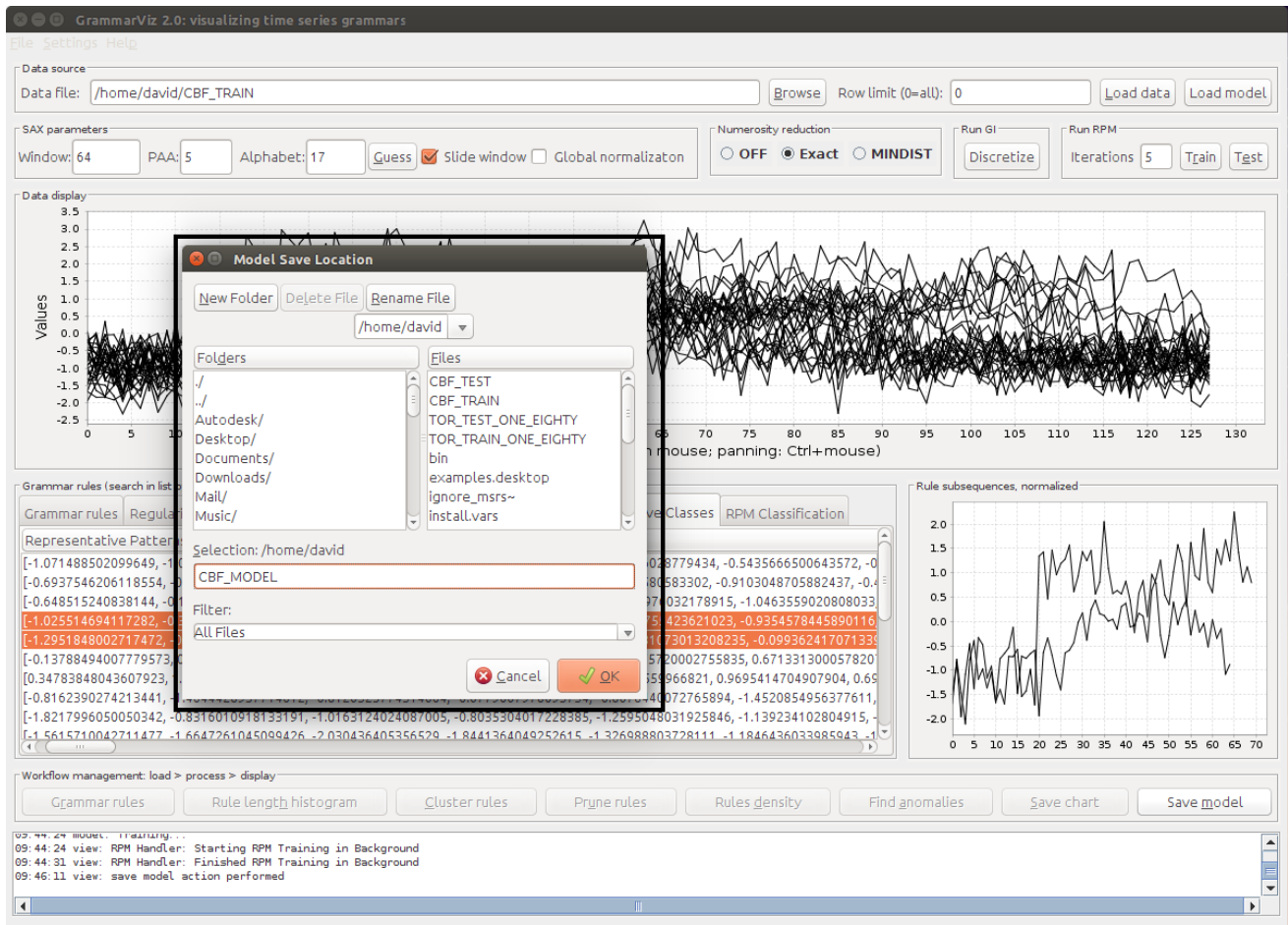


Figure 16: Saving the RPM model to file

2.6 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 17.

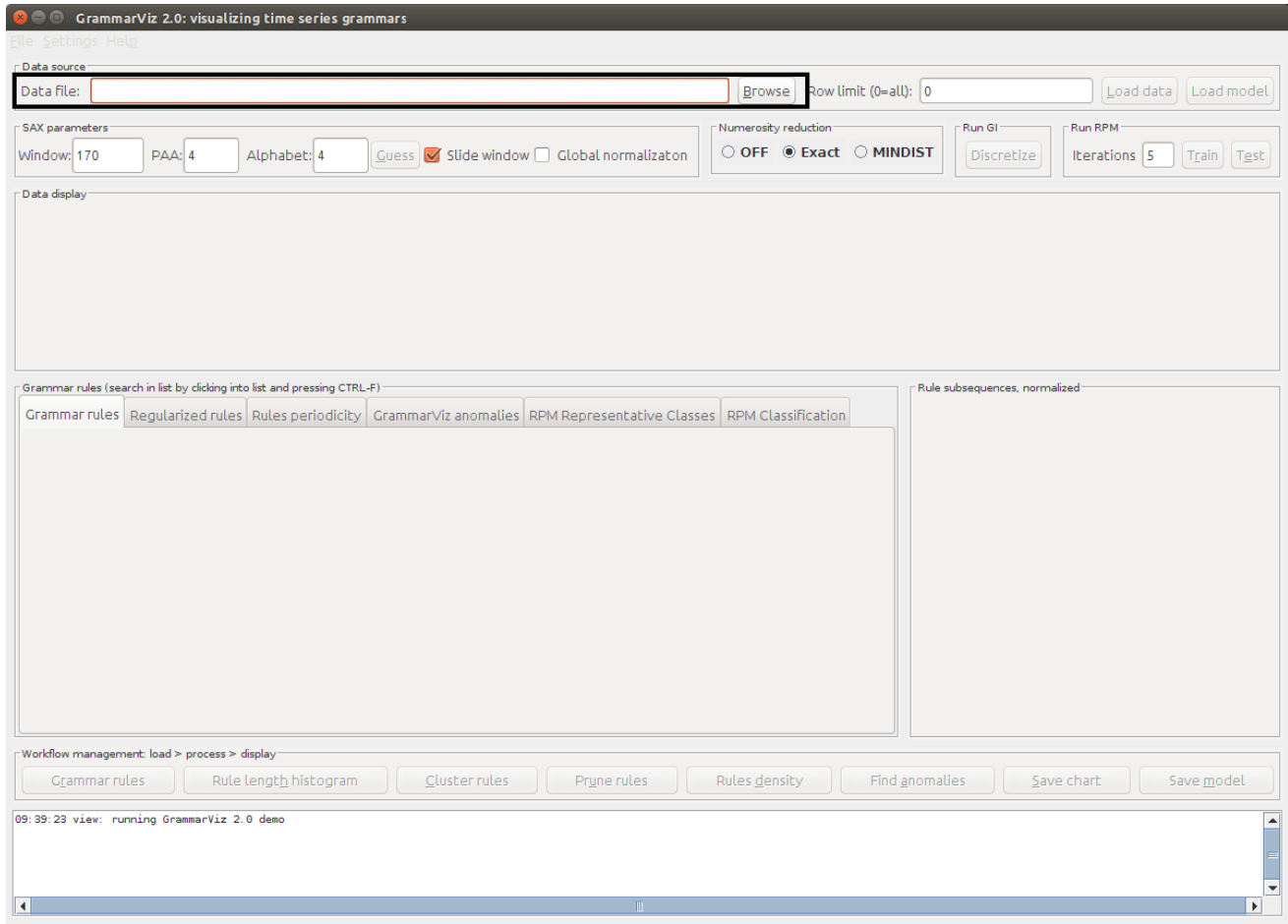


Figure 17: Loading a model

Step 2 This should bring up the file browser prompt in figure 18. Using this prompt select the previously saved model.

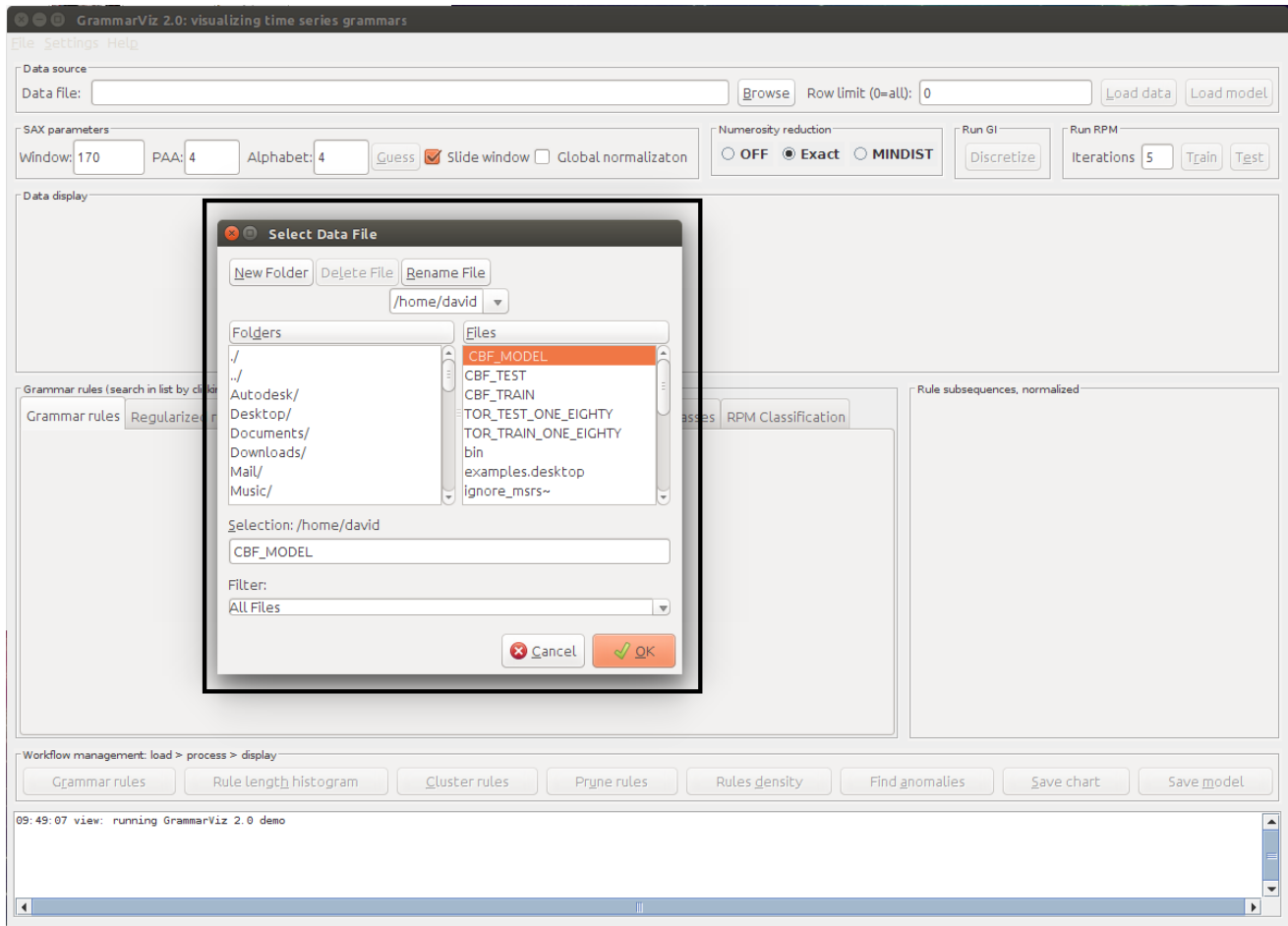


Figure 18: Open the file browser prompt

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 20, simply provide the data and the model will finish loading.

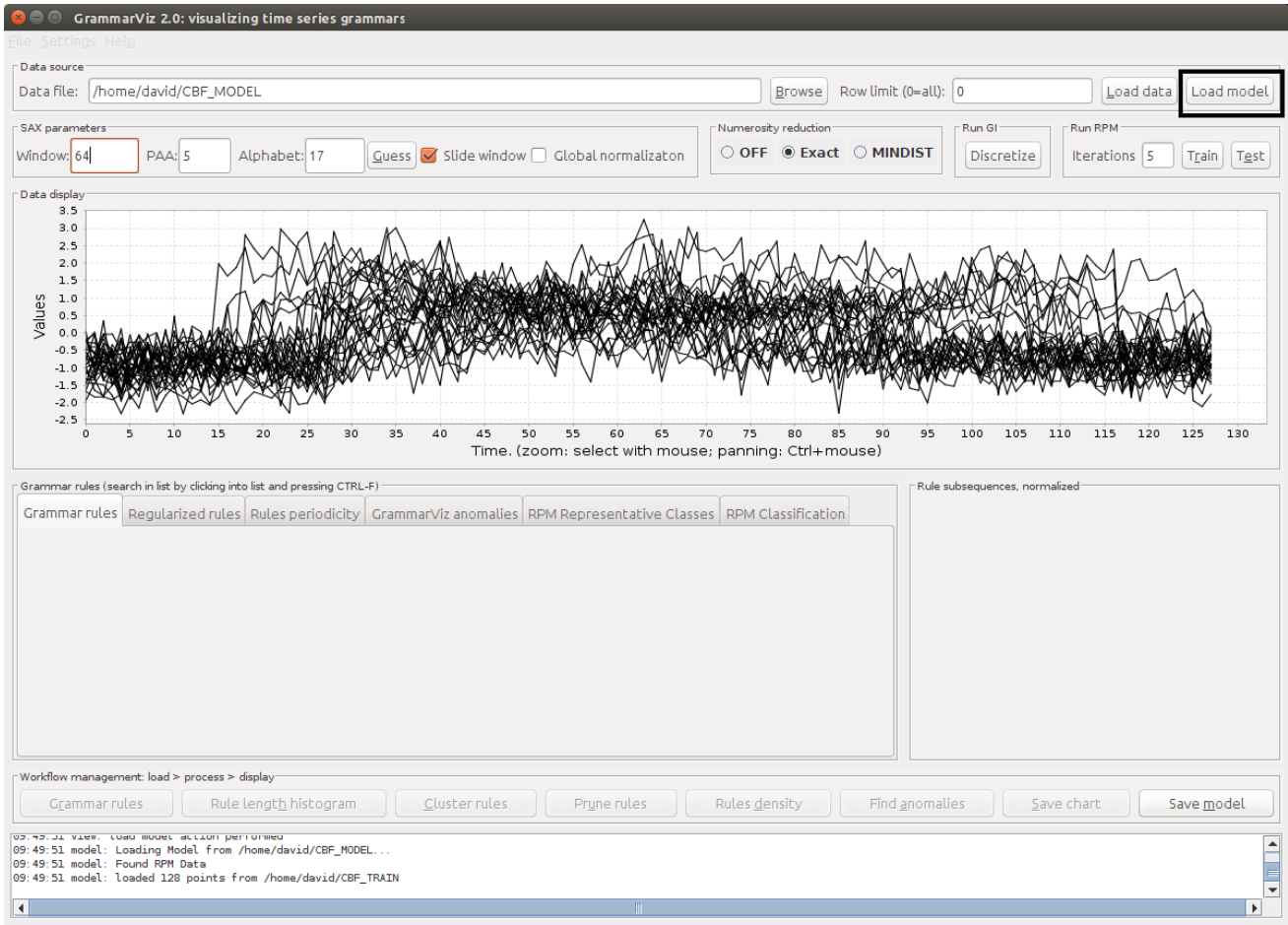


Figure 19: Model loaded

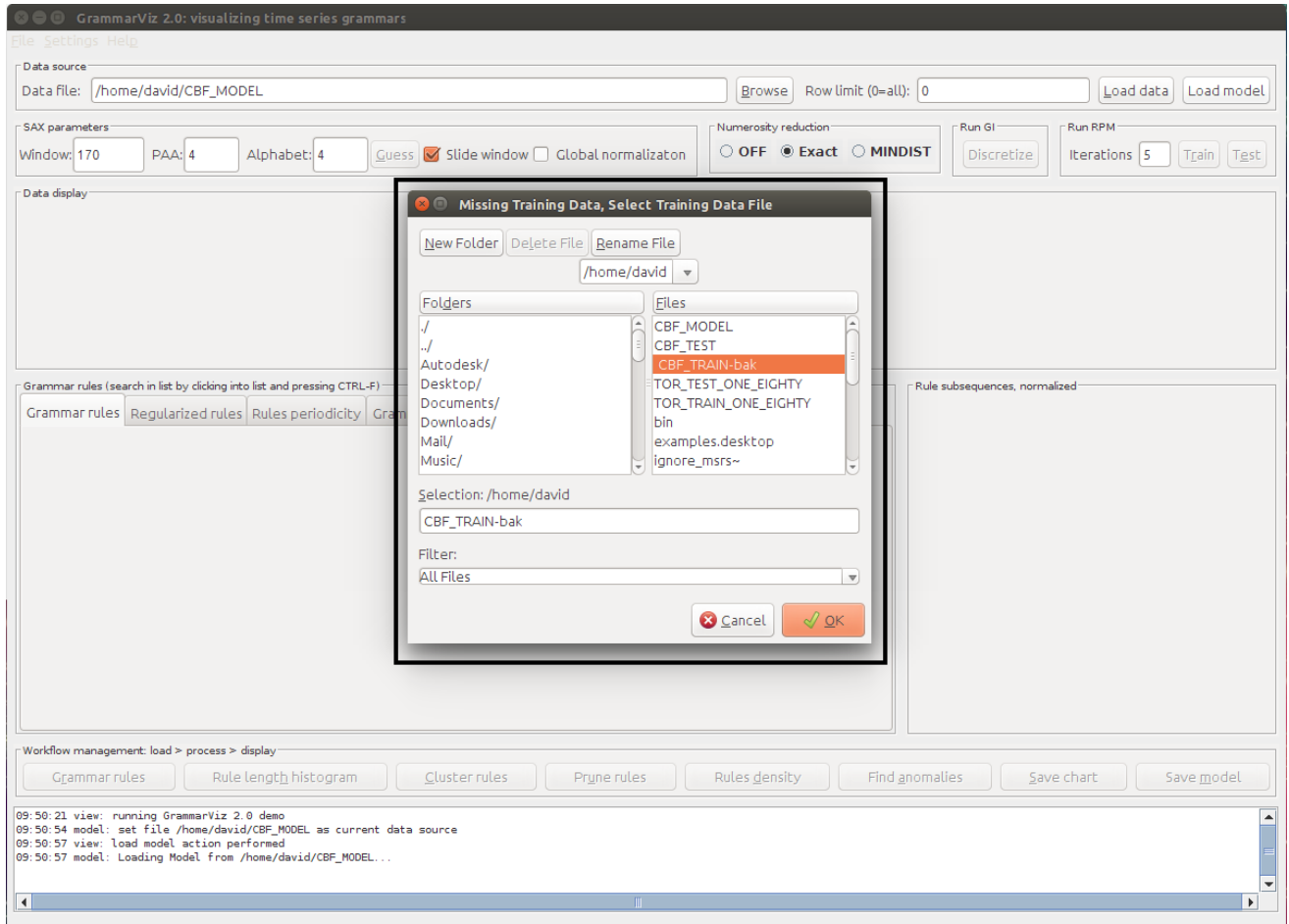


Figure 20: Missing training data file browser prompt

2.7 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.

2.7.1 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. There are a few good rules when choosing a window size, for one a window size greater than 10 will usually give bad results so 10 is considered a good starting point. Often for the more common types of data a 3-5 window size can be 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 21.

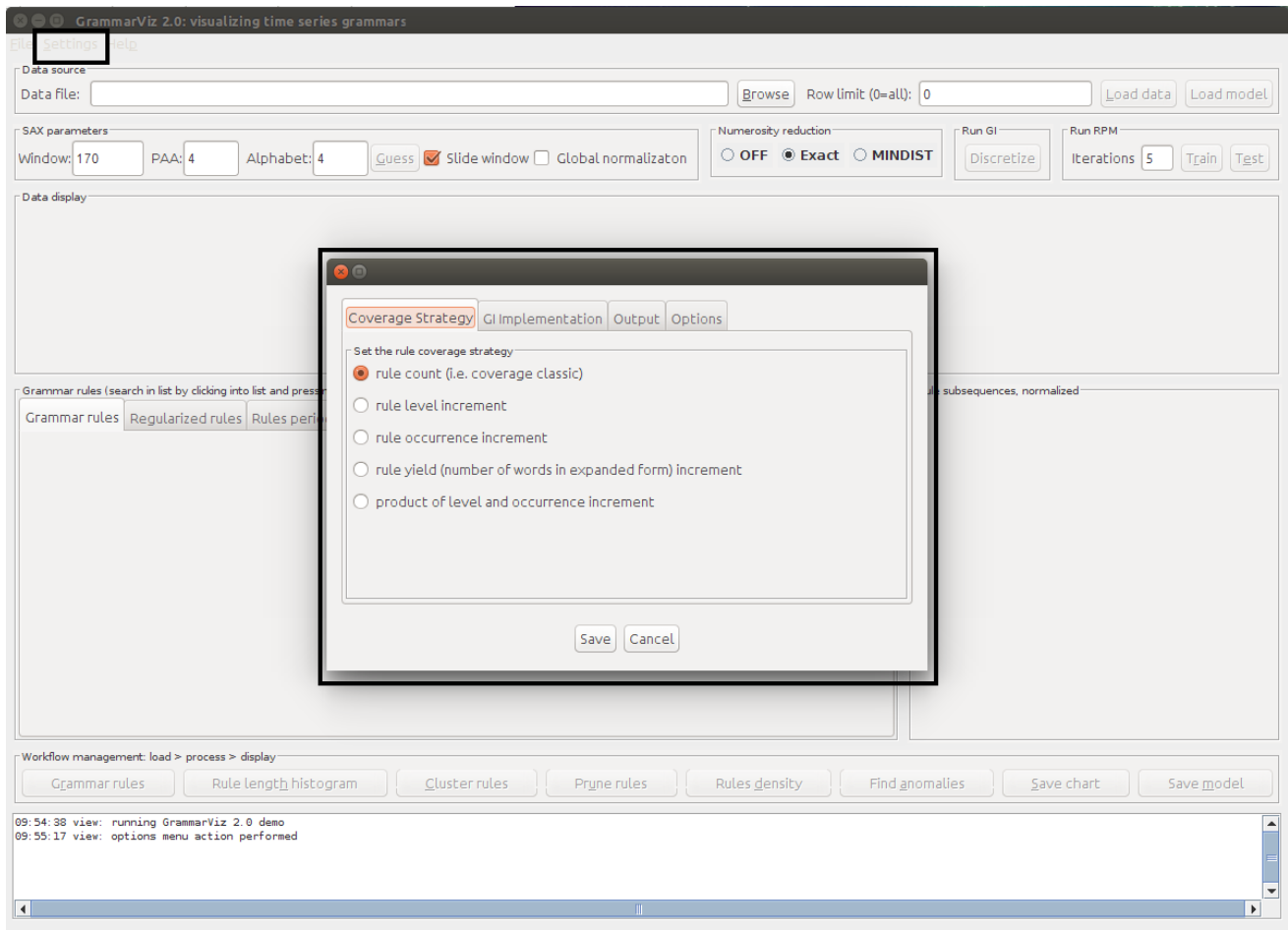


Figure 21: TSAT Settings Dialog

Step 2 Now click on the “Options” tab.

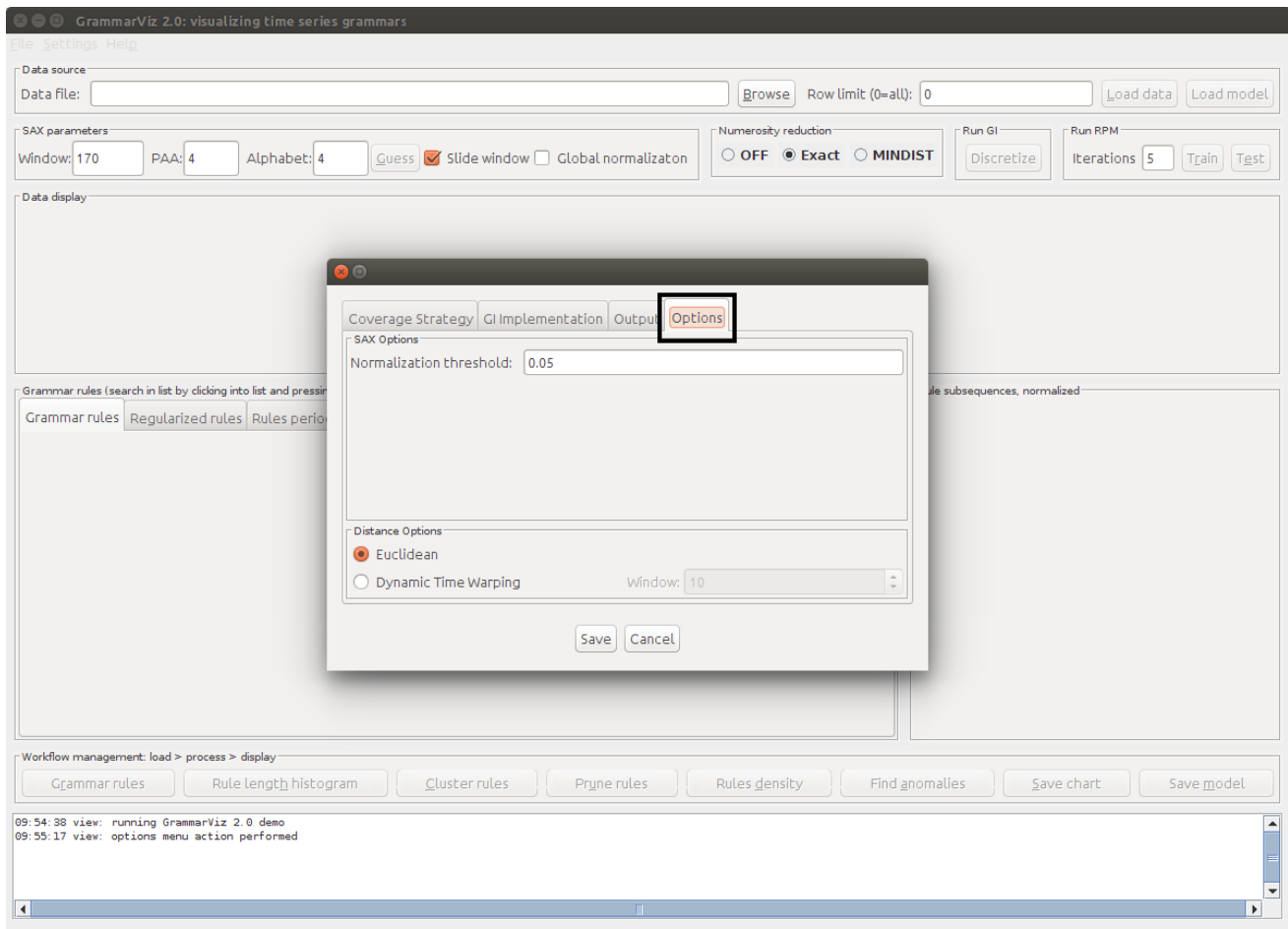


Figure 22: TSAT Settings Dialog Options

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

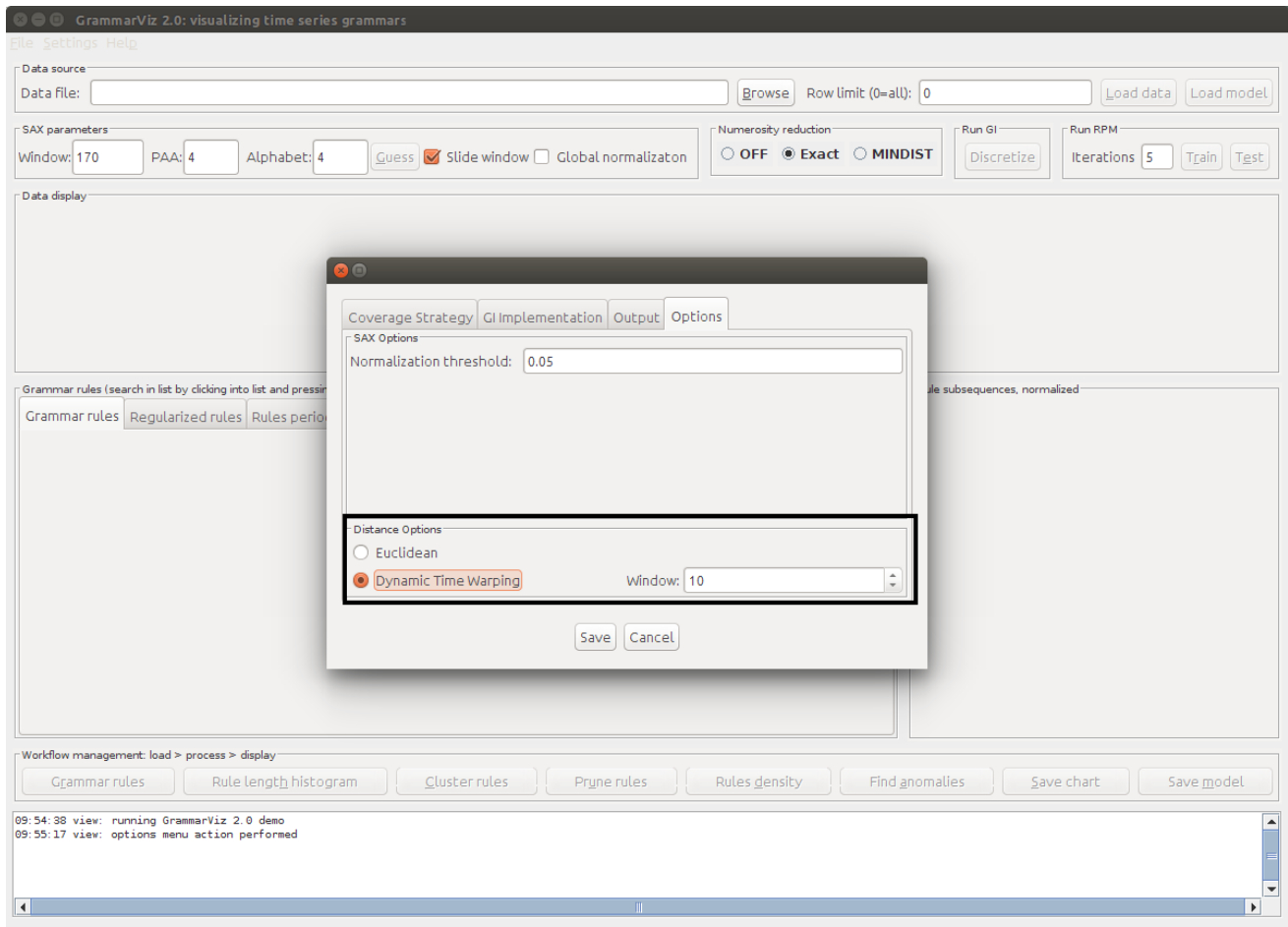


Figure 23: TSAT Settings Dialog Options DTW

2.7.2 Iterations

During the operation of RPM it goes through 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 24. 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 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 become 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.

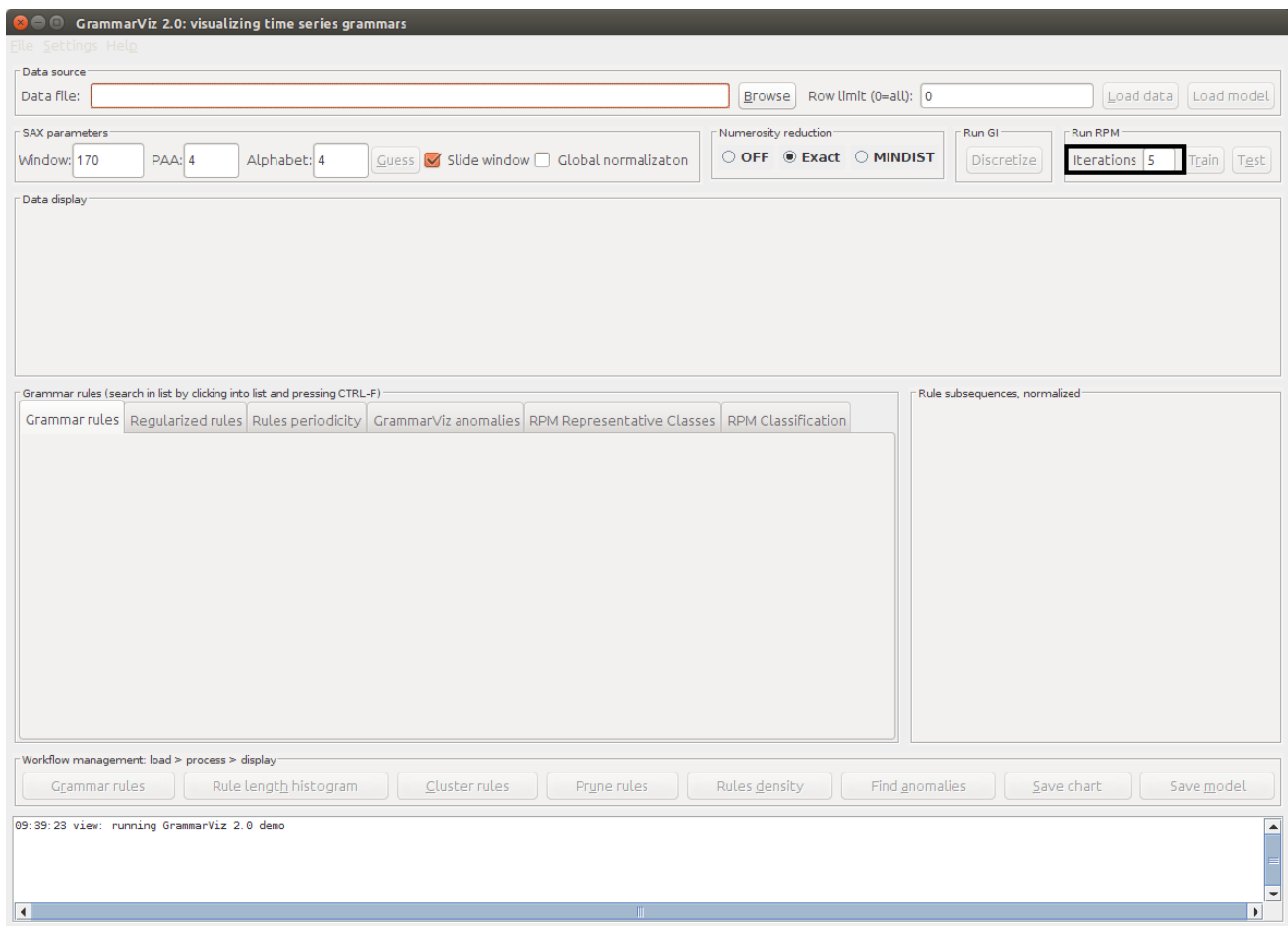


Figure 24: TSAT RPM Iteration Setting

3 Motif Discovery

3.1 File format

3.2 Guide to Motif Discovery

“Note however, that grammar induction step effectively mitigates improper sliding window selection.” (http://grammarviz2.github.io/grammarviz2_site/morea/motif/experience-m1.html).

4 Anomaly Detection

4.1 File format

4.2 Guide to Anomaly Detection

5 Error Messages

5.1 File Errors

5.2 RPM Errors

6 FAQs

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-0.0.1-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-0.0.1-SNAPSHOT-jar-with-dependencies.jar
```

To run the GUI from a shell you can do:

```
$ java -Xmx2g -jar target/tsat-0.0.1-SNAPSHOT-jar-with-dependencies.jar
```

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

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