Exploring Time Series Classification Techniques for ECG Data: Capturing Feature Dependencies for Reliable Diagnosis

Viveka Salinamakki
Electrical, Computer and Energy Engineering
University of Colorado Boulder
Colorado, United States
viveka.sg@gmail.com

***Abstract*—A time series refers to a sequence of observations or measurements taken at regular intervals of time. For an ECG signal, the observations are voltage levels over time. Machine Learning algorithms classify the target labels by considering the features as independent. This paper seeks to investigate different methods for reliably classifying these labels by taking into account the potential bias of features towards their previous timestamp values.**

***Index Terms*—ECG, Time Series Classification, PCA, RNN, LSTM, GRU, Random Forest for Time Series Classification, Dynamic Time Warping SVM, SVDD-TS**

I. Introduction

Electrocardiogram (ECG) measures electrical signal signals that depict the heart's activity. Doctors look at the PQRST points on the ECG as shown in Fig. 1 to determine if the person has a healthy heart or a few classifiable conditions which the techniques here aim to replace. The dataset used in this paper has 140 columns and 5 target labels out of which the first label means a healthy heart.

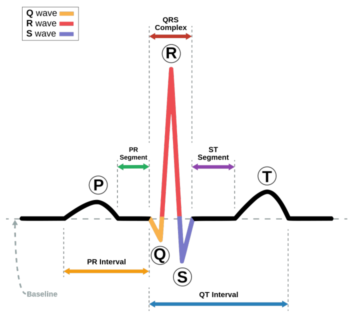
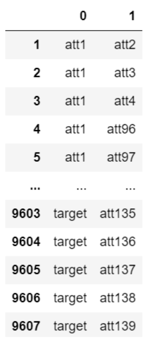


Fig. 1. ECG of a normal heart

The techniques aim to make sure that a dependence will be created between the data points at each time stamp. Hence the techniques are verified by testing the accuracy of the models for shuffled and unshuffled data. The unshuffled data are expected to perform well as the prediction of these conditions can be determined on the data alone. The shuffled data on the other hand even though have the same numbers the order of these numbers is different meaning that they represent a different signal and so must be misclassified.



	0	1
1	att1	att2
2	att1	att3
3	att1	att4
4	att1	att96
5	att1	att97
...	...	...
9603	target	att135
9604	target	att136
9605	target	att137
9606	target	att138
9607	target	att139

Fig. 2. Pairs of columns which are correlated with a value greater than $|0.3|$

To check the correlation between the columns or features of the data, the pairs of features with correlation value greater than $|0.3|$. As seen in Fig. 2, there are 9607 pairs of columns including the target label, and hence the features are correlated, not independent.

II. Logistic Regression with Principal Component Analysis (PCA)

The data was initially tested with Logistic regression, a multi-class classification algorithm to verify that it considers the columns as independent. The data is processed to obtain the principal components using Principal Component Analysis (PCA) to reduce the dimension of the data.

StandardScaler is used to resize the distribution before applying PCA so that the mean becomes 0 and the standard deviation becomes 1. The idea behind the StandardScaler is that variables that are measured at different scales do not contribute equally to the model's fit and the model's learning function, so it could create a bias. To deal with this potential problem, the data is standardized $(\mu = 0, \sigma = 1)$.

Principal Component Analysis (PCA) was performed on the standardized data and logistic regression was used for prediction using the scikit-learn library [4]. The accuracy versus the number of principal components is shown in Fig. 4

and Fig. 3. The accuracy doesn't vary by a significant value for the shuffled data. Although PCA yielded good results, it doesn't matter if the columns are jumbled, the same result remains the same. Hence the dependency of the columns is not considered and different techniques must be explored.

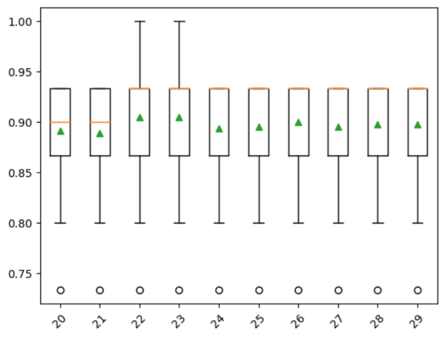


Fig. 3. Accuracy vs number of principal components for unshuffled data

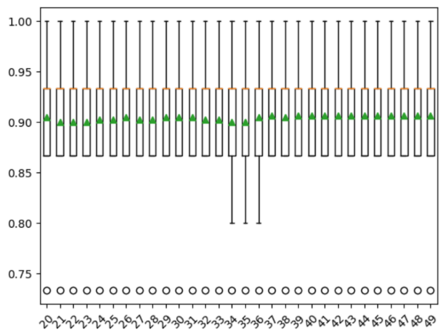


Fig. 4. Accuracy vs number of principal components for shuffled data

III. The Neural Network Approach: Recurrent Neural Network

In neural networks, feed-forward architecture sends the outputs of one layer as input to the next layer, while in convolutional neural networks (CNN), it is suitable for handling spatial data like images and videos by capturing local patterns, but not suitable for capturing temporal dependencies in sequential data.

On the other hand a recurrent neural network, by contrast, retains a memory of what it has processed in its recent previous steps. The process involves creating recurrent connections through temporal feedback loops. The output of each preceding step is used as input for the current step in the process. RNNs are therefore more appropriate for sequential data, like natural language processing (NLP) and time series data, because they capture the temporal dependencies between data points.

Preprocessing and data transformation is not required for this approach as the neural network takes care of the bias and transformations. So, Standardization is not necessary. Tanh activation function present in the neural network scales the range of the values to range between -1 and 1. This is performed so that the mathematical operations in the neural network don't blow up these values into huge numbers. However, the standard RNN architecture is susceptible to the ”Vanishing Gradient Problem,” which limits its ability to learn and make accurate predictions.

The vanishing gradient problem is a problem in standard RNNs where the gradient becomes too small during back-propagation, limiting the model's ability to capture long-term dependencies in sequential data. The problem is caused by the repeated multiplication of small gradient values during back-propagation, which can lead to earlier time steps having a smaller impact on the final output than later time steps. To reduce this problem, specialized RNN architectures like long short-term memory (LSTM) and gated recurrent unit (GRU) networks have been developed to better capture long-term dependencies in sequential data.

A. *Long Short Term Memory (LSTM)*

LSTM as stated was developed to capture the long-term dependencies and hence eliminate the vanishing gradient problem from the standard RNN. LSTMs have three gates as mentioned below.

1) The Forget Gate determines what information to discard from the previous cell state, based on the current input and previous hidden state.
2) The input gate determines what new information to add to the cell state, based on the current input and previous hidden state.
3) The output gate decides the next hidden state, based on the current input and cell state.

The memory component of LSTM is the cell state as shown in Fig. 5.

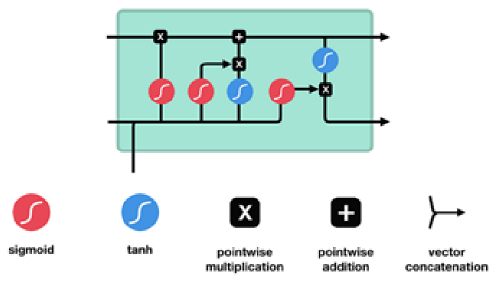
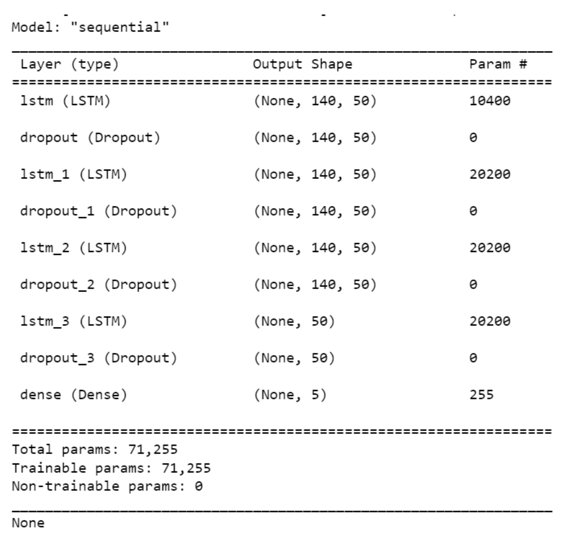


Fig. 5. Architecture of LSTM

In addition to LSTMs, another technique that can improve the performance of neural networks is bagging, which involves training multiple models on different subsets of the training data and combining their predictions. Dropout is a popular form of regularization used in neural networks that can be thought of as a form of bagging, where a random subset of neurons is temporarily ”dropped out” during training to prevent overfitting.

A four-layer model was used as it theoretically captures the long-term dependencies as shown in Fig. 6. The four LSTM layers were each followed by a corresponding dropout layer.

A Dense layer followed the last dropout layer which is used for the prediction.



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 140, 50)           10400

 dropout (Dropout)           (None, 140, 50)           0

 lstm_1 (LSTM)               (None, 140, 50)           20200

 dropout_1 (Dropout)         (None, 140, 50)           0

 lstm_2 (LSTM)               (None, 140, 50)           20200

 dropout_2 (Dropout)         (None, 140, 50)           0

 lstm_3 (LSTM)               (None, 50)                20200

 dropout_3 (Dropout)         (None, 50)                0

 dense (Dense)               (None, 5)                 255

=================================================================
Total params: 71,255
Trainable params: 71,255
Non-trainable params: 0
_________________________________________________________________
None
```

Fig. 6. LSTM model summary

The probabilities for the labels were computed with the help of the softmax function. The softmax function is an extension of the sigmoid function which is an activation function for one label.

(i) The sigmoid activation function gives a value between 0 and 1. The probability that the data point belongs to one class does not take into account the probability of the other classes.
(ii) Similar to the sigmoid activation function the SoftMax function returns the probability of each class as shown in the following equation.

$$softmax(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

Where z_j are the labels for the target variable

The confusion matrices for unshuffled and shuffled data are as shown in Fig. 7 and Fig. 9 respectively.

The description for scores for the classifications in Fig. 8 and Fig. 10 are as follows. The accuracy scores will be the main focus of the analysis.

(i) Accuracy measures overall correctness
(ii) Precision measures true positives among predicted positives
(iii) Recall measures true positives among actual positives
(iv) F1-score balances precision and recall

The simulation of causality or creating the bias towards the previous data is working as expected as shuffling of the columns did not yield the same result as the unshuffled data. The accuracy is very low for the shuffled data compared to the unshuffled data(92.67% compared to 6% in this case) meaning that a dependency has been created between the columns.

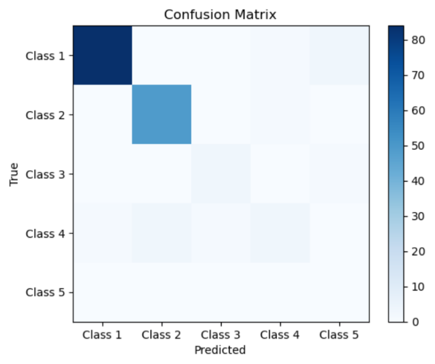
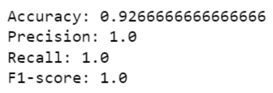


Fig. 7. Confusion matrix for the four-layer LSTM model for unshuffled data



```
Accuracy: 0.9266666666666666
Precision: 1.0
Recall: 1.0
F1-score: 1.0
```

Fig. 8. Scores for the four-layer LSTM model for unshuffled data

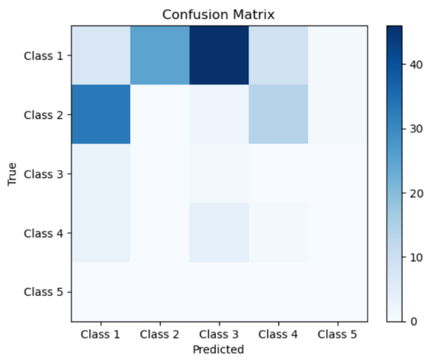
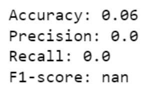


Fig. 9. Confusion matrix for the four-layer LSTM model for unshuffled data



```
Accuracy: 0.06
Precision: 0.0
Recall: 0.0
F1-score: nan
```

Fig. 10. Scores for the four-layer LSTM model for shuffled data

B. Gated Recurrent Unit (GRU)

Gated Recurrent Unit (GRU) is a faster version of the LSTM. It has two main gates as shown in Fig. 11.

1) The update gate acts similarly to the forget and input gates in an LSTM, deciding what information to keep and what new information to add.
2) The reset gate is another gate used to determine how much past information to forget.

In contrast to the LSTM, the GRU has two gates. This simplified architecture makes the GRU faster to train and requires fewer parameters than the LSTM, which has three gates and a cell state. However, the LSTM may be better theoretically at capturing long-term dependencies in sequential data due to its more complex architecture but there was not much difference for this dataset. GRU does not have a memory

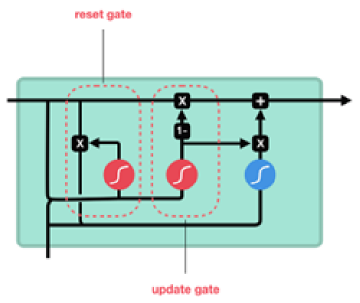
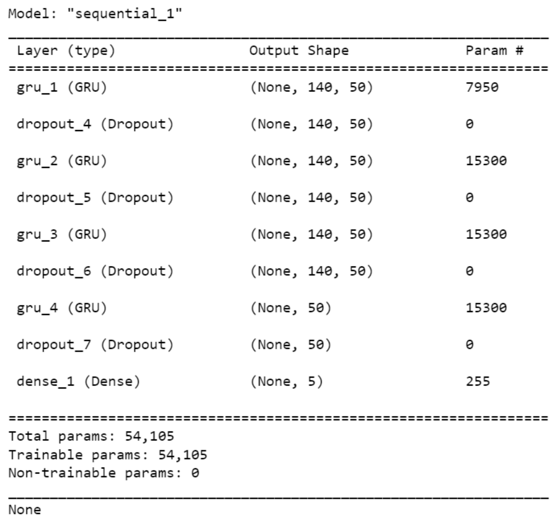


Fig. 11. Architecture of GRU

and so has the potential of getting affected by the vanishing gradient for long-term dependencies.

A similar four-layer model was used replacing LSTM with GRU to compare the performance between LSTM and GRU as shown in Fig. 12.



```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 gru_1 (GRU)                 (None, 140, 50)           7950

 dropout_4 (Dropout)         (None, 140, 50)           0

 gru_2 (GRU)                 (None, 140, 50)           15300

 dropout_5 (Dropout)         (None, 140, 50)           0

 gru_3 (GRU)                 (None, 140, 50)           15300

 dropout_6 (Dropout)         (None, 140, 50)           0

 gru_4 (GRU)                 (None, 50)                15300

 dropout_7 (Dropout)         (None, 50)                0

 dense_1 (Dense)             (None, 5)                 255

=================================================================
Total params: 54,105
Trainable params: 54,105
Non-trainable params: 0
_________________________________________________________________
None
```

Fig. 12. GRU model summary

The confusion matrices for unshuffled and shuffled data are shown in Fig. 13 and Fig. 15 respectively.

The GRU captures the dependencies just as well as the LSTM for this case even though LSTM theoretically captures the long-term dependencies better due to the presence of a memory component. So, LSTM was chosen and continued.

C. Robustness of LSTM

The robustness of the model was tested by randomly dropping some values in the test data. Test data was tested with 0.2, 0.4, and 0.6 fractions of values being dropped. The performance is shown in Fig. 17, Fig. 18, and Fig. 19 respectively.

The performance doesn't drop significantly and intuitively seems like the model is using the patterns of the PQRST regions[I] to predict the labels. As the columns are well correlated, values can be inferred using the neighboring columns which is probably the reason for the good accuracy.

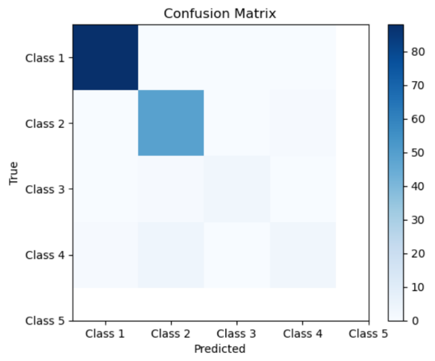


Fig. 13. Confusion matrix for the four-layer GRU model for unshuffled data

```
Accuracy: 0.9533333333333334
Precision: 1.0
Recall: 1.0
F1-score: 1.0
```

Fig. 14. Scores for the four-layer GRU model for unshuffled data

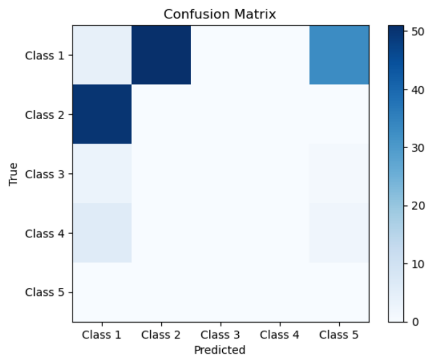


Fig. 15. Confusion matrix for the four-layer GRU model for unshuffled data

```
Accuracy: 0.02666666666666667
Precision: 0.0
Recall: 0.0
F1-score: nan
```

Fig. 16. Scores for the four-layer GRU model for shuffled data

IV. Using the rows as vectors to train the models

The idea was to provide the models with vectors to observe the performance of these models. Further, the vector would have been broken into different-sized vectors to find the optimal size for the best accuracy. It was unsuccessful as the model inherently broke down these vectors into individual elements which is exactly how the dataset is broken down and used to train the models.

V. Random Forest Classification

Random Forest is an ensemble learning algorithm used for classification and regression. It generates multiple decision trees by selecting random subsets of the training data and then combines the predictions of those trees to make a final prediction. The trees however are not put together sequentially according to the observations seen through the implementation.

```
Percentage of test values dropped: 20.0
5/5 [==============================] - 0s 82ms/step
Accuracy: 0.8933333333333333
Precision: 0.9772727272727273
Recall: 0.8958333333333334
F1-score: 0.9347826086956522
```

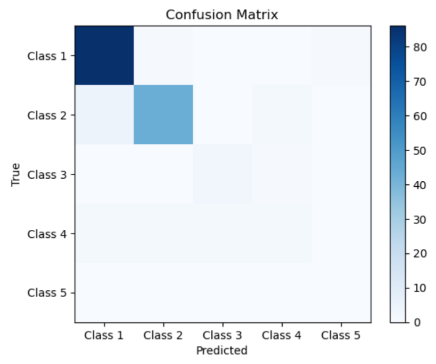


Fig. 17. Confusion matrix for the LSTM model with 20% of the values dropped

```
Percentage of test values dropped: 40.0
5/5 [==============================] - 0s 77ms/step
Accuracy: 0.9
Precision: 1.0
Recall: 0.8958333333333334
F1-score: 0.945054945054945
```

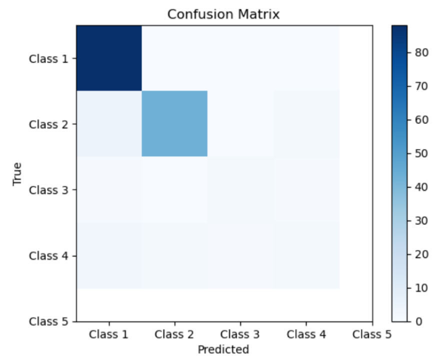


Fig. 18. Confusion matrix for the LSTM model with 40% of the values dropped

```
Percentage of test values dropped: 60.0
5/5 [==============================] - 0s 72ms/step
Accuracy: 0.88
Precision: 0.9767441860465116
Recall: 0.875
F1-score: 0.923076923076923
```

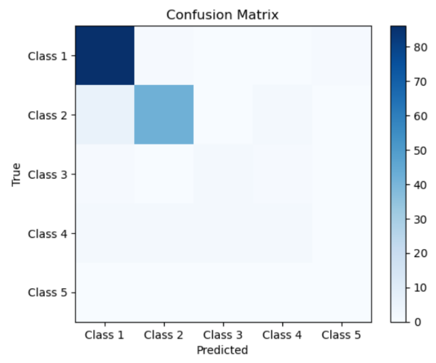


Fig. 19. Confusion matrix for the LSTM model with 60% of the values dropped

Time series classification can be challenging because of complex temporal dependencies such as trends and seasonal patterns. This was verified using implementation. The accuracies were similar for both shuffled and unshuffled data.

VI. Random Forest for Time Series Classification

The Random Forest Classifier is adapted to time series by converting the time series dataset to a supervised learning dataset. It involves transforming the dataset from a sequence prediction problem to a standard regression or classification problem. This is done by creating a new dataset in which each row represents a sample or observation and each column represents a feature or input variable. Therefore the previous values create a bias toward the current value.

The conversion of the dataset to a supervised dataset was implemented from scratch. The results of this approach for unshuffled and shuffled data can be seen in Fig. 20 and Fig. 21 respectively.

```
The mean absolute error was found to be  0.12
```

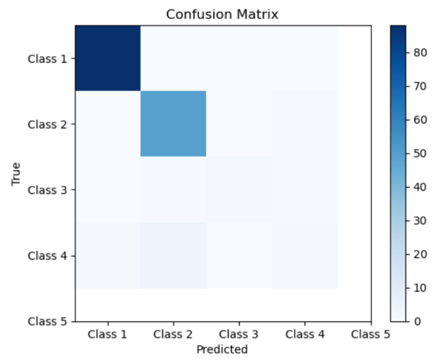


Fig. 20. Mean absolute error and the confusion matrix prediction made by the Random Forest Classifier for Time Series for unshuffled data

```
The mean absolute error was found to be  0.12
```

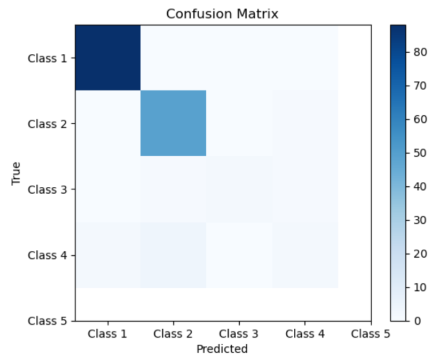


Fig. 21. Mean absolute error and the confusion matrix prediction made by the Random Forest Classifier for Time Series for shuffled data

The accuracies for the shuffled and unshuffled data came out similar according to Fig. 20 and Fig. 21. According to the papers [10] and [11], this approach assumes that the time series is stationary, meaning that its statistical properties (e.g., mean, variance) do not change over time but ECG data is typically a non-stationary time series because it exhibits changes in mean, variance, and correlation over time due to various physiological factors such as heart rate, respiratory rate, and body position.

VII. Dynamic Time Warping SVM (DTW-SVM)

Dynamic Time Warping Support Vector Machine (DTWSVM) is a modified version of the standard Support Vector Machine (SVM) algorithm designed to work with time series data. It uses Dynamic Time Warping (DTW) to measure the similarity between time series, and this similarity measure is incorporated into the SVM optimization problem to improve classification accuracy.

(i) DTW distance as the kernel function: In DTWSVM, the DTW distance between two time series is used as the kernel function. It measure of similarity between two time series by finding the distance between them. The differences in the timing and duration of events are taken into account making it well-suited for time series classification problems.
(ii) Weighted SVM: To incorporate the DTW distance into the SVM optimization problem, the weighted SVM algorithm is used. The data points are normalized with these DTW weights.
(iii) Multi-class classification: SVM is a binary classification algorithm and to extend it to a multi-class classification problem, multiple binary classifiers are trained, and the outputs are combined i.e., each class is classified separately and combined in the end.

This approach was implemented from scratch and suffered the same problem as the KNN algorithm done from scratch. Calculating the distance is time-consuming and so the algorithm did not stop running in a reasonable amount of time.

VIII. Support Vector Data Description for Time Series (SVDD-TS)

SVDD-TS is an algorithm that is designed to work with time series data and is based on the Support Vector Data Description (SVDD) algorithm. In one-class classification problems, the SVDD algorithm maps data points into a high-dimensional space and finds the smallest sphere in that space that contains all the data points. This sphere is called the support region, and the boundary between the support region and the rest of the space is called the support boundary.

The SVDD-TS algorithm uses the same principle as SVDD but is specifically designed for time series data. To represent each time series as a high-dimensional vector, SVDD-TS uses a time-delay embedding technique. This technique converts a one-dimensional time series into a multi-dimensional space by embedding each time point into a higher-dimensional space based on the values of neighboring time points. This allows the temporal dependencies within the time series to be captured and used for classification.

After transforming the time series into high-dimensional vectors, SVDD-TS applies the SVDD algorithm to find the smallest sphere that contains all the vectors, which defines the support region. The distance between each time series and the center of the support region is then used as a measure of similarity or anomaly score, with smaller distances indicating that the time series is more similar to the training data. This implementation was started from scratch as it is based on SVM but due to time constraints, this implementation could not be completed.

SVDD-TS is an effective algorithm for time series anomaly detection and classification problems, especially those involving non-linear or non-stationary data. It is also robust and can handle missing or incomplete data, making it a versatile approach for time series analysis.

IX. Future Scope

As this implementation is not exhaustive research on time series classification, the following can be explored further.

- The SVDD for time series[VIII] can be explored to see its effectiveness on time series classification.
- Transfer Learning can be used as mentioned in [13] and [14].

Acknowledgment

I am extremely grateful to Prof. Mahesh Varanasi and Altug Karakurt for their unwavering support and guidance through the concepts of Machine Learning and for inspiring me to pursue this topic for the paper. I cannot thank my peers enough for their invaluable help and feedback during the course of this paper.

References

[1] http://www.timeseriesclassification.com/description.php?Dataset=ECG5000
[2] Bagnall, A., Lines, J., Bostrom, A. et al. The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. Data Min Knowl Disc 31, 606–660 (2017). https://doi.org/10.1007/s10618-016-0483-9
[3] https://en.wikipedia.org/wiki/Electrocardiography
[4] https://scikit-learn.org/stable/
[5] https://keras.io/api/
[6] https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21
[7] https://python-heart-rate-analysis-toolkit.readthedocs.io/en/latest/heartpy.heartpy.html
[8] https://machinelearningmastery.com/random-forest-for-time-series-forecasting/
[9] https://www.analyticsvidhya.com/blog/2021/06/random-forest-for-time-series-forecasting/
[10] https://doi.org/10.48550/arXiv.1611.06455
[11] M. G. Baydogan, G. Runger and E. Tuv, ”A Bag-of-Features Framework to Classify Time Series,” in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 35, no. 11, pp. 2796-2802, Nov. 2013, doi: 10.1109/TPAMI.2013.72.
[12] https://effectiveml.com/dynamic-time-warping-for-sequence-classification
[13] https://doi.org/10.48550/arXiv.1811.01533
[14] https://towardsdatascience.com/transfer-learning-for-time-series-forecasting-51f023bc159c