University of Malta



Applied Machine Learning

**EEG Eye State Prediction**

*Study Unit Code*ICS5110

*Students*Gabriel Sammut (184395M)

Nicholas Frendo (311186M)

Contents

[1. Introduction 3](#_Toc503510951)

[1.1. Explaining the Dataset 3](#_Toc503510952)

[2. Background 4](#_Toc503510953)

[2.1. Machine Learning Algorithms 4](#_Toc503510954)

[2.1.1. Support Vector Machine 4](#_Toc503510955)

[2.1.2. Recurrent Neural Network with Long Short-Term Memory 5](#_Toc503510956)

[2.2. Rescaling & Normalisation 6](#_Toc503510957)

[2.3. Overfitting 7](#_Toc503510958)

[2.4. Regularization 7](#_Toc503510959)

[2.5. Cross Validation 7](#_Toc503510960)

[2.6. Feature Extraction 8](#_Toc503510961)

[2.6.1. Dimensionality Reduction 8](#_Toc503510962)

[2.6.2. Feature Selection 8](#_Toc503510963)

[2.7. Quantitative Measures 9](#_Toc503510964)

[3. Experiments 10](#_Toc503510965)

[3.1. Methodology Used and Steps Taken 10](#_Toc503510966)

[3.2. Experiments Carried Out 11](#_Toc503510967)

[3.2.1. SVM 11](#_Toc503510968)

[3.2.2. Recurrent Neural Network with Long Short-Term Memory 15](#_Toc503510969)

[4. Conclusions 17](#_Toc503510970)

[4.1. Experiment Conclusion 17](#_Toc503510971)

[4.1.1. SVM 17](#_Toc503510972)

[4.1.2. Recurrent Neural Network with Long Short-Term Memory 19](#_Toc503510973)

[4.2. Final Remarks and Future Improvements 20](#_Toc503510974)

[5. References 20](#_Toc503510975)

# 1. Introduction

Electroencephalography (EEG) eye state classification is a widely active research domain, with many studies concerning EEG signals being pursued in varied fields. The results of these studies are imperative not only in the medical field, but also tend to be significant in day to day tasks, as they tend to be attributed with human cognitive state classification. Amongst different domains, EEG eye state classification has been successfully attributed to areas of infant sleep-waking state identification, driving drowsiness detection, epileptic seizure detection, mood disorder classification and other areas. Generally EEG eye state data usually categorizes as a continuous type of time series readings, in which a number of machine learning and statistical approaches can be implemented to handle these classification problems. [1]

## Explaining the Dataset

The EEG Eye State data set [1] will serve as the basis of this scientific investigation. The dataset consists of data from one continuous EEG measurement carried out on a single subject. The duration of the measurement totals up to 117 seconds, during which time the eye state of the subject was detected via a camera during the EEG measurement and appended later manually to the data after the video frames were analysed. All values are recorded in chronological order, with the first measured value at the top of the data. The accumulated data was captured using the Emotiv EEG Neuroheadset.[2]

The dataset is considered Multivariate, Sequential and Time-Series in nature. The dataset consists of a total of 14980 data points, including outliers, made up from a total of fifteen attributes consisting of fourteen input features and a single output label denoting ‘1’ for eye-closed and ‘0’ for eye-open states. Hence, we can categorize this as a binary classification problem of detecting the patient’s eye state, depending on the current EEG readings obtained from all input features. The respective dataset attributes are detailed below, as well as the naming alias referred to for each variable:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Feature Name** | **Alias Name** | **Feature Mean** | **Feature Standard Deviation** | **Feature Minimum Value** | **Feature Maximum Value** |
| AF3 | var1(t) | 4321.92 | 2849.09 | 1030.77 | 309231 |
| F7 | var2(t) | 4009.77 | 2871.75 | 2830.77 | 7804.62 |
| F3 | var3(t) | 4264.02 | 2980.13 | 1040 | 6880.51 |
| FC | var4(t) | 4164.95 | 3100.84 | 2453.33 | 642564 |
| T7 | var5(t) | 4341.74 | 2830.66 | 2089.74 | 6474.36 |
| P7 | var6(t) | 4644.02 | 2966.25 | 2768.21 | 362564 |
| O1 | var7(t) | 4110.40 | 2960.32 | 2086.15 | 567179 |
| O2 | var8(t) | 4616.06 | 2683.99 | 4567.18 | 7264.1 |
| P8 | var9(t) | 4218.83 | 2851.31 | 1357.95 | 265641 |
| T8 | var10(t) | 4231.32 | 2945.14 | 1816.41 | 6674.36 |
| FC | var11(t) | 4202.46 | 3212.45 | 3273.33 | 6823.08 |
| F4 | var12(t) | 4279.23 | 3570.21 | 2257.95 | 7002.56 |
| F8 | var13(t) | 4615.21 | 4073.37 | 86.67 | 152308 |
| AF4 | var14(t) | 4416.44 | 4714.63 | 1366.15 | 715897 |
| eyeDetection | var15(t) | 0.45 | 0.50 | 0 | 1 |

Table 1 – Dataset Description

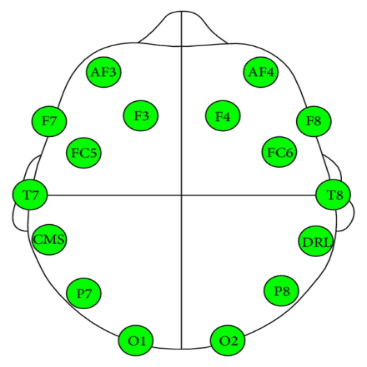


Figure 1 - Emotiv EEG Neuroheadset Sensors

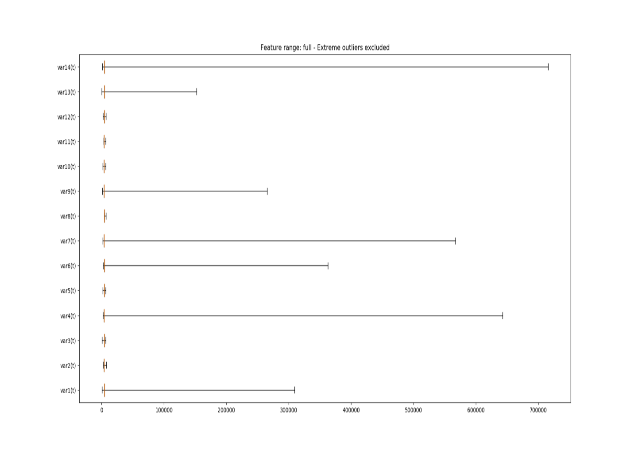


Figure 2 - Box Plot Including Outliers

# Background

The following consists of an overview regarding different machine learning techniques. Not only limited to the classification techniques used in this report, the following section also entails related machine learning techniques and observations which will be applied to the analysis and evaluation of the EEG Eye State data set.

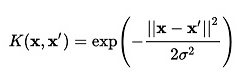
## Machine Learning Algorithms

### Support Vector Machine

Support Vector Machines (SVM) constitute part of a supervised subset of machine learning algorithms useful for regression and classification problems, and are commonly coined as maximal-margin classifiers. Similar to other discriminant machine learning algorithms, SVMs make use of a hyperplane (a separator in n dimensions) to classify data into different categories, by maximising the margins (the distance between hyperplane and nearest data points) from the nearest training data points, referred to as support vectors. By maximizing the margins, the hyperplane separator is capable of better generalizing new (unseen feature vectors which were not used for the training of the supervised model) instances, hence the more accurate the prediction outcome of the SVM. This, combined with other below mentioned hyperparameters make the SVM machine learning model a very attractive one in varied domains, in particular that of EEG eye state detection, a considerate problem which can be tackled as a classification problem of eye state open, or closed [5].

SVMs were originally designed for binary classification problems [6], however different subsets have been successfully adopted for multi-classification problems, including 1-against-all, 1-against-1 and all-together approaches [7]. In general, SVMs are considered robust for handling of linearly and non-linearly separated problems, capable of adopting a number of kernels with which to establish an optimum hyperplane. This feat makes the SVM algorithm a practical model to apply in real world problems, due to handling of noise and ‘dirty’ data. In addition to the input feature vector, a set of coefficients/weights (also referred to as slack variables) are introduced to the model in order to establish the margin with a degree of ‘wiggle room’ in each dimension. Together with the multiplied sum of all data instances and the relative attributed weight, a tuning parameter C is introduced to the SVM model. This regularization parameter dictates the margin size to be established during the training phase of the algorithm, where smaller margins are considered for larger values of C, and vice-versa. Another important hyper parameter for optimum SVM applicability is the gamma value attributed to the machine learning model. The gamma parameter defines the amount of influence established by the support vectors in relation to the hyperplane separator. A high value of gamma considers only those points nearest to the hyperplane. Conversely, the lower the value of gamma, the further away points are considered when establishing margins from the separation line.

The core of the SVM classifier lies within the well establishment of the hyperplane which is implemented in turn in the form of a kernel, transforming the problem into non/linear algebra implementations. Amongst the most common of Kernels, SVMs can make use of Linear Kernels, Polynomial Kernels (further signified using the ‘degree’ hyperparameter), sigmoid Kernel, radial basis function (RBF) and custom kernels. Such kernels can be rephrased as an expression using the inner product between two given observations, rather than the data instances by themselves. This is referred to as the ‘Kernel Trick’ and is used to project data onto a higher dimension so as to achieve better separability by the hyperplane. Due to the heavy usage of the RBF kernel for the duration of this experiment, the radial kernel is formally expressed as [5]:



* x = new input
* xi = support vector
* || = Euclidean distance
* 1/(2 \* θ^2) = gamma

The RBF kernel function is always expected to return a numeric value between 0 and 1, thus in classification problems, such values are compared to a numeric threshold so as to identify the input features with a particular category. If x and xi are equal, RBF gives a value of 1 (Euclidean distance measured between both points), whereas in the case of x and xi being distant from each other, the RBF value becomes lower (yet never quite reaching 0).

### Recurrent Neural Network with Long Short-Term Memory

The vanilla Recurrent Neural Network (RNN) suffers from unstable backpropagation error when attempting to model time series datasets containing long temporal information. Namely, vanilla RNNs training error signals flowing backward tend to either vanish or blow up, resulting in unstable weight oscillations. The RNN with Long Short-Term Memory (LSTM) was specifically designed to handle such a shortcoming. This was achieved through the introduction of a explicitly designed network component, termed Memory Cell [9, 10].

As depicted by Figure 3, a vanilla LSTM memory cell may be split into three main components, namely the input component, the memory component and the output component, each having a dedicated neural network associated with. At its core, the memory cell contains a loop connection that stores a memory cell state over time, responsible for learning temporal information in access of 1000 discrete time steps [9]. The challenge is to ensure that during training, the memory component learns the input patterns of interest across time, whilst ignoring any irrelevant signals. Similarly, given that LSTM memory cells can be stacked on top of each other, we do not want the memory cell state to perturb the state of subsequent stacked cells.

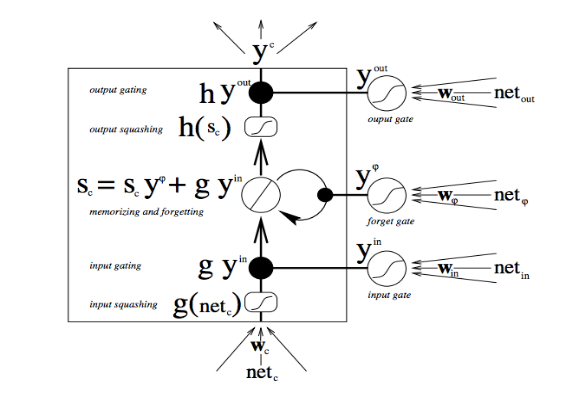


Figure 3 – Vanilla LSTM Memory Cell Block [101]

Output Component

Memory Component

Input Component

In order to meet such requirements, the memory cell has an input multiplicative gate that serves as a valve mechanism. Through the associated neural network, the input gate learns to let through to the memory component, the input patterns of interest, whilst blocking the rest. The resulting output of the input gate neural network is squashed through a sigmoid function resulting in weighting values in the range [0,1], thus achieving the desired regulation through a multiplication operation. Similarly, in the output gate, the associated neural network learns which signals to let through to the subsequent blocks.

Therefore, by means of the memory cell state input and output protection, a memory cell is said to act like a *Constant Error Carrousel (CEC).* This is the cardinal advantage over vanilla RNNs, given that a constant error flow is ensured in both feedforward and backpropagation directions, hence solving the vanishing and unstable gradient issues.

## Rescaling & Normalisation

The following are machine learning and statistical data pre-processing techniques used to ‘clean’ the dataset so as to achieve a uniform scale. They are considered to be pre-processing methods due to being applied before the training of the predictor black box, as well as being performed on any new data instances which is used to make new predictions.

* **Rescaling** – The process used to convert feature data into values of a particular scale, commonly in the range of [0,1]. The process scales each and every individual instance as follows:

Where:  
 – Normalized value  
 – Highest feature value  
 – Lowest feature value  
 – Input data feature value

This technique allows a certain measure of robustness over the data due to the safe assumption that the data is constrained between a minimum and maximum value. This technique is generally applied to machine learning algorithms which use a form of distance measure for its generalizing function. [3]

* **Standardization** – The attributed data is known to “standardize” around a mean of 0, with a standard deviation of 1, whilst taking the shape of a Gaussian distribution. It is generally applicable and well used with regression models. This technique is also referred to as a Z-Score, calculated as follows:

Where:

z – Standardized score  
x – Input data feature value  
μ – Feature mean  
 – Sample standard deviation

* **Normalization** – The technique is used to rescale individual data instances of feature vectors in order to achieve unit norm. It is carried out by dividing the feature vector by the total number of input variables, as long as the total summation of the data points in the feature vector amount to 1. [3]

Where:

– Input feature vector  
 – Vector magnitude  
 – Normalized vector

## Overfitting

Overfitting is a frequently encountered problem in machine learning, whereby the model is said to have *memorized* the training data, rather than learning the underlying problem at hand. In a practical scenario, training data will contain noise, which, if closely modelled will degrade prediction performance on unseen data.

One of the symptoms attributed to overfitting is that of observing a substantial discrepancy in model prediction performance when applied to the training dataset and the testing dataset. Namely, if the model has overfit the training dataset, one would expect to see high prediction performance on the training dataset and low prediction performance on the unseen testing dataset. Conversely, if a model has generalized the underlying problem, it is expected that the model lets through more incorrect predictions on the training dataset whilst boosting performance on the unseen testing dataset.

Overfitting is mainly caused by incorrect model hyperparameter configuration. Additionally, having a high feature-count-to-data-point-count ratio tends to increase the likelihood of overfitting the model.

## Regularization

Regularization is a strategy attempting to minimize model overfitting. Various regularization strategies exist which are closely related to the machine learning algorithm under consideration. For example, in the case of neural networks, one such regularization strategy is the dropout method. Namely, dropout being a model hyperparameter, specifying the percentage of neurons within a given network that should be removed from the network during training. Hence, fine tuning the dropout parameter may prevent the network from closely modelling the training dataset.

## Cross Validation

Cross validation is a machine learning technique involving the withholding of a fraction of the entire dataset for validation purposes whilst training a classifier. The data which is not used to train the classifier is eventually used to evaluate the trained model, allowing the user to validate the model on data which has never been consumed by the machine learning black box. If a model is validated on the same data which it has been trained on, this could lead to a good performance when acting on the training data but poor generalization on new unseen data (as would be in the case of cross validation).

It is important to note, that validation and testing do not refer to the same process in machine learning evaluation. It is therefore important to clarify the following terms:

* **Training Dataset** – The sample of data which is used to feed the machine learning model.
* **Validation Dataset** – The sample of data reserved for the evaluation of the trained model, in order to gauge initial accuracy of the model whilst tuning model hyperparameters.
* **Test Dataset** – The sample of data used to provide an unbiased evaluation of the final model on the training set it has been trained and validated on.

There are multiple types of cross validation techniques, amongst which the following are the most commonly used:

* **Hold Out cross validation** – Mostgeneric method, in which a percentage of the data is held out for validation purposes. It is normal to take split values at around 60/40, 70/30, 80/20, were each first value denotes the training sample percentage and the remaining is reserved for validation.
* **Random sub-sampling** – Randomly splits the dataset into K data segments, and repeats the algorithm classification for each of the folds. The final error estimate is calculated as an average of the individual segments.
* **Leave one out cross validation** – A subset case of the K-Fold approach, this method places K=1, and for each experiment uses N-1 examples for training and the remaining one for testing.
* **K-Fold cross validation** – Holds out equal K-fold partitions in the dataset reserving them for validating the dataset (similar to Random sub-sampling, but without the degree of randomness). Similarly, this method equates the average of the error estimate of each of the K-folds.

## Feature Extraction

### Dimensionality Reduction

The art of machine learning is influenced heavily with the design of appropriate data representations, with better performance achieved using derived information from the original features. Dimensionality reduction is a pre-processing technique where a dataset of N-dimensions has the number of dimensions decreased to be less vast, whilst still retaining the information it used to contain. Data can span in the hundreds of thousands of features, a fact especially true for gene selection for microarray data in the bio informatics domain and text categorization where documents are represented as ‘bag-of-word-models’ (the size of the vocabulary depends on the word frequency counts), the process of ‘feature compression’ or better referred to as dimensionality reduction, is useful for several reasons [4]:

* Is able to achieve better reconstruction of data with respect to model prediction performance.
* Better visualization of data, as the number of dimensions decrease.
* Has the potential to convert non-tractable problems into solvable ones, due to the reduction of dimension for particular problems (faster model training and predictions).

Building a feature representation of existing features can be a very application specific task. There are however a number of generic feature construction methods, involving a combination of algebraic linear methods (PCA,LDA), as well as other sophisticated linear ones including but not limited to wavelet transforms, kernel convolutions and the application of simple product or monomial functions to subsets of variables. Other feature reduction methods make use of non-supervised techniques in an attempt to express the original input features into a fewer, newly constructed set of features. Such a technique makes use of algorithms such as K-means, Hierarchical clustering and Self-Organized Maps, replacing similar input features by their cluster centroid equivalent, based on a given metric.

### 2.6.2. Feature Selection

In contrast to dimensionality reduction, feature selection is a process where in it does not concern itself with the preservation of less important features (where importance is a degree of variance in combination to the output label), but instead seeks to eliminate that data which is considered “redundant” with respect to the output label. This enforces the methodology to seek out features with low interdependency between each other, whilst assigning importance to those features with a high level of correlation with respect to labels. The most common advantages of feature selection include:

* Improvement of model prediction accuracy by limiting to input features which are highly correlated to output label (curse of dimensionality).
* Faster and more cost-effective predictors (requires less time to train the optimum model).
* Better data visualization and interpretability.
* Reducing the measurement and storage requirements required by features.

There are varied feature selection methodologies, each applicable in their own right, the most prominent being methods categorized under the following categories:

* **Filter Methods** – Statistical methodologies useful with respect to single factor analysis, in order to gauge the amount of correlation between a single feature and an output label in isolation. Considered to be a principal and auxiliary selection mechanism, these statistical methods are considered as pre-processing methods, making them irrelevant to the model fit on the data. Common techniques include the calculation of correlation criteria (eg: Pearson’s Correlation) for linear relationships between an input feature and the output class. For non-linear data, it is recommended to default to Information Theoretic Ranking (a measure of dependency between the density of 2 variables), or other methods such as a Chi squared test.
* **Wrapper Methods** – Involves the usage of prediction performance of a given machine learning algorithm to assess the relative usefulness of variable subsets. In contrast to filter methods, wrapper methods are capable of evaluating combinations of features rather than features in isolation. The machine learning method used to standardize the feature importance is irrelevant to the actual model to be trained, thus we can once more we can consider the final predictor as a black box during the feature selection process. Common approaches to wrapper feature ranking takes the form of greedy lookup searches:
  + Forward selection – Start with individual variables and progressively incorporate promising variables into subsets.
  + Backward elimination – Start with all variables and progressively drop the least promising variables.
  + Hybrid searches – A combination of the above, progressively dropping and incorporating feature subsets.
* **Embedded Methods** – These methods for generalized linear predictors can be explained as adding a penalty against low correlated features to reduce the degree of overfitting or variance. Such methods guide their search by estimating changes in the objective function value incurred by making moves in variable subspace, assigning low weighting schemes to features of low correlation, and higher weighting to features of higher label intra-dependency. Unlike filter and wrapper methodologies, embedded methods cannot be described as a pre-processing technique, due to the fact that all features are incorporated into the model, which arbitrarily assigns weighting to different features.

## Quantitative Measures

The following metrics will be considered for evaluation and estimation of how well the chosen classifiers are acting:

* **Accuracy** – The closeness of a measured value to a particular standard (generally the validation/testing data).
* **Precision** – The ratio of correctly predicted true positives in comparison to all observations of the class, both true positives and false positives. Precision can be defined as:
* **Recall** – The ratio of correctly predicted true positives in comparison to all observations of the actual class. Recall can be defined as:
* **F1-Score** –The weighted average of the precision and recall metric, a score in which both the false positives and false negatives are taken into account. It gives equal importance to precision and recall, and can be defined as:

# Experiments

The following code libraries were utilised for the duration of the following experiments:

* Scikit Learn
* TensorFlow
* Keras

## Methodology Used and Steps Taken

The EEG dataset contains an inherent temporal relation between past inputs & outputs and the current classified output at time *t.* In addition to that, Figure 4 continues to show that when the output label changes state, the new state is retained for a number of consecutive samples. Thus, valuable information aiding in better modelling the problem is contained across time.

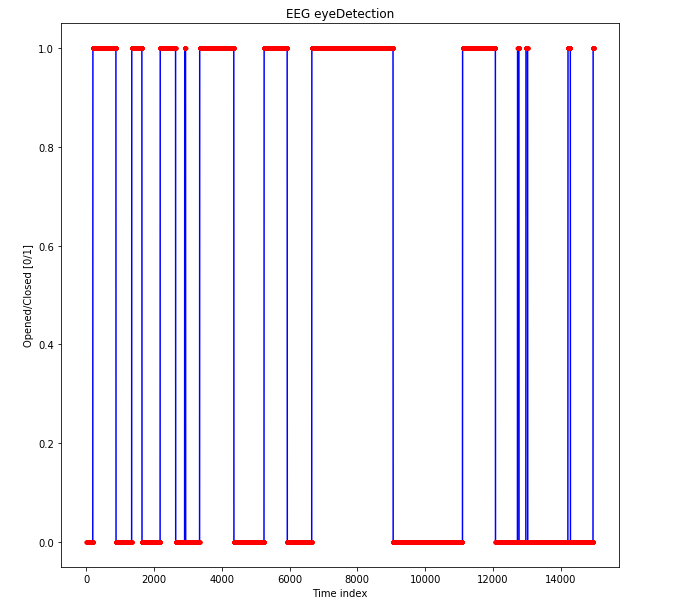


Figure 4 - Output Label

To preserve and take advantage of the temporal nature of the EEG dataset, machine learning techniques such as K-Fold and random dataset sampling could not be applied. In addition, in order to apply supervised machine learning algorithms, the time series dataset required additional preparation steps. Namely, the time series problem needed to be re-framed as a supervised learning problem, achieved by introducing a lag. As depicted by Figure 5, by shifting the dataset in time, we create new features out of past input and output features. These new features will then act as input features to the supervised machine learning model. Therefore, for a dataset having one output feature, the new input feature count in relation to the lag count, will increase as defined by the following relationship [8]:

Where:

NFC – New feature count

OFC – Old feature count

LV – Lag value

F-0-0

F-0-1

F-0-2

F-0-3

F-0-4

F-0-5

F-1-0

F-1-1

F-1-2

F-1-3

F-1-4

F-1-5

O-0

O-1

O-2

O-3

O-4

O-5

F-0-0

F-0-1

F-0-2

F-0-3

F-0-4

F-1-0

F-1-1

F-1-2

F-1-3

F-1-4

O-0

O-1

O-2

O-3

O-4

F-0-1

F-0-2

F-0-3

F-0-4

F-0-5

F-1-1

F-1-2

F-1-3

F-1-4

F-1-5

O-1

O-2

O-3

O-4

O-5

Input

Features

Output

Feature

Input

Features

Output

Feature

Lag-Zero

Lag-One

Figure 5 - Time-Series to Supervised Conversion

F-X-Y: Represents input feature X at timestep Y

O-Z: Represents the output classifier at timestep Z

Following the re-framing of the time series dataset, the newly created dataset was split in two sets, namely the training dataset (≈67%) and the testing dataset (≈33%). In addition, the training data was further divided into a model training dataset and a validation dataset. During experimentation and model tuning, the model training set was used to train the model for a given configuration setting. For each setting, model prediction performance was computed on the validation dataset, whilst the training dataset was kept unused throughout the model tuning stage.

Entire Dataset

Training

Set

Test

Set

Model

Training

Set

Validation

Dataset

Figure 6 - Train-Validation-Test Split

Once all model parameter combinations of interest were exhausted through a grid search approach, the resulting prediction performances were sorted by their respective F-Score value. The model setting leading to the best F-Score value was then re-tested using the unseen testing dataset.

## Experiments Carried Out

### SVM

The first step consisted of detection and exclusion of outlier data in both training and testing data subsets,

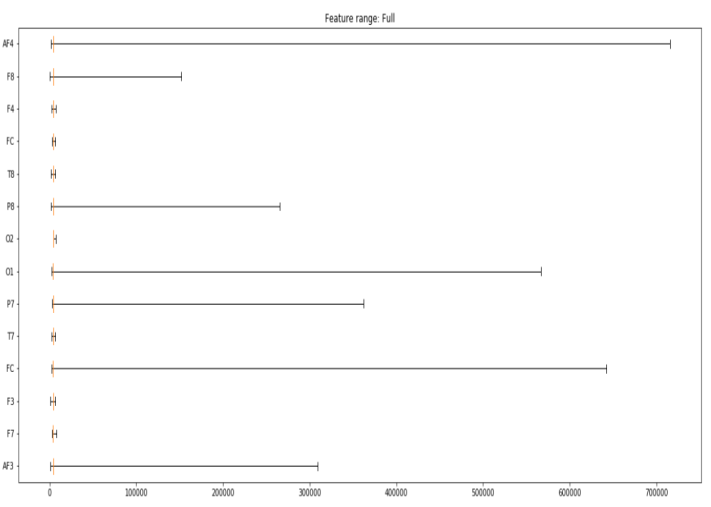
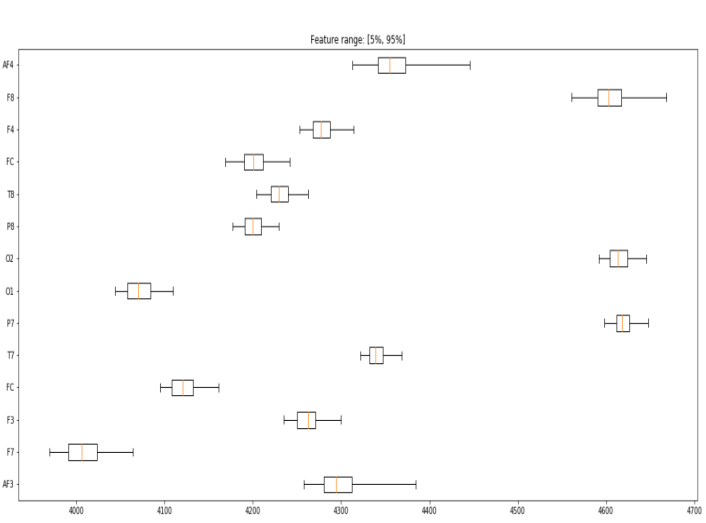
in order to reduce external noise which could potentially influence the final result. The following box plot figures show the effect and skewness of the data before and after the outliers were removed.

Figure 8 - Box Plot (5% - 95%) – Outliers Excluded

Figure 7 - Box Plot - Raw Data

Due to the time series aspect of the EEG Eye State data, the dataset was converted using a time shifting function. It should be noted that for the remainder of this experiment subsection concerning SVMs, only lag values of 0 were considered. Therefore, the dataset was not subjected to any time shifts, and only the default values presented in the original dataset were applied. This decision was made by testing different time shifts in the training dataset, and it was found that lag 0 features gave the best validation results, as indicated in table 2.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Ranking Method** | **Lag Value** | **Feature Count** | **Accuracy** | **F-Measure** | **CV Split** | **Normalization** |
| Random Forest | 0 | 13 | 89.79% | 89.70% | 80/20 | L2 Norm |
| Gradient Boosting | 0 | 14 | 81.21% | 81.05% | 80/20 | L2 Norm |
| Random Forest | 1 | 22 | 89.25% | 89.18% | 80/20 | L2 Norm |
| Gradient Boosting | 1 | 24 | 82.07% | 81.94% | 80/20 | L2 Norm |
| Random Forest | 2 | 39 | 89.08% | 89.01% | 80/20 | L2 Norm |
| Gradient Boosting | 2 | 32 | 82.07% | 81.95% | 80/20 | L2 Norm |
| Random Forest | 5 | 72 | 88.48% | 88.38% | 80/20 | L2 Norm |
| Gradient Boosting | 5 | 71 | 82.30% | 82.18% | 80/20 | L2 Norm |

Table 2 – Lag Relevance

This claim is further reinforced by an autocorrelation plot, stating that the nearer the lag values are to 0, the stronger the correlation to the output label in the time series.

A range of data transformation techniques was considered as denoted below:

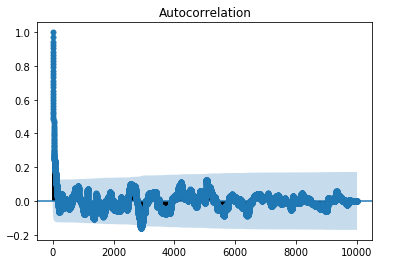


Figure 9 - Dataset Autocorrelation Plot (All 14980 instances)

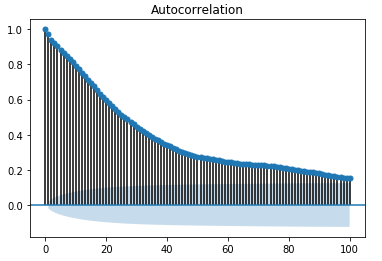


Figure 10 - Dataset Autocorrelation Plot (First 100 instances)

* Normalization (L1 & L2)
* Min-Max Feature Scaling

After testing different combination of the above feature transformation techniques, the Min-Max normalization approach was adopted after evaluating all scoring metrics, for show of greater combined accuracy and F-Measure throughout recurrent tests. It should be noted, that the output label was not transformed and left as the original raw counterpart due to its Boolean nature (eye opened/closed).

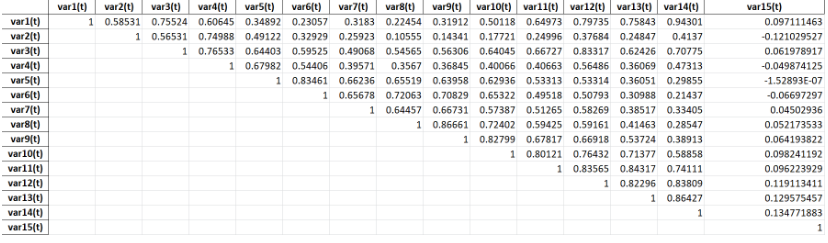
Following this, several feature selection techniques was utilised to identify those features with greatest correlation to the output label. After correlation values were drafted, it was evident that the dataset is subject to a high level of inter-feature dependency, and a low level of feature-label correlation.

Figure 11 - Pearson Correlation Matrix Results

To solidify further the above results, as well as to rely on other non-linear dependant metrics, a mutual information scoring evaluation was carried out on each of the features as denoted in table 3.

|  |  |
| --- | --- |
| **Variable Name** | **Mutual Information** |
| var1(t) | 0.068001631969438509 |
| var14(t) | 0.061312983255733783 |
| var6(t) | 0.056815681032305346 |
| var13(t) | 0.056309625332207881 |
| var7(t) | 0.04282873920644046 |
| var11(t) | 0.040537269025248061 |
| var2(t) | 0.03461132683140055 |
| var12(t) | 0.0283772790896249 |
| var9(t) | 0.02320555796089771 |
| var3(t) | 0.020713608072078921 |
| var4(t) | 0.02026262824227585 |
| var10(t) | 0.020162418824560796 |
| var8(t) | 0.019820144087840025 |
| var5(t) | 0.0086360493779071828 |

Table 3 – Mutual Information Ranking

A blurred relationship can be established between feature ranking of the Pearson correlation matrix and mutual information metrics. Although a rough indication of which features correlate the highest with the output label can already be noticed, it is still unclear which features are to be optimally used with the output label. In addition, such evaluations have been made individually of other neighbour features, therefor such information dependency is lost. Therefore, further tests are required to be conducted in the shape of wrapper importance ranking.

|  |  |  |
| --- | --- | --- |
| **Variable Name** | **Feature Importance (Random Forest)** | **Feature Importance (Gradient Boosting)** |
| Var6 | 0.113461 | 0.134323 |
| var7(t) | 0.099624 | 0.099543 |
| var2(t) | 0.094403 | 0.136585 |
| var13(t) | 0.088216 | 0.116963 |
| var1(t) | 0.079337 | 0.059840 |
| var14(t) | 0.076207 | 0.081333 |
| var4(t) | 0.070021 | 0.079329 |
| var11(t) | 0.068164 | 0.046642 |
| var12(t) | 0.060754 | 0.056994 |
| var3(t) | 0.056630 | 0.034189 |
| var5(t) | 0.052414 | 0.059171 |
| var10(t) | 0.051833 | 0.050292 |
| var8(t) | 0.050861 | 0.038432 |
| var9(t) | 0.038076 | 0.006365 |

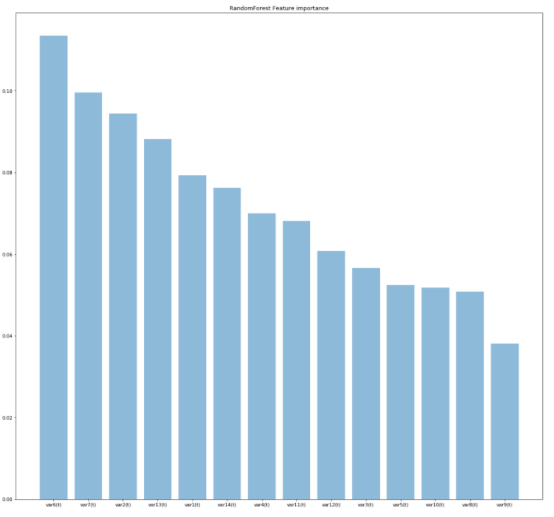


Figure 12 - Random Forest Feature Importance

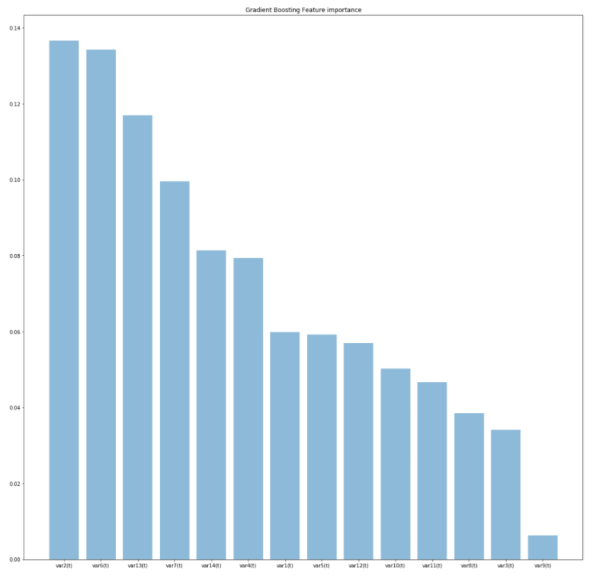


Table 4 – Wrapper Feature Ranking

Any evaluation carried out so far has seen features being evaluated independently of other features in relation to the output label. This is a common downside of filter methods, due to the isolation with which they gauge feature importance.   
  
To alleviate this drawback, a wrapper feature analysis methodology was implemented by using a backward elimination approach. By considering all feature combinations and gauging the scoring metrics produced by a classifier, and stripping away one feature at a time, we can get a measure of confidence for the optimum number and which features to use going forward. These results are tabulated in table 5, using a combination of two different algorithm ranking methods for better result confidence.

Figure 13 - Gradient Boosting Feature Importance

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Feature Ranking Algorithm** | **Feature Index with max score** | **Accuracy** | **Precision** | **Recall** | **F-Measure** | **Cross Validation Split** |
| Random Forest | 12 | 91.75% | 91.81% | 91.75% | 91.76% | 80/20 |
| Gradient Boosting | 13 | 84.65% | 84.64% | 84.65% | 84.55% | 80/20 |
| Random Forest | 13 | 91.86% | 91.99% | 91.86% | 91.88% | 70/30 |
| Gradient Boosting | 13 | 84.33% | 84.33% | 84.33% | 84.25% | 70/30 |
| Random Forest | 13 | 90.12% | 90.28% | 90.12% | 90.15% | 60/40 |
| Gradient Boosting | 14 | 84.6% | 84.57% | 84.6% | 84.53% | 60/40 |

Table 5 – Backward Elimination Feature Ranking

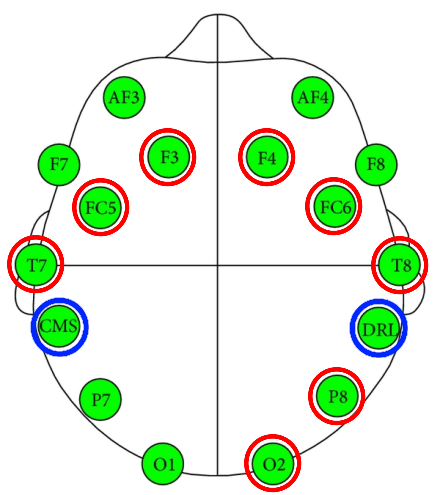
By adopting both Random Forest and Gradient boosting suggestions and applying them separately on different test runs using different and varied training/validation splits on an SVM classifier, it was found that greater accuracy was achieved for validation instances, when aiming for the 12-14 feature count mark. However, the trained model did not generalize well with the recommended features provided by the wrapper methods when applied to the test file (refer to next chapter for final results). Therefore, the number of features was further reduced until the top 6 features were retained (as highlighted by the Random Forest wrapper), at the cost of decreasing the validation scoring metrics but at the benefit of better model generalization (and therefore, higher test file scoring metrics).

Figure 14 - Blue: Non-existent features  
Red: Redacted features  
Non-Circled: Retained features

### Recurrent Neural Network with Long Short-Term Memory

As discussed in the previous sections, during experimentation, the goal was to determine the model configuration that results in the best F-Score as computed on the validation dataset. Following is a list of all the model parameters considered:

* **Min-Max Normalization**

Model training was performed both with and without input feature normalization

* **Lag Count**

When converting the dataset into a supervised problem, two distinct lag values were considered, namely:

* + Lag Count 1: Total input feature count of 19
  + Lag Count 10: Total input feature count of 164
* **Epoch**

The epoch count represents the number of consecutive training iterations performed. Two values considered, namely 50 and 1000 epochs.

* **Batch Size**

Represents the number of training data points used per epoch. Two values were considered, namely 50 and 500.

* **LSTM Stack**

As discussed previously, LSTM memory cells may be stacked on top of each other. During experimentation, two network architectures were considered, namely:

* + One memory cell
  + Five stacked memory cells

In both cases, the final layer output vector was condensed and squashed into one single neuron, resulting in the binary classification output.

* **LSTM Layer Output Dimension**

Represents the output vector dimension of each LSTM layer. Two values were considered, namely 5 and 50.

* **Loss Function**

Represents the method used in measuring the error between the expected labelled output and the predicted output during training. Mean absolute error function was considered.

* **Optimizer Function**

Represents the mechanism used in order to update the network weights during training. Adam optimization algorithm was considered.

The model configurations considered during testing are tabulated in Table 6 - RNN with LSTM Results. For each model setting, the associated validation data *F-Score* as well as the model training time is included. Note, the results were sorted by the *F-Score* and the training time respectively.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **TestID** | **F-Score** | **Train Time [sec]** | **Outliers Norm** | **Lag Count** | **Epoch** | **Batch Size** | **LSTM Stack** | **LSTM Layer OutDim** |
| 1 | 99.9499989104% | 346.928 | Yes | 10 | 1000 | 500 | 5 | 50 |
| 2 | 99.9499989104% | 939.754 | 50 | 5 |
| 3 | 99.9499989104% | 984.552 | 1 |
| 4 | 99.9499989104% | 1014.594 | 50 |
| 5 | 99.9499989104% | 1038.468 | 5 |
| 6 | 99.9499986839% | 2.821 | 1 | 50 | 500 | 1 | 5 |
| 7 | 99.9499986839% | 4.724 | 50 |
| 8 | 99.9499986839% | 14.317 | 50 | 5 |
| 9 | 99.9499986839% | 17.482 | 50 |
| 10 | 99.9499986839% | 36.144 | 1000 | 500 | 5 | 5 |
| 11 | 99.9499986839% | 64.184 | 50 |
| 12 | 99.9499986839% | 278.932 | 50 | 1 | 5 |
| 13 | 99.9499986839% | 290.546 | 5 |
| 14 | 99.9499986839% | 310.918 | 50 |
| 15 | 99.9499986839% | 321.213 | 1 |
| 16 | 99.8999955915% | 46.907 | 10 | 50 |
| 17 | 99.8499899676% | 15.519 | 500 |
| 18 | 99.8499899676% | 123.690 | 1000 | 5 | 5 |
| 19 | 99.7999819635% | 48.056 | 50 | 50 | 1 |
| 20 | 99.7499715035% | 7.764 | 500 |
| 21 | 36.3186701605% | 14.651 | No | 1 | 50 |
| 22 | 36.3186701605% | 17.242 | 50 |
| 23 | 36.3186701605% | 277.617 | 1000 | 5 |
| 24 | 36.3186701605% | 289.225 | 5 |
| 25 | 36.3186701605% | 318.350 | 1 | 50 |
| 26 | 36.3186701605% | 322.166 | 5 |
| 27 | 35.8060446781% | 47.524 | 10 | 50 | 1 | 5 |
| 28 | 35.8060446781% | 52.860 | 50 |
| 29 | 35.8060446781% | 917.253 | 1000 | 5 |
| 30 | 35.8060446781% | 941.024 | 5 |
| 31 | 35.8060446781% | 1089.231 | 50 |
| 32 | 35.8060446781% | 1150.837 | 1 |

Table 6 - RNN with LSTM Results

# Conclusions

## Experiment Conclusion

### SVM

This section is dedicated at interpreting SVM performance with comparison to RFC results.

A number of machine learning classifiers were applied in order satisfy the question as to which algorithm was the most optimum for the classification problem at hand. Each algorithm was tested several times using a grid search approach to allow the testing and fulfilment of not only the most suitable hyperparameters applicable to the respective algorithm, but also to gauge the highest optimum values for cross validation splits. The highest results which were generated on the validation data subset were then applied on the test file data subset in order to ensure that the model managed to generalize well, and to avoid overfitting. Table 7 consists of the test sample space within which each algorithm was searched for in order to achieve the highest optimum score metric, followed immediately by table 8 which pertains the best results applied to each machine learning algorithm.

|  |  |  |
| --- | --- | --- |
| **Algorithm** | **Parameter Name** | **Parameter Space** |
| Logistic Regression | CV Split | 80/20, 70/30, 60/40 |
| Logistic Regression | C | 1-15 |
| Logistic Regression | Solver | Liblinear, sag, saga, newton-cg |
| Logistic Regression | Normalization | L1,L2, Min-Max |
| Logistic Regression | Feature Selection | Single Feature Analysis (Pearson and MI scorings, Random Forest Ranking, Gradient Boosting Ranking), Backward elimination (Random Forest, Gradient Boosting) |
| Linear Discriminant Analysis | CV Split | 80/20, 70/30, 60/40 |
| Linear Discriminant Analysis | Normalization | L1,L2, Min-Max |
| Linear Discriminant Analysis | Feature Selection | Single Feature Analysis (Pearson and MI scorings, Random Forest Ranking, Gradient Boosting Ranking), Backward elimination (Random Forest, Gradient Boosting) |
| Random Forest Classification | CV Split | 80/20, 70/30, 60/40 |
| Random Forest Classification | N\_estimators | 10,100,1000 |
| Random Forest Classification | Max\_features | 6, ‘sqrt’, ‘log2’ |
| Random Forest Classification | Criterion | Gini, Entropy |
| Random Forest Classification | Normalization | L1,L2, Min-Max |
| Random Forest Classification | Feature Selection | Single Feature Analysis (Pearson and MI scorings, Random Forest Ranking, Gradient Boosting Ranking), Backward elimination (Random Forest, Gradient Boosting) |
| SVM | CV Split | 80/20, 70/30, 60/40 |
| SVM | Kernel | Rbf, linear, sigmoid |
| SVM | C | 1-15 |
| SVM | gamma | 0.001, 0.01, 0.1, 1-30 |
| SVM | Normalization | L1,L2, Min-Max |
| SVM | Feature Selection | Single Feature Analysis (Pearson and MI scorings, Random Forest Ranking, Gradient Boosting Ranking), Backward elimination (Random Forest, Gradient Boosting, SVM feature ranking) |

Table 7 – Hyperparameter / Parameter Search Space

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Classifier** | **CV Split** | **Hyperparameters** | **Features Excluded (var N)** | **Validation Accuracy** | **Validation Precision** | **Validation Recall** | **Validation F-Measure** |
| Logistic Regression | 60/40 | C=13, Solver=’saga’ | N = 9, 8, 10, 5, 3, 12, 11, 4 | 62.63% | 62.25% | 62.63% | 60.38% |
| Linear Discriminant  Analysis | 60/40 |  | N = 9, 8, 10, 5, 3, 12, 11, 4 | 62.33% | 62.02% | 62.33% | 59.71% |
| Random Forest Classification | 60/40 | N\_estimators=1000, max\_features=sqrt,  Criterion=’entropy’ | N=9, 8 | 91.18% | 91.21% | 91.18% | 91.14% |
| SVM | 60/40 | Kernel=’rbf’, C=8, gamma=13 | N = 9, 8, 10, 5, 3, 12, 11, 4 | 95.08% | 95.10% | 95.08% | 95.06% |

Table 8 – Top Performant Machine Learning Results

The highest results obtained after trying different algorithm and parameter sets were recorded above. Considering only the top two performing classifiers, each algorithm was then applied to the test data set so as to evaluate performance on new unseen data. The following results indicate a level of overfitting on both algorithms. By further exclusion of features being input into both classifiers and further hyper parameter tuning, the model was tuned to better generalize on new unseen data, with a final F-Measure result of 81.53% / 78.52% for validation and testing subsets respectively.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Classifier** | **Parameters / Configuration** | **Features Excluded (var N)** | **Validation Accuracy** | **Validation F-Measure** | **Test Accuracy** | **Test F-Measure** |
| Random Forest Classification | 60/40, N\_estimators=1000, max\_features=sqrt,  Criterion=’entropy’ | N=9, 8 | 91.18% | 91.21% | 35.64% | 36.68% |
| SVM | 60/40, Kernel=’rbf’, C=8, gamma=13 | N=9, 8 | 95.08% | 95.06% | 40.53% | 42.95% |
| Random Forest Classification | 60/40, N\_estimators=1000, max\_features=sqrt,  Criterion=’entropy’ | N = 9, 8, 10, 5, 3, 12, 11, 4 | 83.625% | 83.5403827751% | 65.1999196303% | 67.3871843183% |
| SVM | 70/30, Kernel=’rbf’, C=5, gamma=10 | N = 9, 8, 10, 5, 3, 12, 11, 4 | 81.70% | 81.53% | 78.78% | 78.52% |

Table 9 – Test File Evaluation

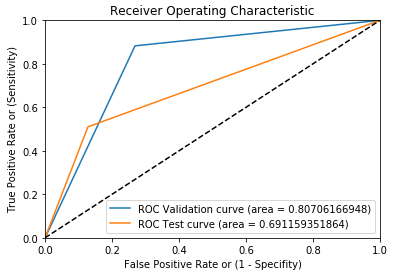
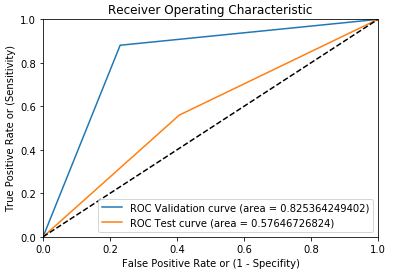


Figure 15 - SVM Validation/Testing Performance

Figure 16 - RFC Validation/Testing Performance

### Recurrent Neural Network with Long Short-Term Memory

This section is dedicated at interpreting the RNN with LSTM results obtained during experimentation.

* **Normalization**

Normalization is the factor which effects model convergence the most. Namely, there is a drop of ≈63.43% in the F-Score between the worst performing test case with normalization applied (Test ID: 20) against the best performing test case without normalization being applied (Test ID: 21).

* **LSTM layer output dimension**

As depicted by Figure 17, by varying only the *LSTM output dimension* across test cases with similar model configuration, it may be observed that better performance was observed for an output dimension of 50.

* **LSTM Stacked layers**

As depicted by Figure 18, by varying the *LSTM stacked layer count* across test cases with similar model configuration, it may be observed that better performance was observed for network architectures containing 5 stacked LSTM cells.

* **Lag Count**

As depicted by Figure 19, counter intuitively, it was observed that better F-Score performance metrics were observed for a lag count of 1 across similar test cases.

* **Final results**

As observed in Table 6, tests with IDs 1 – 5, all resulted in the same *F-Score* performance on the validation dataset. The model configuration used in Test ID 1 was preferred over the other equally performing cases due to the relatively efficient training execution, attributed to the training batch size. A training execution time improvement of ≈63% was observed against the test case with the second-best execution time (Test ID 2).

Finally, the resulting *F-Score* performance on the unseen testing dataset for Test ID 1, was found to be equal to 99.8187403733%.

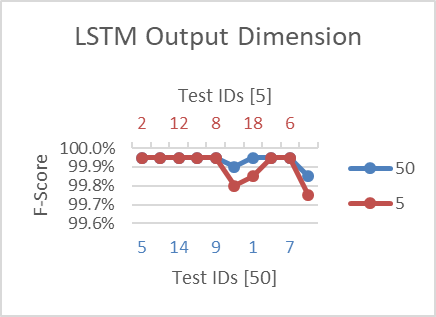


Figure 17 - LSTM Output Dimension Variation

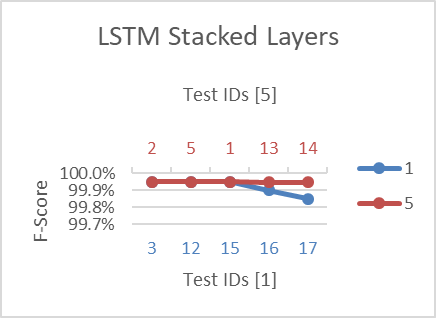


Figure 18 - LSTM Stacked Layers Variation

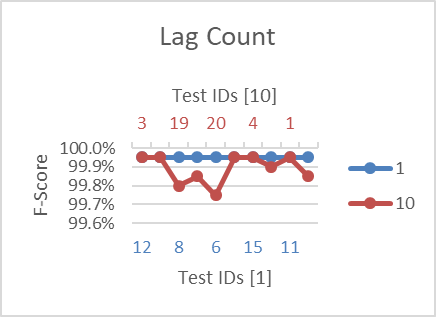


Figure 19 - Lag Count Variation

## Final Remarks and Future Improvements

A limiting factor during the EEG Eye State data set experimentation was the lack of data available for training each respective machine learning model, especially so after splitting the main data entries into their respective training and testing subsets. In addition, the dataset itself was not well generalized since all readings were accumulated from a single subject. A model trained on data that pertained to more than one subject would have made a better fit on the classifier to be able to generalize future readings from different patients. Another constraint consisted of the time series restriction on K-Fold testing. Due to the pattern and continuous state of the readings, K-Fold was not considered a beneficial cross validation method. Further to this, a substitute could have been applied, where in data is cross validated on different sized groupings of the time series and always incorporating values from position one up till the varying cross validation limit. Such an approach would have retained the time series attributes from position 0, but would have allowed the variation of the cross validation split enough to mimic a K-Fold approach.

# 5. References

[1] T. Wing, S. Guan, K. L. Man and T. O. Ting, “EEG Eye State Identification Using Incremental Attribute Learning with Time-Series Classification”, 2014

[2] UCI: Machine Learning Repository, “EEG Database Data Set”. [Online]. Available: <https://archive.ics.uci.edu/ml/datasets/eeg+database>

[3] M. Sanders, “Scaling? Normalizing/? Standardizing? Which do I use when?”, *Data Reality Blog*. 2016 [Online]. Available: <http://datareality.blogspot.com.mt/2016/11/scaling-normalizing-standardizing-which.html>

[4] I. Guyon and A. Elisseeff, “An Introduction to Variable and Feature Selection”, 2003.

[5] F. Lotte, M. Congedo, A. Lécuyer, F. Lamarche and B. Arnaldi, “A review of classification algorithms for EEG-based brain-computer interfaces”, 2007.

[6] V. N. Vapnik, *Statistical Learning Theory*. New York, 1998.

[7] N. Y. Liang, P. Saratchandran, G. B. Huang and N. Sundararajan, “Classification of mental tasks from EEG signals using extreme learning machines”, 2006.

[8] J. Brownlee, “How to Convert a Time Series to a Supervised Learning Problem in Python”, *Machine Learning Mastery Blog.* 2014 [Online]. Available: <https://machinelearningmastery.com/convert-time-series-supervised-learning-problem-python/>

[9] S. Hochreiter and J. Schmidhuber, “Long Short-Term Memory”, 1997.

[10] F. A. Gers, J. Schmidhuber and F. Cummins, “Learning to Forget: Continual Prediction with LSTM”, 1999.

[11] W. Zaremba, I. Sutskever and O. Vinyals, “Recurrent Neural Network Regularization”, 2015.