

APRIL 27, 2021

Total words count: 4223



MACHINE LEARNING - COIY065H7-AAA. COURSEWORK REPORT

STUDENT NAME: ROHAN KHANOLKAR
MSC ADVANCED COMPUTING TECHNOLOGIES

Emails: rkhano04@student.bbk.ac.uk
khanolkar.rohan@gmail.com

Academic Declaration

I have read and understood the sections of plagiarism in the College Policy on assessment offences and confirm that the work is my own, with the work of others clearly acknowledged. I give my permission to submit my report to the plagiarism testing database that the College is using and test it using plagiarism detection software, search engines or meta-searching software.

Table of Contents

1. Introduction	2
2. Methodology.....	3
3. Flow of the coursework.....	4
4. Building Neural Network Model and WAME Optimizer.....	7
4.1. Optimizers:	7
4.2. Initialization	7
4.3. Epochs & Batches	7
4.4. Regularization of Neural Network.....	7
4.5. WAME optimizer	8
5. Experiments and finding:.....	8
6. Conclusion.....	13
7. References.....	14
8. Appendix 1:.....	16
9. Appendix 2:.....	20

1. Introduction

This course work is a learning experience for using neural networks on real-world data which helps us explore new avenues of the dataset. The aim of the coursework is to understand and implement 'Weight-wise adaptive learning rates with moving average estimators' (WAME) [1] on UCI 'Adult data-set' [2] which has 15 features that are categorical as well as continuous. This coursework requires to use neural network algorithms with a customize optimizer (WAME) with its changing learning rates. The learning rate adaption was first introduced by Hinton, G. et. al. [30] in 2012 which said that the model computational and accuracy can be improved.

The UCI adult dataset [2] is a part of 1994 census database which was tasked to make prediction on individual's yearly income, and check if the income is above 50K. This data-set has five files namely 'adult.data' is the training data-set, 'adult.test' is the testing data-set, 'adult.name' consist of narration of each feature and its name along with description of various algorithms and their error rates that were used on this data set, 'old.adult.names' consist of previous names given to the feature before transferring them on this data-set along with similar algorithms and their error rates which are mentioned in 'adult.name' file. The file named index consist of the date when these files were created along with the number of characters inside each file. This dataset denotes relationship between weights and the demographic characteristics of people can be similar. The characteristic of the features in this dataset are mentioned below:

Sr. No.	Feature Name	Type of Feature	Narration
1	Age	Continuous	Ranging from 36 to 86 years.
2	Work class	Categorical	Defines where a person works.
3	FNLWGT	Continuous	Defines weighted mean according to their demographic.
4	Education	Categorical	Defines a person's education background.
5	Education-num	Continuous	Number allotted to a person according to their education background.
6	Marital status	Categorical	Defines marital status of a person.
7	Occupation	Categorical	Defines an occupation of a person.

8	Relationship	Categorical	Defines if a person is married/unmarried or if they have any children.
9	Race	Categorical	Defines race of a person (black/white etc.).
10	Sex	Categorical	Male/female
11	Capital-Gain	Continuous	Capital gained by a person.
12	Capital-Loss	Continuous	Capital loss by a person.
13	Hours-per-week	Continuous	Defines number of hours worked by a person per week.
14	Native country	Categorical	Defines a person ethnic native background
15	Class	Categorical (Target)	Defines if a person earns more than or less than 50K per year.

Six out of fifteen features in the dataset are continuous, eight features are categorical, and one feature is a target variable. There are many research papers that cite this data-set amongst which Kohavi, R. [3] is a relevant paper which shows a hybrid method used as supervised learning which consist of combining 'Naive-Bayes Classifier' along with 'Decision Tree Classifier' which was called as 'NBTree' in this method the leaves of the decision tree consist of 'Naive-Bayesian Classifier' and the nodes of the decision tree consist of splits which are univariate. These models were tested on various dataset and proved to be a highly accurate classifier which improved its constituents which help to scaleup its accuracy.

2. Methodology

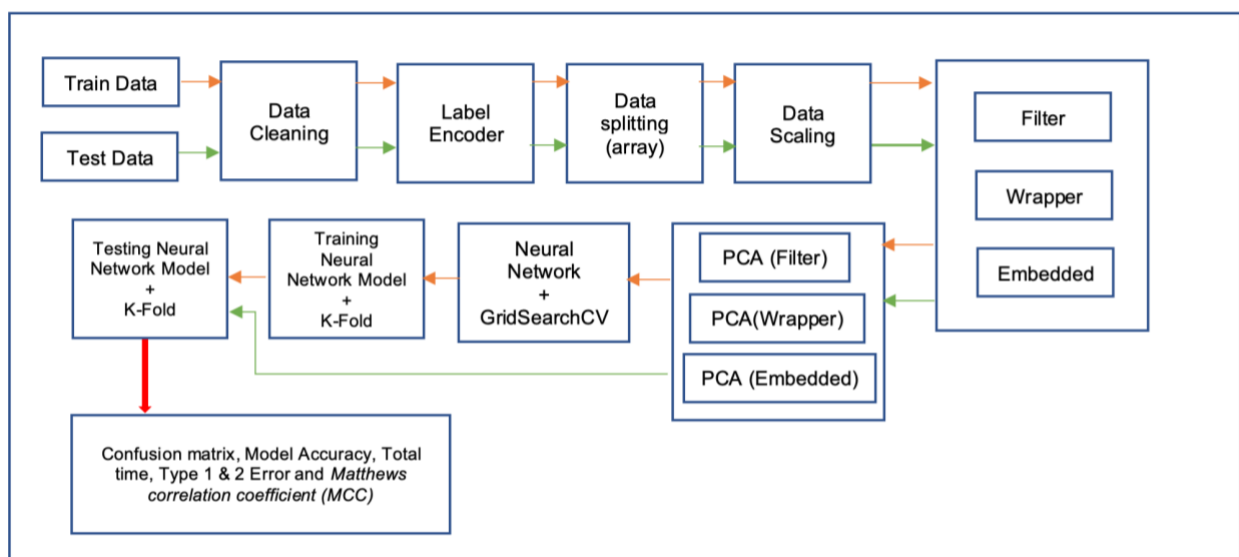
In the paper provided as a reference in this course work by Mosca, A. [1] which introduces us to a new concept of adaptive learning rates which are propagated weight-wise that is derived from 'Stochastic Gradient Descent'. The WAME algorithm is hybrid optimizer that consist of known propagation methods like 'Resilient propagation' (Rprop) and 'Root me square propagation' (RMSprop) along with 'Adaptive Movement Estimation' (Adam) which helps in bias correction. This hybrid optimizer helps a neural network model to perform better along with improving the time taken for model training using the same architecture.

The WAME optimizer works [1] by first checking if the sign of the previous gradient is multiplied along with the current gradient more than or less than zero. If it is more than zero, then the time taken for per-weight acceleration factor is calculated by multiplying

the previous gradient that is assigned with the ' η^+ ' (zeta) and it will be considered as minimum acceleration factor. If it is more than zero, then the per-weight acceleration factor is calculated by multiplying the previous gradient the assigned with the ' η^- ' (zeta) and it will be consider as maximum acceleration factor. This will be feed to calculate the value of zeta (which is a product previous gradient plus one minus alpha) multiplied by output of the time taken for per-weight acceleration factor. After this, the divisor for each gradient value is calculated as the previous gradient value plus product for one minus alpha and square weighted error. Further, the non-linear weighted effect is calculated as the product of learning rate along with weighted error and inverse of the gradient values. After this step, the weighted preceding gradient is calculated as the sum of weights and the non-linear weight. All these values are then updated as input to minimize the time to run and not compromising on accuracy of the neural network.

According to the paper by Mosca, A.[1] this WAME algorithm was performed on convolutional neural network for deep learning classification that could be similar to a human level performance. Convolutional Neural Network (CNN) [5] are highly accurate for image classification and image recognition like a visual perception of human to recognize image pattern. For this course work we have been allotted a non-image dataset (i.e. UCI Adult Dataset) wherein we have to predict the income of the test cases are above 50K or not. Thus, for this course work we have decided to proceed with a regular neural network algorithm which has an input layer of 12 nodes (neurons), a hidden layer of 10 nodes (neurons) and an output layer of 1 node (neurons). The reason for choosing only 1 hidden layer is that to reduce the complexity of relationship between the input and the desired output [4], another reason for keeping minimum layers is to [4] avoid memorization of dataset (training) which can lead to generalization of dataset that can proved to be of no use for any other dataset.

3. Flow of the coursework



To begin with we have used python language to code this course work, Google Colab (colaboratory) notebook [6] which is a web-based IDE used to execute arbitrary machine learning python codes. The major benefit of using Google Colab for this coursework is that we do not need to load an environment to run 'keras' like other IDE such as 'Jupyter Notebook' .

We first open the train and test dataset in a notepad to check if there are any null values or any replacement of null values, we found that the missing values were replaced by ' ?' in the original train (adult.data) and test (adult.test) datasets. After this analysis, we load the train and test datasets by using 'read_csv' a 'pandas' function [7] and denoted the headers of the dataset as 'names' and ' ?' as 'na_values' (which identifies the null values in the dataset and also considers ' ?' as a null value). We now check the shape of the dataset and note that 'adult.data' (training dataset) has 32561 test instances along with 15 features and 'adult.test' has 16282 test instances along with 15 features. To drop the null instances from the data set we used 'dropna()' a 'pandas data frame' function [8] which deleted all the rows from the data set which had 'blank/null' or ' ?' in them. This drop function left us with 30162 test instances (2399 rows were dropped due to null values) along with 15 features and 15060 test instances (1222 rows were dropped due to null values) along with 15 features. Now we check if there are any null/blank or ' ?' in the dataset after using 'dropna()' function, we used 'value_counts()' a 'pandas.series' function [9] where we get to see all the unique values of the dataset and their counts. If there was any blank/? value it would display it as a unique value. It was noted that we had not found any blank/? values after using the 'dropna()' function.

After confirming all the null values are eliminated, we now proceed to convert the categorical feature into categorical numeric features by using 'LabelEncoder' from 'sklearn.preprocessing' [10], this will identify all the unique values of the categorical features and assign them with unique numbers which can be helpful for us in identifying best features for our model. After using this label encoder we will get numerical features and a target feature (which consist of value '0' for income less than 50K and '1' for income more than 50K). In this process we have created a hybrid code which consist of 'apply' function from 'pandas.DataFrame' library [11] and label encoder function along with 'fit_transform' function from 'sklearn' library [12]. This hybrid function performs the fit and transform function along with the label encoder and applying it on the dataset (separately on training and testing data). After performing tis hybrid code we now have the entire dataset which consist of continuous data and numerical categorical data.

After the label encoding process on both train and test data, we will divide the features of both the datasets into two dataframes (arrays) and named it as 'X' (for 14 features of the training dataset) and 'Y' (for target class of the training dataset) along with 'X_t' (for 14 features of the testing dataset) and 'Y_t' (for target class of the testing dataset) which will help us to perform further actions on the dataset. We have now prepared the data ready to be scaled by using the 'MinMaxScaler' from 'sklearn.preprocessing' library [13] which will estimate each feature and perform scale and translate action on every individual feature of both the datasets and bring them down to a variable of 0 and 1. This process allows the data to be normalized for further actions to be taken on the dataset.

After scaling the datasets, we must now perform feature selection methods to select and retain the most important features of the datasets. After the feature selection process (where features are sub-grouped) we pass it through feature extraction process where we would create a new feature from the selected sub-grouped features in both training and testing datasets. In this coursework we have performed three parallel methods (filter, wrapper and embedded) and their outputs to verify which outperforms each other. In the filter method [14] the features are selected based on their characteristic such as correlation with another dependent variable in the dataset. The wrapper method [14] is known to be computationally costly along with being decumbent to over fitting but have proved to be performing better on scaled data. This method has a predictive algorithm model which uses to select the best features from the dataset. The embedded method [14] is known to be less computationally costly than the above methods and it has minimum over fitting compared to other methods, this method is encapsulated in the learning process while building a model for feature selection.

We are using 'SelectKBest' from 'sklearn.feature_selection' library [15] as a filter method along with the score function called as 'chi2' from 'sklearn.feature_selection' library [16] to which will help us select features in the dataset using there k-highest score (a feature of SelectKBest) along with chi-squared statics which removed the feature that do not depend on the class (irrelevant classification feature). In this filter method we assigned the 'SelectKBest' model to choose 9 features out of 14 features from the test and train dataframes.

For wrapper method we are using Recursive Feature Elimination (RFE), 'RFE' from 'sklearn.feature_selection' library [17] which is an external estimators that ranks the features of a given dataset by assigning weights to it, RFE works by considering smaller sets of feature and obtain the importance of each feature by training the estimator that prune the desired number of features. We combine this RFE with a logistic regression solver called as 'liblinear' from 'sklearn.linear_model.LogisticRegression' library [18] which is used for handling sparse and dense dataset, this converts the data into float to help the RFE model select the best features. In this wrapper method we assigned the 'RFE' model to choose 9 features out of 14 features from the test and train data-frames.

For embedded method we are using 'ExtraTreeClassifier' from 'sklearn.ensemble' library [19] which creates subsamples of datasets and fits decision trees randomly to average out the best nodes which hints the classifier to select a certain feature from the dataset. In this embedded method we assigned the 'ExtraTreeClassifier' model to choose 9 features out of 14 features from the test and train data-frames, but we found that this model returned with only 7 selected features as it would have predicted only 7 nodes of the decision trees to be relevant with each other.

We now advance to feature extraction process where we would create a new feature from the selected sub-grouped features in both training and testing datasets. For this feature extraction process we will use Principle Component Analysis (PCA), using 'PCA' from 'sklearn.decomposition' library [20] which reduces the selected features to create new features using dimensionality reduction by adopting Singular Value Decomposition (SVD) that lowers the dimensional space of the data to merge with dependent features. We have performed separate feature extraction process for filter,

wrapper and embedded outputs for training data as well as testing data and extracted 5 features from each method output for training and testing data respectively.

We have now completed the data pre-processing stage that included loading data, cleaning data, scaling data, feature selection and feature extraction. We will now proceed to build a neural network model to train and predict on the extracted PCA features of filter, wrapper, embedded outputs.

4. Building Neural Network Model and WAME Optimizer

The details of building neural network model and WAME (optimizer) algorithm is explained in the methodology section of this coursework. We will further discuss what actions were taken on the neural network model and how the adaptive learning rate was considered to train and test the model. To continue from the neural network model, I have reused some codes which were taught to me in Applied Machine Learning module, we have used Tensorflow Keras library [21] to build this neural network configuration with an activation function as 'relu' [23] (This function brings any negative value to 0) for the hidden layers and 'sigmoid' [24] (it is used when we want to predict the output as 0 or 1) for the output layer. This neural network model is now hybridized with 'GridSearchCV' from 'sklearn.model_selection' library [22] to search the best parameter for the neural network model. The parameter used for finding the best fit for this neural network are:

4.1. Optimizers: These are used to reduce the loss in neural network by configuring/changing attributes like learning rate and weights [23]. We have used 'adam' (which runs on the mean of moment of gradient of the first and second) [1], 'RMSprop' (which uses mini batches derived from Rprop that derives the error looking at the weight) [1]. We have also added the 'WAME' optimizer in this list.

4.2. Initialization ('inits'): Every optimizer needs a point to start in the given neural network space and the 'inits' function gives that point to start, this actually prepares the network's weight [24]. We have used 'uniform' (which denotes uniform distribution of weights) and 'glorot_uniform' (which uses the sample with the limit of the distribution) [24].

4.3. Epochs & Batches: These are the hyperparameter tuning parameters, the batches are the number of samples a model has to work before it updates the internal parameter of model and epochs are the number of times the model has to work through the whole given dataset [25]. We have used epoch as [5, 7, 10] and batches as [20, 30, 40] for this model.

4.4. Regularization of Neural Network: They are techniques used to address overfitting along with the selection of feature in a neural network. These are of two types, L1 'Lasso Regression' (which squares the magnitude of the coefficients of the loss function) and L2 'Ridge regression' (which adds the loss function with absolute value magnitude) [26]. We had used L2 on our neural network model which did not give us desired accuracy, so we decided not to use this function. The accuracy results can be found in Appendix 1B of this course work.

4.5. WAME optimizer: We have used the existing code found on the website of Woolcott, D. [27] and made simple modification to it fit in our model, which is explained in the code file attached with this coursework. In this optimizer created by Woolcott we noticed that he had used the 'backend as k' from the 'keras' tensorflow library [28] which is used for generating unique names for various repetitive layers of the code. He has used 'switch' instead of using a 'for' loop and also used 'clip' instead of a 'if else statement' to create a various loop as mentioned in the algorithms mentioned in the paper by Mosca [1].

5. Experiments and finding:

The codes of this non-regularized neural network are attached as code files (named "ML_CW_13199041.ipynb") are working, and the Appendix 2 explains how to run the codes in Google Colab. Apart from this, I have also attached the code file which consist of regularized neural network and their output (named "Regu_Keras_ML_CW_13199041.ipynb") which had poor performance as compared to the non-regularized neural network, so it not been discussed in this section.

The steps used for running the codes are:

a. We first fit the training data on the keras neural network model with the following parameters for the PCA output of filter wrapper and embedded to see the accuracy and the parameter it uses to get that accuracy.

Parameters used:

```
optimizers = ['adam', 'rmsprop']
inits = ['uniform', 'glorot_uniform']
epochs = [5, 7, 10]
batches = [20, 30, 40]
loss = tf.keras.losses.mean_squared_error
```

b. We first note the results of this run and reset the neural network model by restarting the kernel so that we can make sure the model has unlearned the dataset. The results obtained are:

For filter:

```
Best: 0.831212 using {'batch_size': 20, 'epochs': 10, 'init': 'uniform', 'optimizer': 'rmsprop'}
```

For wrapper:

```
Best: 0.838704 using {'batch_size': 20, 'epochs': 10, 'init': 'uniform', 'optimizer': 'adam'}
```

For Embedded:

```
Best: 0.832240 using {'batch_size': 20, 'epochs': 10, 'init': 'glorot_uniform', 'optimizer': 'adam'}
```

We can note from the above observations that the filter method uses the 'rmsprop' optimizer, where are both wrapper and embedded uses the optimizer 'adam'. Also we can note that the accuracy of all the three are similar in the range of 83%.

C. After resetting the neural network we feed it with the WAME optimizer with the following weights and learning rates. The weights assigned are as per given in the paper by Mosca [1]

	learning_rate	alpha	alpha_2	epsilon	decay	eta_plus	eta_minus	zeta_min	zeta_max	alpha_a
WAME1	0.001	0.9	0.999	1.00E-11	0	1.2	0.1	1.00E-02	1.00E+02	0.9
WAME2	0.0001	0.9	0.999	1.00E-11	0	1.2	0.1	1.00E-02	1.00E+02	0.9
WAME3	0.00001	0.9	0.999	1.00E-11	0	1.2	0.1	1.00E-02	1.00E+02	0.9
WAME4	0.000001	0.9	0.999	1.00E-11	0	1.2	0.1	1.00E-02	1.00E+02	0.9

and we fed it as the optimizer parameter in the neural network. Kindly note that we have created a separate model so that it does not learn the training dataset which was fitted on it earlier.

Parameters used:

```
optimizers = ['adam', 'rmsprop', 'WAME1', 'WAME2', 'WAME3', 'WAME4']
inits = ['uniform', 'glorot_uniform']
epochs = [5, 7, 10]
batches = [20, 30, 40]
```

The following results were obtained with the best results for filter, wrapper and embedded method outputs.

Filter tuned output with WAME:

```
Best: 0.831079 using {'batch_size': 20, 'epochs': 10, 'init': 'uniform', 'optimizer': 'WAME1'}
```

Wrapper tuned Output with WAME:

```
Best: 0.840992 using {'batch_size': 20, 'epochs': 10, 'init': 'glorot_uniform', 'optimizer': 'WAME4'}
```

Embedded tuned Output with WAME:

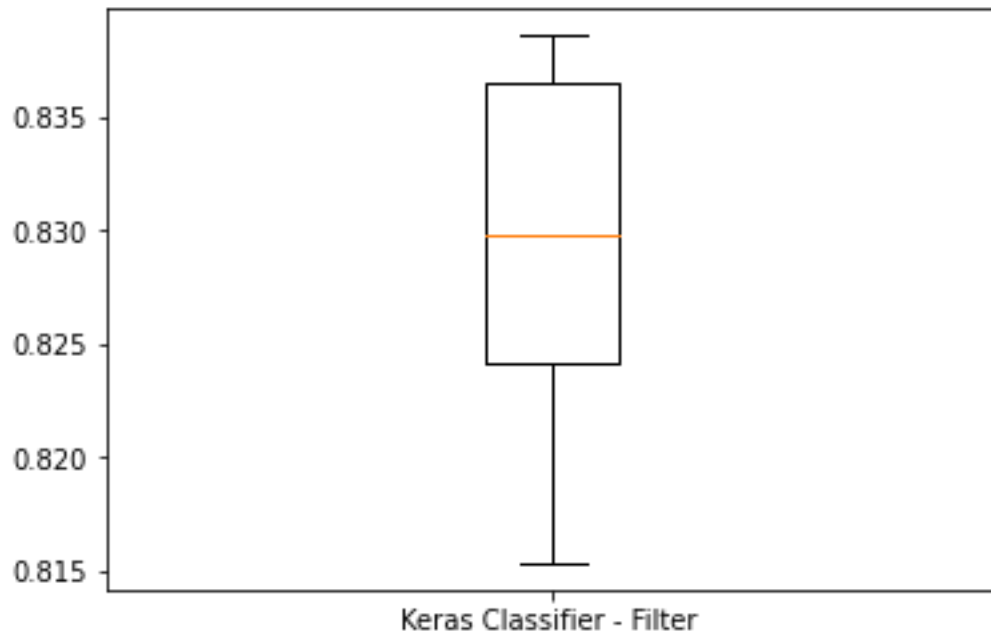
```
Best: 0.833267 using {'batch_size': 40, 'epochs': 10, 'init': 'uniform', 'optimizer': 'WAME3'}
```

After analyzing the output we can note that the GridSearchCV tuned the model with the best parameters and we can note that the filter method (tuned model output) selected 'WAME 1' optimizer which has a learning rate of 0.001, wrapper method (tuned model output) selected 'WAME 4' which has a learning rate of 0.000001 and embedded method (tuned model output) selected 'WAME3' which has a learning rate of 0.00001. On the other hand, the model accuracy are in the similar, range out of which the wrapper method gave a better accuracy out of the three.

d. We now create new Keras Classifier models with the tuned hyperparameters found in the above step and fit it with the training data PCA output. Note that this is done separately for filter, wrapper and embedded. The following is the output for the tuned trained models:

Filter Tuned Model:

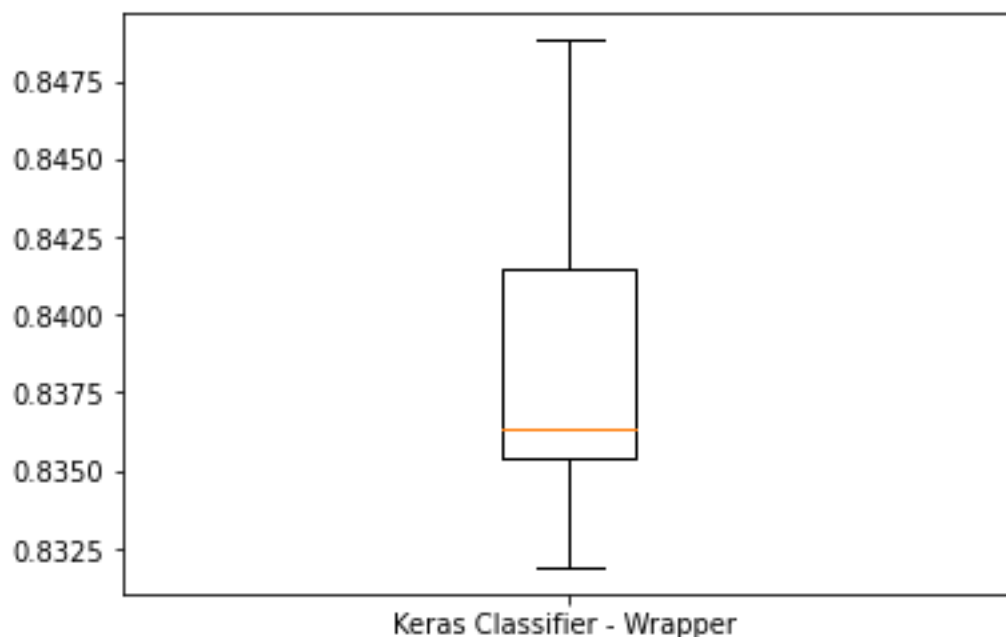
Keras Classifier - Filter Model : 0.829520 (0.007528)



We can note the average model accuracy is 82.95 % with a standard deviation of 0.0075. The highest accuracy achieved by this model is above 83.5% and the lowest is approximately 82%

Wrapper Tuned Model:

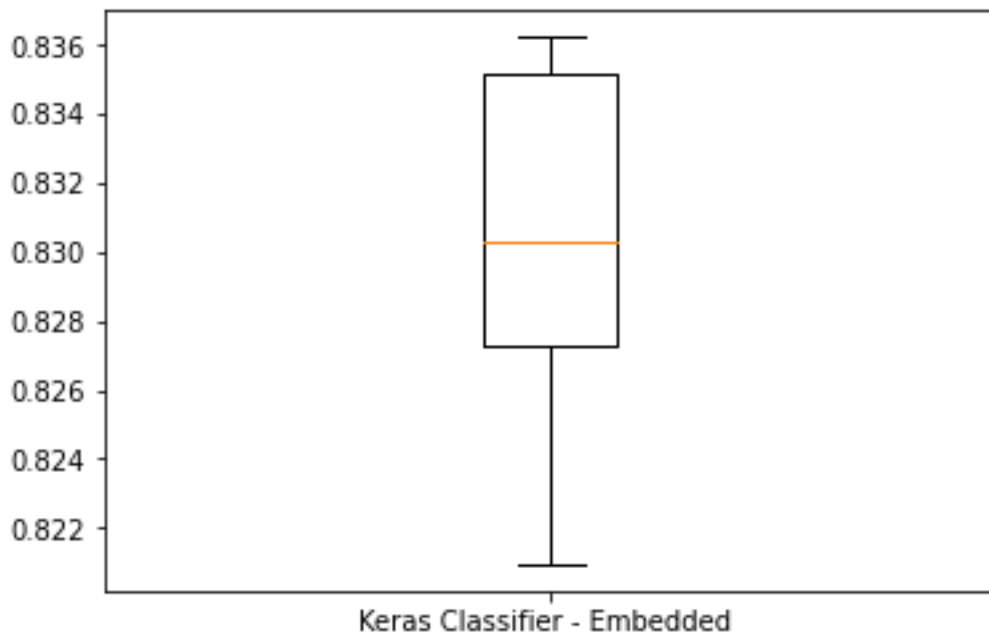
Keras Classifier - Wrapper Model : 0.838870 (0.005190)



We can note the average model accuracy is 83.88 % with a standard deviation of 0.0051. The highest accuracy achieved by this model is approximately 84.25% and the lowest is approximately 83.5%.

Embedded Tuned Model:

Keras Classifier - Embedded Model: 0.830349 (0.004915)



We can note the average model accuracy is 83.03 % with a standard deviation of 0.0049. The highest accuracy achieved by this model is approximately 83.6% and the lowest is approximately 82.8%

e. We now fit the trained model (which consist of the WAME optimizer) with the test data (PCA output of the test data from Filter, Wrapper and embedded method) and evaluate the model using the 'Kfold' cross validation from 'sklearn.model_selection' library [29] to validate the training data to the testing data and find the following parameters:

(i) Time taken to build model : This is the time take for building the model. This is considered as the WAME algorithm claims to improve the computational time.

(ii) Time taken to test model : This is the time taken for model to test. This is considered as the WAME algorithm claims to improve the computational time.

(iii) Model Accuracy : This is considered to check the accuracy and it is derived from the confusion matrix generated by the model after testing.

(iv) Model Error Rate : This is considered to check the inaccuracy of the model and it is the inverse of model accuracy.

(v) False Positive / Type 1 Error: This is derived from the confusion matrix where some test cases are shown as positive but they are actually negative.

(vi) False Negative / Type II Error : This is derived from the confusion matrix where some test cases are shown are negative but they are actually positive.

(vii) Matthews correlation coefficient (MCC): This is used to measure the quality of binary classification which is considered as balance measure that correlates the

observed and predicted class. In MCC if the output is closer +1 it means perfect prediction by the model, if is 0 it means it is predicting randomly and if the output is -1 it means the prediction is not agreeable to the desired output.

The Following output was achieved from the test dataset and we can see the following observations in the below table. The actual raw output can be seen in Appendix 1.

	<i>Filter Method Model</i>	<i>Wrapper Method Model</i>	<i>Embedded Method Model</i>
<i>Time to build model (sec)</i>	17.1888	16.7525	8.8648
<i>Time to test model (sec)</i>	0.4599	0.4636	0.2903
<i>Time elapsed (sec)</i>	17.6678	17.2323	9.1702
<i>Model Accuracy</i>	0.8300	0.8372	0.8276
<i>Model Error Rate</i>	0.1699	0.1627	0.1723
<i>False Positive/Type 1</i>	0.072	0.0802	0.0872
<i>False Negative / Type II Error</i>	0.4697	0.4159	0.4337
<i>Matthews correlation coefficient (MCC)</i>	0.5078	0.5381	0.51

From the above table we can see that the time taken to build and test the model is better with embedded method model which is 9.17 sec which is computationally faster compared to the filter and wrapper method. The model accuracy of Wrapper method model is slightly higher as compared to the filter and embedded method at 83.72%, one of the reasons could be PCA is also a wrapper method feature elimination technique. The Error rate is the inverse of the accuracy, so the Wrapper method will have lesser model error rate as compared to the other two. The false positive (type 1) rate is lowest in the filter method as compare to the other two, this mean that there are less number positive cases detected as negative compared to the other two methods. The false negative (type 2) are almost similar on all the three cases, this means that there is some weight component which need to be tweaked to further reduce this and improve the overall accuracy. The Matthews correlation coefficient are also similar for all the three methods at 0.5, this means that the model prediction are neither good prediction not random prediction. Overall, we can say that the Wrapper method gives better accuracy

6. Conclusion:

Although we have tested the neural network with the WAME algorithm (optimizer) on both regularized ('L2' sum of squared) and non-regularized architecture, we found that the non-regularized architecture performs slightly better than the regularized architecture (refer Appendix 1). Although, I am unsatisfied with the accuracies from these non-regularized neural network models, we can assume some more weight adoptions are necessary to further increase the model accuracy. These weight adoptions can be achieved by using the backpropagation technique of weight balancing method. This coursework has given a deep insight of how learning rates and network weight plays an important role in neural network learning.

7. References

- [1] A. Mosca and G. D. Magoulas, "Training convolutional networks with weight-wise adaptive learning rates," *Ucl.ac.be*. [Online]. Available: <https://www.elen.ucl.ac.be/Proceedings/esann/esannpdf/es2017-50.pdf>. [Accessed: 26-Apr-2021].
- [2] "UCI machine learning repository: Adult data set," *Uci.edu*. [Online]. Available: <https://archive.ics.uci.edu/ml/datasets/adult>. [Accessed: 26-Apr-2021].
- [3] R. Kohavi, "Scaling Up the Accuracy of Naive-Bayes Classifiers: a Decision-Tree Hybrid", *Stanford.edu*. [Online]. Available: <http://robotics.stanford.edu/~ronnyk/nbtree.pdf>. [Accessed: 26-Apr-2021].
- [4] "Training an artificial neural network - intro," *Solver.com*, 23-Mar-2012. [Online]. Available: <https://www.solver.com/training-artificial-neural-network-intro>. [Accessed: 26-Apr-2021].
- [5] K. Maladkar, "Overview of convolutional neural network in image classification," *Analyticsindiamag.com*, 25-Jan-2018. [Online]. Available: <https://analyticsindiamag.com/convolutional-neural-network-image-classification-overview/>. [Accessed: 26-Apr-2021].
- [6] "Colaboratory – Google," *Google.com*. [Online]. Available: <https://research.google.com/colaboratory/faq.html>. [Accessed: 26-Apr-2021].
- [7] "pandas.read_csv — pandas 1.2.4 documentation," *Pydata.org*. [Online]. Available: https://pandas.pydata.org/docs/reference/api/pandas.read_csv.html. [Accessed: 26-Apr-2021].
- [8] "pandas.DataFrame.dropna — pandas 1.2.4 documentation," *Pydata.org*. [Online]. Available: <https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.dropna.html>. [Accessed: 26-Apr-2021].
- [9] "pandas.Series.value_counts — pandas 1.2.4 documentation," *Pydata.org*. [Online]. Available: https://pandas.pydata.org/docs/reference/api/pandas.Series.value_counts.html. [Accessed: 26-Apr-2021].
- [10] "sklearn.preprocessing.LabelEncoder — scikit-learn 0.24.1 documentation," *Scikit-learn.org*. [Online]. Available: <https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html>. [Accessed: 26-Apr-2021].
- [11] "pandas.DataFrame.apply — pandas 1.2.4 documentation," *Pydata.org*. [Online]. Available: <https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.apply.html>. [Accessed: 26-Apr-2021].
- [12] G. Myrianthous, "fit() vs transform() vs fit_transform() in Python scikit-learn," *Geek Culture*, 14-Mar-2021. [Online]. Available: <https://medium.com/geekculture/fit-vs-transform-vs-fit-transform-in-python-scikit-learn-2623d5a691e3>. [Accessed: 26-Apr-2021].
- [13] "sklearn.preprocessing.MinMaxScaler — scikit-learn 0.24.1 documentation," *Scikit-learn.org*. [Online]. Available: <https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html>. [Accessed: 26-Apr-2021].
- [14] "Data Science," *Datasciencesmachinelearning.com*. [Online]. Available: <http://www.datasciencesmachinelearning.com/2019/10/feature-selection-filter-method-wrapper.html>. [Accessed: 26-Apr-2021].
- [15] "sklearn.feature_selection.SelectKBest — scikit-learn 0.24.1 documentation," *Scikit-learn.org*. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html. [Accessed: 26-Apr-2021].

- learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html. [Accessed: 26-Apr-2021].
- [16] "sklearn.feature_selection.chi2 — scikit-learn 0.24.1 documentation," *Scikit-learn.org*. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.chi2.html. [Accessed: 26-Apr-2021].
- [17] "sklearn.feature_selection.RFE — scikit-learn 0.24.1 documentation," *Scikit-learn.org*. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.RFE.html. [Accessed: 26-Apr-2021].
- [18] "sklearn.linear_model.LogisticRegression — scikit-learn 0.24.1 documentation," *Scikit-learn.org*. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html. [Accessed: 26-Apr-2021].
- [19] "sklearn.ensemble.ExtraTreesClassifier — scikit-learn 0.24.1 documentation," *Scikit-learn.org*. [Online]. Available: <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.ExtraTreesClassifier.html>. [Accessed: 26-Apr-2021].
- [20] "sklearn.decomposition.PCA — scikit-learn 0.24.1 documentation," *Scikit-learn.org*. [Online]. Available: <https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html>. [Accessed: 26-Apr-2021].
- [21] "tf.keras.wrappers.scikit_learn.KerasClassifier," *Tensorflow.org*. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/wrappers/scikit_learn/KerasClassifier. [Accessed: 27-Apr-2021].
- [22] "sklearn.model_selection.GridSearchCV — scikit-learn 0.24.1 documentation," *Scikit-learn.org*. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html. [Accessed: 27-Apr-2021].
- [23] S. Doshi, "Various optimization algorithms for training neural network," *Towards Data Science*, 13-Jan-2019. [Online]. Available: <https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6>. [Accessed: 27-Apr-2021].
- [24] Keras Team, "Layer weight initializers," *Keras.io*. [Online]. Available: <https://keras.io/api/layers/initializers/>. [Accessed: 27-Apr-2021].
- [25] J. Brownlee, "Difference between a batch and an epoch in a neural network," *Machinelearningmastery.com*, 19-Jul-2018. [Online]. Available: <https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/>. [Accessed: 27-Apr-2021].
- [26] A. Nagpal, "L1 and L2 Regularization Methods," *Towards Data Science*, 13-Oct-2017. [Online]. Available: <https://towardsdatascience.com/l1-and-l2-regularization-methods-ce25e7fc831c>. [Accessed: 27-Apr-2021].
- [27] "D Woolcott," *Danielwoolcott.info*. [Online]. Available: https://www.danielwoolcott.info/projects/keras_custom_optimiser. [Accessed: 27-Apr-2021].
- [28] Keras Team, "Backend utilities," *Keras.io*. [Online]. Available: https://keras.io/api/utils/backend_utils/. [Accessed: 27-Apr-2021].
- [29] "sklearn.model_selection.KFold — scikit-learn 0.24.1 documentation," *Scikit-learn.org*. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html. [Accessed: 27-Apr-2021].

learn.org/stable/modules/generated/sklearn.model_selection.KFold.html. [Accessed: 27-Apr-2021].

- [30] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv [cs.NE]*, 2012.

8. Appendix 1:

A) Output of the Neural network model which is explained in the above 'Experiment and Findings' section

A. 1: Actual test output of Filter method model

```

Model : Keras Classifier - Filter
Epoch 1/10
1509/1509 [=====] - 2s 1ms/step - loss: 0.1662 - accuracy: 0.7676
Epoch 2/10
1509/1509 [=====] - 2s 1ms/step - loss: 0.1261 - accuracy: 0.8129
Epoch 3/10
1509/1509 [=====] - 2s 1ms/step - loss: 0.1234 - accuracy: 0.8133
Epoch 4/10
1509/1509 [=====] - 2s 1ms/step - loss: 0.1222 - accuracy: 0.8174
Epoch 5/10
1509/1509 [=====] - 2s 1ms/step - loss: 0.1196 - accuracy: 0.8253
Epoch 6/10
1509/1509 [=====] - 2s 1ms/step - loss: 0.1192 - accuracy: 0.8229
Epoch 7/10
1509/1509 [=====] - 2s 1ms/step - loss: 0.1182 - accuracy: 0.8283
Epoch 8/10
1509/1509 [=====] - 2s 1ms/step - loss: 0.1175 - accuracy: 0.8256
Epoch 9/10
1509/1509 [=====] - 2s 1ms/step - loss: 0.1146 - accuracy: 0.8304
Epoch 10/10
1509/1509 [=====] - 2s 1ms/step - loss: 0.1139 - accuracy: 0.8303
Time to build model (sec) : 17.1888
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/engine/sequential.py:450: UserWarning: `model.predict_classes()` is deprecated and will be removed in a future version. Use `model.predict` instead.
warnings.warn("`model.predict_classes()` is deprecated and will be removed in a future version. Use `model.predict` instead.")
Time to test model (sec) : 0.4599
Time elapsed (sec): 17.6678
[[10538 822]
 [1738 1962]]
Model Accuracy : 0.8300132802124834
Model Error Rate : 0.16998671978751656
Detection Rate : 0.5302702702702703
False Positive Type 1 : 0.07235915492957747
Type 2 Error : 0.4697297297297297
Matthews correlation coefficient (MCC): 0.5078180787927452

```

A. 2: Actual test output of Wrapper method model

```

Model : Keras Classifier - Wrapper
Epoch 1/10
1509/1509 [=====] - 2s 1ms/step - loss: 0.1579 - accuracy: 0.7850
Epoch 2/10
1509/1509 [=====] - 2s 1ms/step - loss: 0.1195 - accuracy: 0.8291
Epoch 3/10
1509/1509 [=====] - 2s 1ms/step - loss: 0.1151 - accuracy: 0.8306
Epoch 4/10
1509/1509 [=====] - 2s 1ms/step - loss: 0.1158 - accuracy: 0.8292
Epoch 5/10
1509/1509 [=====] - 2s 1ms/step - loss: 0.1144 - accuracy: 0.8275
Epoch 6/10
1509/1509 [=====] - 2s 1ms/step - loss: 0.1148 - accuracy: 0.8322
Epoch 7/10
1509/1509 [=====] - 2s 1ms/step - loss: 0.1120 - accuracy: 0.8366
Epoch 8/10
1509/1509 [=====] - 2s 1ms/step - loss: 0.1127 - accuracy: 0.8357
Epoch 9/10
1509/1509 [=====] - 2s 1ms/step - loss: 0.1116 - accuracy: 0.8379
Epoch 10/10
1509/1509 [=====] - 2s 1ms/step - loss: 0.1101 - accuracy: 0.8421
Time to build model (sec) : 16.7525
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/engine/sequential.py:450: UserWarning: `model.predict_classes()` is deprecated and will be re
warnings.warn('model.predict_classes()` is deprecated and `
Time to test model (sec) : 0.4636
Time elapsed (sec): 17.2323
[[10448  912]
 [ 1539 2161]]
Model Accuracy : 0.8372509960159362
Model Error Rate : 0.16274900398406378
Detection Rate :0.5840540540540541
False Positive Type 1 :0.08028169014084507
Type 2 Error : 0.41594594594594597
Matthews correlation coefficient (MCC):0.5381307807007474

```

A. 3: Actual test output of Embedded method model

```

Model : Keras Classifier - Embedded
Epoch 1/10
755/755 [=====] - 1s 1ms/step - loss: 0.1753 - accuracy: 0.7793
Epoch 2/10
755/755 [=====] - 1s 1ms/step - loss: 0.1296 - accuracy: 0.8175
Epoch 3/10
755/755 [=====] - 1s 1ms/step - loss: 0.1249 - accuracy: 0.8222
Epoch 4/10
755/755 [=====] - 1s 1ms/step - loss: 0.1186 - accuracy: 0.8258
Epoch 5/10
755/755 [=====] - 1s 1ms/step - loss: 0.1158 - accuracy: 0.8281
Epoch 6/10
755/755 [=====] - 1s 1ms/step - loss: 0.1159 - accuracy: 0.8288
Epoch 7/10
755/755 [=====] - 1s 1ms/step - loss: 0.1145 - accuracy: 0.8321
Epoch 8/10
755/755 [=====] - 1s 1ms/step - loss: 0.1131 - accuracy: 0.8334
Epoch 9/10
755/755 [=====] - 1s 1ms/step - loss: 0.1130 - accuracy: 0.8336
Epoch 10/10
755/755 [=====] - 1s 1ms/step - loss: 0.1120 - accuracy: 0.8341
Time to build model (sec) : 8.8648
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/engine/sequential.py:450: UserWarning: `model.predict_classes()` is deprecated and will be re
warnings.warn('model.predict_classes()` is deprecated and `
Time to test model (sec) : 0.2903
Time elapsed (sec): 9.1702
[[10369  991]
 [ 1605 2095]]
Model Accuracy : 0.8276228419654714
Model Error Rate : 0.17237715803452858
Detection Rate :0.5662162162162162
False Positive Type 1 :0.08723591549295774
Type 2 Error : 0.4337837837837838
Matthews correlation coefficient (MCC):0.5108461119040855

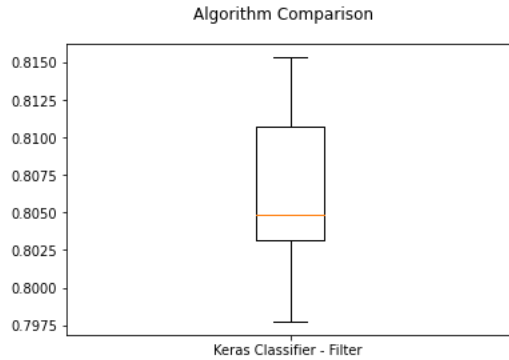
```

B. The Output of Regularized ('L2') neural network.

B. 1. Actual test output of Regularized neural network Filter method model

Best : 0.808700 using {'batch_size': 20, 'epochs': 10, 'init': 'glorot_uniform', 'optimizer': 'WAME1'}

Keras Classifier - Filter (Training): 0.806280 (0.005541)



```

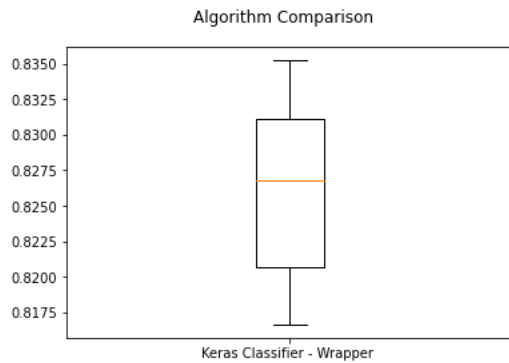
Model : Keras Classifier - Filter
Epoch 1/10
1509/1509 [=====] - 2s 1ms/step - loss: 0.2341 - accuracy: 0.7492
Epoch 2/10
1509/1509 [=====] - 2s 1ms/step - loss: 0.1627 - accuracy: 0.7964
Epoch 3/10
1509/1509 [=====] - 2s 1ms/step - loss: 0.1570 - accuracy: 0.7966
Epoch 4/10
1509/1509 [=====] - 2s 1ms/step - loss: 0.1551 - accuracy: 0.7962
Epoch 5/10
1509/1509 [=====] - 2s 1ms/step - loss: 0.1494 - accuracy: 0.8039
Epoch 6/10
1509/1509 [=====] - 2s 1ms/step - loss: 0.1478 - accuracy: 0.8065
Epoch 7/10
1509/1509 [=====] - 2s 1ms/step - loss: 0.1483 - accuracy: 0.8009
Epoch 8/10
1509/1509 [=====] - 2s 1ms/step - loss: 0.1457 - accuracy: 0.8061
Epoch 9/10
1509/1509 [=====] - 2s 1ms/step - loss: 0.1429 - accuracy: 0.8112
Epoch 10/10
1509/1509 [=====] - 2s 1ms/step - loss: 0.1430 - accuracy: 0.8099
Time to build model (sec) : 17.6510
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/engine/sequential.py:450: UserWarning: `model.predict_classes()` is deprecated and will be re
warnings.warn("`model.predict_classes()` is deprecated and ")
Time to test model (sec) : 0.4761
Time elapsed (sec): 18.1435
[[10631  729]
 [ 2161 1539]]
Model Accuracy : 0.8081009296148738
Model Error Rate : 0.1918990703851262
Detection Rate : 0.41594594594594597
False Positive Type 1 : 0.0641725352112676
Type 2 Error : 0.5840540540540541
Matthews correlation coefficient (MCC): 0.42341114652864553

```

B. 2. Actual test output of Regularized neural network Wrapper method model

Best: 0.828095 using {'batch_size': 20, 'epochs': 10, 'init': 'glorot_uniform', 'optimizer': 'WAME4'}

Keras Classifier - Wrapper (Training) : 0.826039 (0.005873)



```

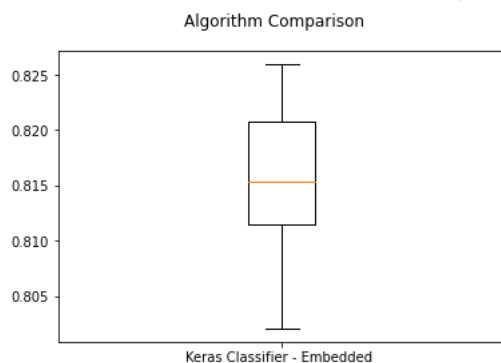
Model : Keras Classifier - Wrapper
Epoch 1/10
1509/1509 [=====] - 2s 1ms/step - loss: 0.2256 - accuracy: 0.7534
Epoch 2/10
1509/1509 [=====] - 2s 1ms/step - loss: 0.1613 - accuracy: 0.8051
Epoch 3/10
1509/1509 [=====] - 2s 1ms/step - loss: 0.1530 - accuracy: 0.8225
Epoch 4/10
1509/1509 [=====] - 2s 1ms/step - loss: 0.1511 - accuracy: 0.8188
Epoch 5/10
1509/1509 [=====] - 2s 1ms/step - loss: 0.1439 - accuracy: 0.8289
Epoch 6/10
1509/1509 [=====] - 2s 1ms/step - loss: 0.1434 - accuracy: 0.8280
Epoch 7/10
1509/1509 [=====] - 2s 1ms/step - loss: 0.1421 - accuracy: 0.8257
Epoch 8/10
1509/1509 [=====] - 2s 1ms/step - loss: 0.1404 - accuracy: 0.8286
Epoch 9/10
1509/1509 [=====] - 2s 1ms/step - loss: 0.1397 - accuracy: 0.8277
Epoch 10/10
1509/1509 [=====] - 2s 1ms/step - loss: 0.1388 - accuracy: 0.8252
Time to build model (sec) : 17.2878
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/engine/sequential.py:450: UserWarning: `model.predict_classes()` is deprecated and will be removed in a future version. Use `model.predict` instead.
warnings.warn("`model.predict_classes()` is deprecated and will be removed in a future version. Use `model.predict` instead.")
Time to test model (sec) : 0.4686
Time elapsed (sec): 17.7718
[[10644  716]
 [1893 1807]]
Model Accuracy : 0.8267596281540505
Model Error Rate : 0.17324037184594954
Detection Rate : 0.4883783783783784
False Positive Type 1 : 0.0630281690140845
Type 2 Error : 0.5116216216216216
Matthews correlation coefficient (MCC): 0.49032184264367223

```

B. 3. Actual test output of Regularized neural network Embedded method model

Best: 0.817651 using {'batch_size': 20, 'epochs': 10, 'init': 'glorot_uniform', 'optimizer': 'adam'}

Keras Classifier - Embedded (Training): 0.814933 (0.007052)



```

C: -----
Model : Keras Classifier - Embedded
Epoch 1/10
1509/1509 [=====] - 2s 1ms/step - loss: 0.2352 - accuracy: 0.7770
Epoch 2/10
1509/1509 [=====] - 2s 1ms/step - loss: 0.1648 - accuracy: 0.7925
Epoch 3/10
1509/1509 [=====] - 2s 1ms/step - loss: 0.1588 - accuracy: 0.7993
Epoch 4/10
1509/1509 [=====] - 2s 1ms/step - loss: 0.1575 - accuracy: 0.7994
Epoch 5/10
1509/1509 [=====] - 2s 1ms/step - loss: 0.1523 - accuracy: 0.8074
Epoch 6/10
1509/1509 [=====] - 2s 1ms/step - loss: 0.1519 - accuracy: 0.8073
Epoch 7/10
1509/1509 [=====] - 2s 1ms/step - loss: 0.1489 - accuracy: 0.8130
Epoch 8/10
1509/1509 [=====] - 2s 1ms/step - loss: 0.1464 - accuracy: 0.8163
Epoch 9/10
1509/1509 [=====] - 2s 1ms/step - loss: 0.1455 - accuracy: 0.8159
Epoch 10/10
1509/1509 [=====] - 2s 1ms/step - loss: 0.1462 - accuracy: 0.8151
Time to build model (sec) : 17.4145
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/engine/sequential.py:450: UserWarning: `model.predict_classes()` is deprecated and will be removed in a future version. Use `model.predict` instead.
warnings.warn("`model.predict_classes()` is deprecated and will be removed in a future version. Use `model.predict` instead.")
Time to test model (sec) : 0.4760
Time elapsed (sec): 17.9057
[[7899 3461]]
[[1939 1761]]
Model Accuracy : 0.6414342629482072
Model Error Rate : 0.35856573705179284
Detection Rate :0.47594594594594597
False Positive Type 1 :0.3046654929577465
Type 2 Error : 0.524054054054054
Matthews correlation coefficient (MCC):0.1549265991905138

```


9. Appendix 2:


Steps to run the code file Th “**ML_CW_13199041.ipynb**” and “**Regu_Keras_ML_CW_13199041.ipynb**”

Steps are the same to run and check both files

1. Open Google Colab on your browser - <https://colab.research.google.com/>
2. Login with your google account
3. Click on ‘File’ > ‘Upload Notebook’ and upload the .ipynb file.

4. After the file is loaded, click on the  folder button on the left-hand-side bar

and then click  the upload button and upload the ‘adult.data’ (training data) and ‘adult.test’ (testing data)

5. After that you can click on the  play button next to the code block and run the coded to see the output.