Letter Classification with Neural Networks

Abhishek Jallawaram
Department of Computer Science (CS580)
George Mason University
VA, USA
ajallawa@gmu.edu or G01373042

***Abstract*—The document depicts the implementation of Neural network for letter classification.**

***Index Terms*—Sklearn, Keras, Multi Layer Perceptron Classifier, Random Forest Classifier, Neural Network, Standard Scalar, Min Max Scalar**

I. Introduction

Implementation of machine Multi Layer Perceptron classifier, Random Forest classifier and Keras Neural network classification to achieve the best classification model for letter classification.

II. Problem Scenario

A. Classification

The objective is to identify each of a large number of black-and-white rectangular pixel displays as one of the 26 capital letters in the English alphabet.

III. Classification

A. Data Pre-Processing

The label ('lettr') is converted into numerical data by using the Label encoder.

Label Encoding: Assign each categorical value an integer value based on alphabetical order.

We have observed the below frequency for the labels.

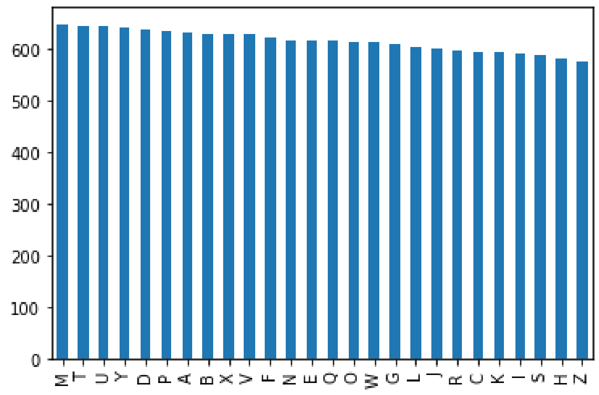


Fig. 1. Train Labels

MinMaxSclar is an estimator which scales and translates each feature individually such that it is in the given range on the training set, e.g. between zero and one.

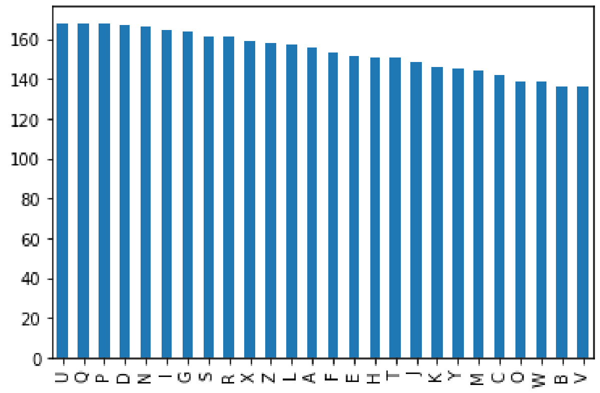


Fig. 2. Train Labels

X_std = (X-X.min(axis=0))/(X.max(axis=0)-X.min(axis=0))

X_scaled = X_std*(max-min)+min

StandardScaler follows Standard Normal Distribution wherein, it makes mean = 0 and scales the data to unit variance. Standardize features by removing the mean and scaling to unit variance. The standard score of a sample x is calculated as:

z = (x - u) / s

Where **u** is the mean of the training samples and **s** is the standard deviation of the training samples.

The different classification methods are used on the scaled data to determine the best model which provides the best metrics. Classification Models used:

- MLPClassifier (sklearn)
- Random Forests (sklearn)
- Keras (TensorFlow)

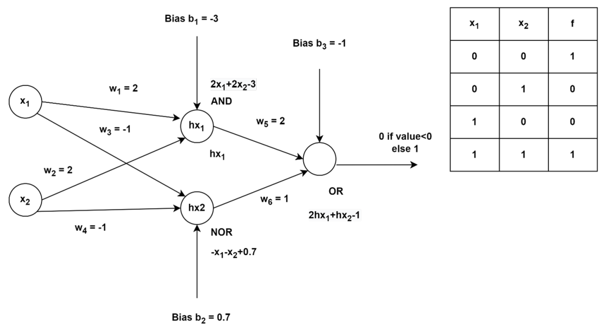
B. MLPClassifier

Multi-layer Perceptron (MLP) is a supervised learning algorithm that learns a function by training on a dataset.

f$(\cdot) : R^m \rightarrow R^o$

Where m is the number of dimensions for input and o is the number of dimensions for output.Given a set of features X and a target y, it can learn a non-linear function approximator for either classification or regression.

The leftmost layer, known as the input layer, consists of a set of neurons representing the input features. Each neuron in the hidden layer transforms the values from the previous layer with a weighted linear summation followed by a non-linear activation function - like the hyperbolic tan function. The output layer receives the values from the last hidden layer and transforms them into output values.



x_1	x_2	f
0	0	1
0	1	0
1	0	0
1	1	1

Fig. 3. Perceptron model for XNOR gate

From Fig.1 we can observe that the x1 and x2 are taken as input layers and hx1 and hx2 can be considered to be hidden layers of the perceptron model. The Relu activation function is used to generate the output and the final reult is calculated a the output layer y. Weights and bias the key factors in building the neural network.

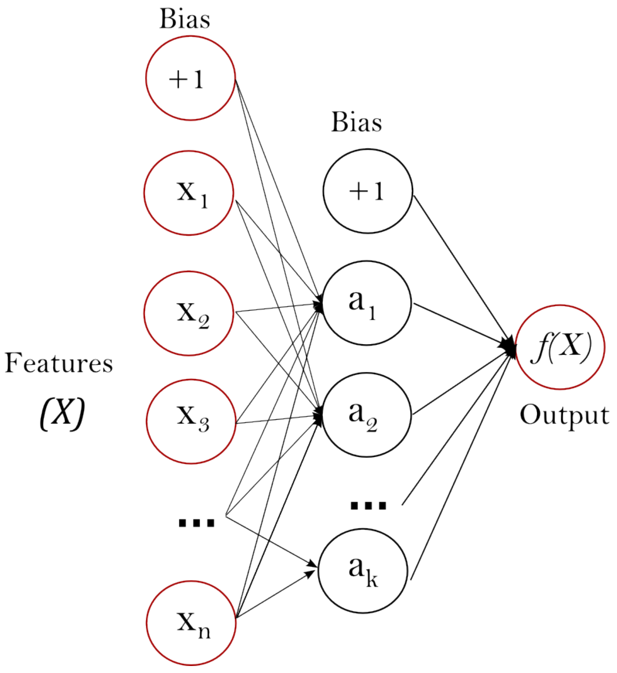


Fig. 4. Example for neural Network

In our scenario, we have implemented the **sklearn.MLPClassifier** to build the neural network with default metrics.

- hidden_layer_sizes=(100,)
- activation= relu
- solver= adam
- alpha=0.0001
- batch_size= auto
- learning_rate='constant'
- learning_rate_init=0.001
- power_t=0.5
- max_iter=200
- shuffle=True
- random_state=None
- tol=0.0001
- verbose=False
- warm_start=False
- momentum=0.9
- nesterovs_momentum=True
- early_stopping=False
- validation_fraction=0.1
- beta_1=0.9
- beta_2=0.999
- epsilon=1e-08
- n_iter_no_change=10
- max_fun=15000

We have observed the below accuracy when the classifier was applied.
MinMax Scalar + X_Test : Accuracy 0.82
MinMax Scalar + Test : Accuracy 0.80
Standard Scalar + X_Test : Accuracy 0.93
Standard Scalar + Test : Accuracy 0.95

Hypertuning was performed on the below parameters with accuracy as the metric and GridSearhCV was deployed to determine the best model.

- hidden_layer_sizes=(100,),(100,3),(250,300)
- activation= 'tanh', 'relu','logistic'
- solver= 'adam','sgd', 'adam','lbfgs'
- alpha= 10,1,0.1,0.01,0.001,0.0001,0.00001
- learning_rate='constant', 'adaptive'

1) Activation:: Relu : The rectified linear unit function, returns f(x) = max(0, x)
'tanh' : The hyperbolic tan function,
returns f(x) = tanh(x).
'logistic' : The logistic sigmoid function,
returns f(x) = 1 / (1 + exp(-x)).

2) Solvers:: 'adam' : Refers to a stochastic gradient-based optimizer proposed by Kingma, Diederik, and Jimmy Ba
'sgd' : Refers to stochastic gradient descent.
'lbfgs' : This is an optimizer in the family of quasi-Newton methods.

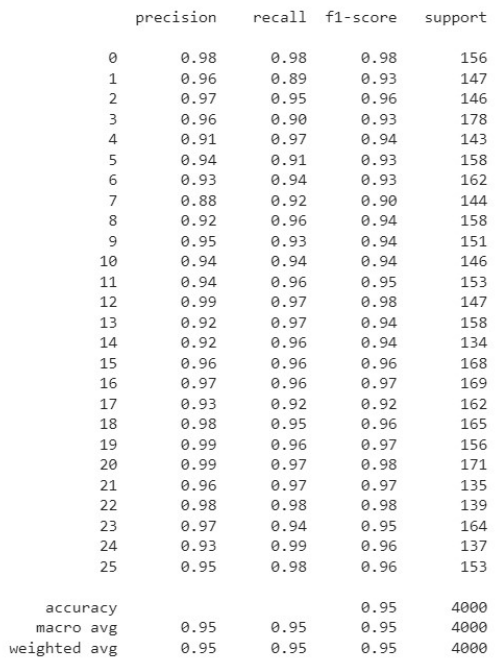
We have run multiple iterations and observed the best model for the below metrics.

- hidden_layer_sizes=(250,300)
- activation= logistic
- solver= adam
- alpha=0.0001
- batch_size= auto
- learning_rate='constant'
- learning_rate_init=0.001
- power_t=0.5
- max_iter=200
- shuffle=True
- random_state=None
- tol=0.0001
- verbose=False
- warm_start=False
- momentum=0.9
- nesterovs_momentum=True
- early_stopping=False
- validation_fraction=0.1
- beta_1=0.9
- beta_2=0.999
- epsilon=1e-08
- n_iter_no_change=10
- max_fun=15000

We have observed the below accuracy when the classifier was applied post hypertuning. Best results were observed when **logistic function(Sigmoid)** was used as activation function for hidden layers.

f(x) = 1 / (1 + exp(-x)).

Cross Validation Accuracy Average = 0.94
Standard Scalar + Test : Accuracy = 0.97

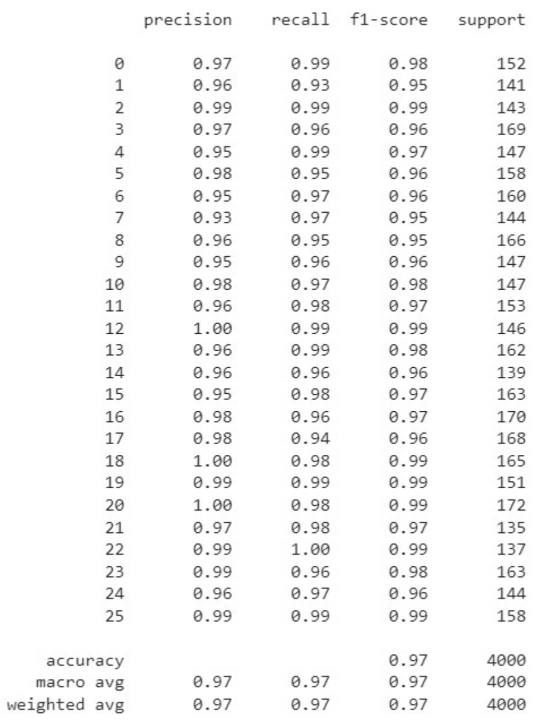


```
              precision    recall  f1-score   support

           0       0.98      0.98      0.98       156
           1       0.96      0.89      0.93       147
           2       0.97      0.95      0.96       146
           3       0.96      0.90      0.93       178
           4       0.91      0.97      0.94       143
           5       0.94      0.91      0.93       158
           6       0.93      0.94      0.93       162
           7       0.88      0.92      0.90       144
           8       0.92      0.96      0.94       158
           9       0.95      0.93      0.94       151
          10       0.94      0.94      0.94       146
          11       0.94      0.96      0.95       153
          12       0.99      0.97      0.98       147
          13       0.92      0.97      0.94       158
          14       0.92      0.96      0.94       134
          15       0.96      0.96      0.96       168
          16       0.97      0.96      0.97       169
          17       0.93      0.92      0.92       162
          18       0.98      0.95      0.96       165
          19       0.99      0.96      0.97       156
          20       0.99      0.97      0.98       171
          21       0.96      0.97      0.97       135
          22       0.98      0.98      0.98       139
          23       0.97      0.94      0.95       164
          24       0.93      0.99      0.96       137
          25       0.95      0.98      0.96       153

    accuracy                           0.95      4000
   macro avg       0.95      0.95      0.95      4000
weighted avg       0.95      0.95      0.95      4000
```

Fig. 5. Test Metrics before Hypertuning MLPClassifier



```
              precision    recall  f1-score   support

           0       0.97      0.99      0.98       152
           1       0.96      0.93      0.95       141
           2       0.99      0.99      0.99       143
           3       0.97      0.96      0.96       169
           4       0.95      0.99      0.97       147
           5       0.98      0.95      0.96       158
           6       0.95      0.97      0.96       160
           7       0.93      0.97      0.95       144
           8       0.96      0.95      0.95       166
           9       0.95      0.96      0.96       147
          10       0.98      0.97      0.98       147
          11       0.96      0.98      0.97       153
          12       1.00      0.99      0.99       146
          13       0.96      0.99      0.98       162
          14       0.96      0.96      0.96       139
          15       0.95      0.98      0.97       163
          16       0.98      0.96      0.97       170
          17       0.98      0.94      0.96       168
          18       1.00      0.98      0.99       165
          19       0.99      0.99      0.99       151
          20       1.00      0.98      0.99       172
          21       0.97      0.98      0.97       135
          22       0.99      1.00      0.99       137
          23       0.99      0.96      0.98       163
          24       0.96      0.97      0.96       144
          25       0.99      0.99      0.99       158

    accuracy                           0.97      4000
   macro avg       0.97      0.97      0.97      4000
weighted avg       0.97      0.97      0.97      4000
```

Fig. 6. Test Metrics after Hypertuning MLPClassifier

C. Random Forest:

A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.The goal of ensemble methods is to combine the predictions of several base estimators built with a given learning algorithm in order to improve generalizability / robustness over a single estimator.

In random forests,each tree in the ensemble is built from a sample drawn with replacement from the training set. In addition, when splitting a node during the construction of the tree, the split that is chosen is no longer the best split among all features. Instead, the split that is picked is the best split among a random subset of the features. As a result of this randomness, the bias of the forest usually slightly increases but, due to averaging, its variance also decreases, usually more than compensating for the increase in bias, hence yielding an overall better model.
In our scenario, we have implemented the **sklearn.RandomForestClassifier** to build the RandomForest with default metrics.

- n_estimators=100
- criterion= gini
- max_depth=None
- min_samples_leaf=1
- min_samples_split=2
- min_weight_fraction_leaf=0.0
- learning_rate_init=0.001
- max_features='sqrt'

- max_leaf_nodes=None
- min_impurity_decrease=0.0
- bootstrap=True
- oob_score=False
- n_jobs=None
- random_state=None
- verbose=0
- warm_start=False
- class_weight=None
- ccp_alpha=0.0
- , max_samples=None

We have observed the below accuracy when the classifier was applied. Standard Scalar + X_Test : Accuracy 0.95
Standard Scalar + Test : Accuracy 0.96

Hypertuning was performed on the below parameters with accuracy as the metric and GridSearhCV was deployed to determine the best model.

- n_estimators= [100,200,300,400,.......,1400]
- bootstrap=False
- criterion = [’gini’,’entropy’]

1) Gini: Calculates the amount of probability of a specific feature that is classified incorrectly when selected randomly. H(Q$_m$) $= \sum_k p_{mk}(1 - p_{mk})$

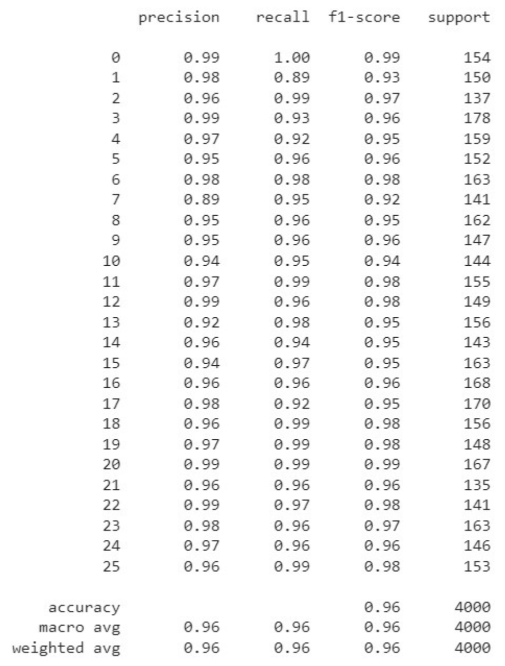
2) Entropy: It is the measurement of the impurity or randomness in the data points.

We have run multiple iterations and observed the best model for the below metrics.

- n_estimators=1400
- criterion= gini
- max_depth=None
- min_samples_leaf=1
- min_samples_split=2
- min_weight_fraction_leaf=0.0
- learning_rate_init=0.001
- max_features=’sqrt’
- max_leaf_nodes=None
- min_impurity_decrease=0.0
- bootstrap=False
- oob_score=False
- n_jobs=None
- random_state=None
- verbose=0
- warm_start=False
- class_weight=None
- ccp_alpha=0.0
- , max_samples=None

We have observed the below accuracy when the classifier was applied post hypertuning. Best results were observed when **n_estimators** were increased. As number of trees were increased, the accuracy increased.

Cross Validation Accuracy Average = 0.96
Standard Scalar + Test : Accuracy = 0.97

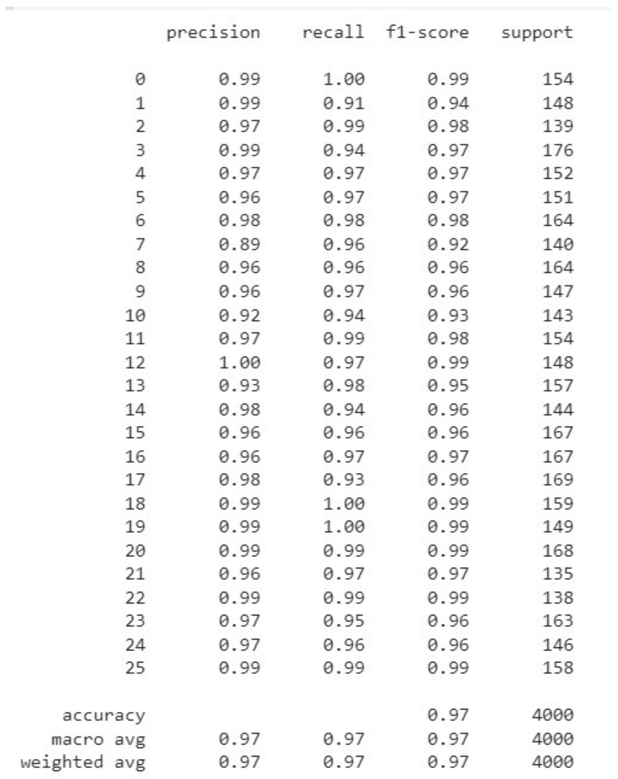


```
              precision    recall  f1-score   support

           0       0.99      1.00      0.99       154
           1       0.98      0.89      0.93       150
           2       0.96      0.99      0.97       137
           3       0.99      0.93      0.96       178
           4       0.97      0.92      0.95       159
           5       0.95      0.96      0.96       152
           6       0.98      0.98      0.98       163
           7       0.89      0.95      0.92       141
           8       0.95      0.96      0.95       162
           9       0.95      0.96      0.96       147
          10       0.94      0.95      0.94       144
          11       0.97      0.99      0.98       155
          12       0.99      0.96      0.98       149
          13       0.92      0.98      0.95       156
          14       0.96      0.94      0.95       143
          15       0.94      0.97      0.95       163
          16       0.96      0.96      0.96       168
          17       0.98      0.92      0.95       170
          18       0.96      0.99      0.98       156
          19       0.97      0.99      0.98       148
          20       0.99      0.99      0.99       167
          21       0.96      0.96      0.96       135
          22       0.99      0.97      0.98       141
          23       0.98      0.96      0.97       163
          24       0.97      0.96      0.96       146
          25       0.96      0.99      0.98       153

    accuracy                           0.96      4000
   macro avg       0.96      0.96      0.96      4000
weighted avg       0.96      0.96      0.96      4000
```

Fig. 7. Test Metrics before Hypertuning RandomForest Classifier



```
              precision    recall  f1-score   support

           0       0.99      1.00      0.99       154
           1       0.99      0.91      0.94       148
           2       0.97      0.99      0.98       139
           3       0.99      0.94      0.97       176
           4       0.97      0.97      0.97       152
           5       0.96      0.97      0.97       151
           6       0.98      0.98      0.98       164
           7       0.89      0.96      0.92       140
           8       0.96      0.96      0.96       164
           9       0.96      0.97      0.96       147
          10       0.92      0.94      0.93       143
          11       0.97      0.99      0.98       154
          12       1.00      0.97      0.99       148
          13       0.93      0.98      0.95       157
          14       0.98      0.94      0.96       144
          15       0.96      0.96      0.96       167
          16       0.96      0.97      0.97       167
          17       0.98      0.93      0.96       169
          18       0.99      1.00      0.99       159
          19       0.99      1.00      0.99       149
          20       0.99      0.99      0.99       168
          21       0.96      0.97      0.97       135
          22       0.99      0.99      0.99       138
          23       0.97      0.95      0.96       163
          24       0.97      0.96      0.96       146
          25       0.99      0.99      0.99       158

    accuracy                           0.97      4000
   macro avg       0.97      0.97      0.97      4000
weighted avg       0.97      0.97      0.97      4000
```

Fig. 8. Test Metrics after Hypertuning RandomForest Classifier

D. Neural Networs :Tensor Flow

Keras API from TensorFlow is used to build the neural network.

The model is build based on the below details.

- Input_shape: Contains the input of the each layer
- Dense : To apply the activation function over ((w • x) + b) and the number of hidden nodes in the each layer.
- Loss : The goal of the neural network is to minimize the loss function, i.e., the difference between predicted and observed values. In our case we pick SparseCategorical-Crossentropy.
- Optimizer: We use the optimizer function Adam.
- Epoch : Number of runs on the model
- Metrics : SparseCategoricalAccuracy is used as metric.
- fit() : Trains the model,calculates the weights and biases.

We define a Keras model with one input layer and three dense layers (two hidden layers and one output layer) to get the required output as shown in Fig.9. We have observed the accuracy as 0.9687.

- Input_layer : 16 nodes
- Hidden_layer1 : 128 nodes
- Hidden_layer2 : 128 nodes
- Output_layer : 26 nodes

Standard Scalar + Test : Accuracy = 0.9697

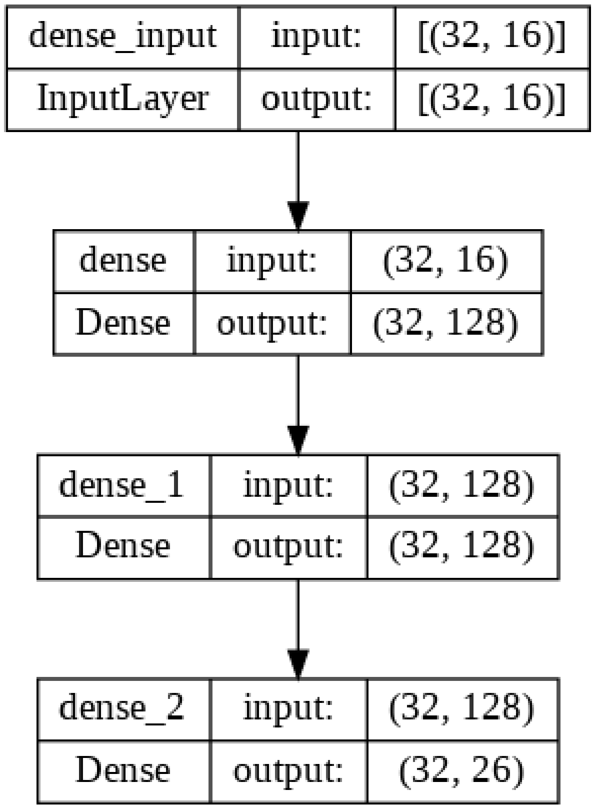
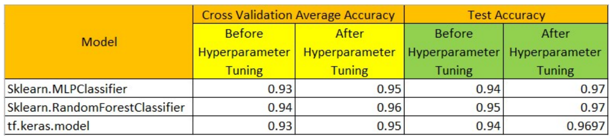


Fig. 9. Model built from Keras



Model	Cross Validation Average Accuracy		Test Accuracy	
	Before Hyperparameter Tuning	After Hyperparameter Tuning	Before Hyperparameter Tuning	After Hyperparameter Tuning
Sklearn.MLPClassifier	0.93	0.95	0.94	0.97
Sklearn.RandomForestClassifier	0.94	0.96	0.95	0.97
tf.keras.model	0.93	0.95	0.94	0.9697

Fig. 10. Model built from Keras

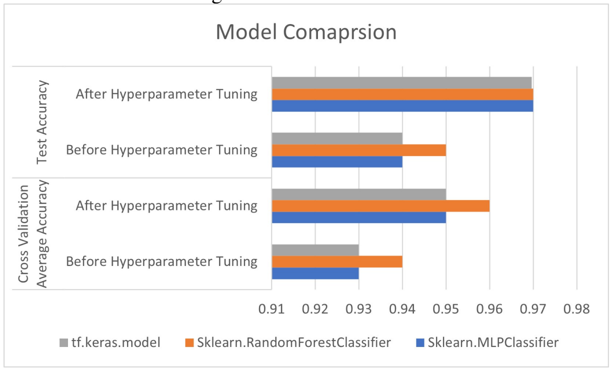


Fig. 11. Model built from Keras

REFERENCES

[1] S. Russel, B. Noble, and P.Norvig, ‘Artificial Intelligence A Modern Approach - Fourth Edition’