# Analysis Report

## SmartGuide: A smart campus guide using BLE based indoor localization

4/26/2020
UET, LHR
Submitted to: Dr. Sheikh Faisal Rasheed

# Introduction:

## Project Description:

Outdoor and indoor localization is an integral component of IoT (Internet of Things) in this era of mobile computing. Indoor localization can open new horizons for ubiquitous applications targeting university departments, government small institutes, airports, shopping malls, museums etc. Our project will find the location of a specific person by using appropriate machine learning approach using BLE based Android application. This location will be used to provide guided tour of the indoor building (Computer Engineering Department at UET, Lahore) that we will use to validate our work. This project will guide persons who are not much familiar with visiting place. It has an Android application that will predict the indoor location of a person at room level and also gives information of current room location and nearby rooms in the form of text, images, audio and videos.

## Description of the analysis study:

This study aimed to focus on detailed analysis of the performance of k-NN, Random Forest and ANN on our collected data set. The reason of choosing these three algorithms is because that when we did literature review, we realized that these three algorithms mostly used on this kind of dataset that we have, because they give better performance as compared to others, so we decided to choose these three algorithms and then check their performance on our collected data set and at the end after the comparison and detailed analysis of the results of these algorithms, we picked the better one.

# Background and Data Description:

## Background:

Our proposed approach is basically to find the indoor location(at room level) of a person by using suitable machine learning approach which recognizes the patterns of the collected fingerprints that are basically RSSI (Received Signal Strength Indicator) values of different beacons and predicts the output. BLE beacons are Bluetooth radio transmitter powered by

batteries and they broadcasts BLE signals. The Bluetooth enabled smart phones are capable of scanning and displaying these signals in the form of RSSI values.

## Data Description:

Each beacon has its own MAC address which is used for its identification. Our collected fingerprints data looks like Figure 1.

| | | | | | |
|---|---|---|---|---|---|
| L1 | RSSI 1 | RSSI 2 | RSSI 3 | • • | RSSI M |
| L2 | RSSI 1 | RSSI 2 | RSSI 3 | • • | RSSI M |
| • • | • • | • • | • • | • • | • • |
| Ln | RSSI 1 | RSSI 2 | RSSI 3 | • • | RSSI M |

Figure 1

We have total 24 beacons, that's why we have 24 input features and each feature contains the RSSI values of specific beacon associated with that feature, and the output is the labeled name of the location. We have total 23 labels which mean we have total 23 classes hence it is a multi-classification problem. RSSI values ranges from -95dBm (low RSSI ~ far off Access point) to -50dBm (demonstrating high RSSI close to Access point). **dBm stands** for decibel milliwatts. **dBm** can be used in radio waves and microwaves as a measure of absolute power because of its capability to express both very large and very small values in short form. The closer the number is to 0, the better off your signal strength is.

# Experimental Area:

The experimental area that we covered is the Computer Engineering building of UET, LHR. We skip some rooms of that building because they are locked most of the time. The representation of the deployment of beacons in dept is shown in the following figures. Figure 2 represents the ground floor and Figure 3 represents 1st floor of the building. Small blue circle represent BLE beacon. Following figures only represents those rooms that we have considered in our experimental area. Each room is assigned a label as shown in the figures. Corridors are divided into 4 portions. Each portion is represented by a dotted box and assigned a label shown in the figures.

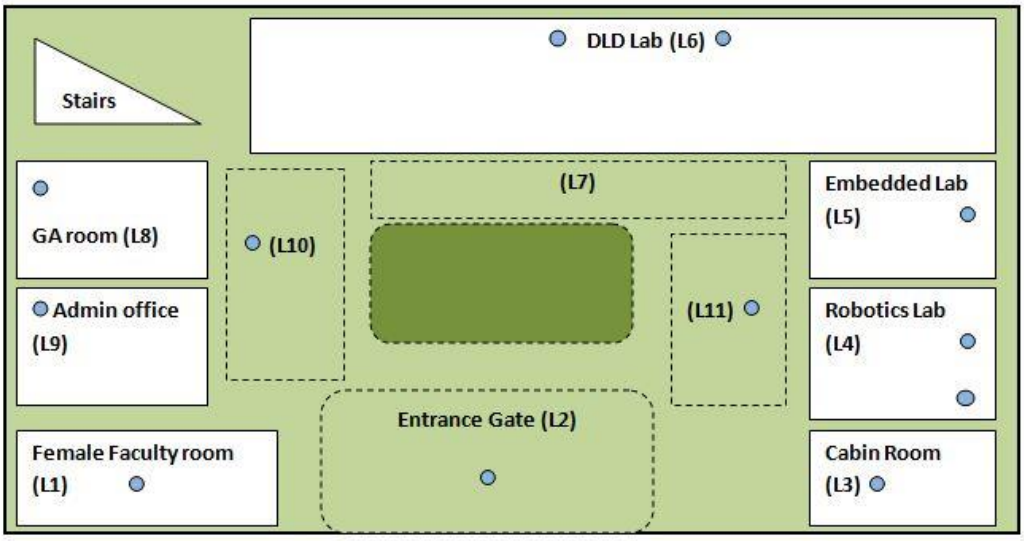



Figure 2

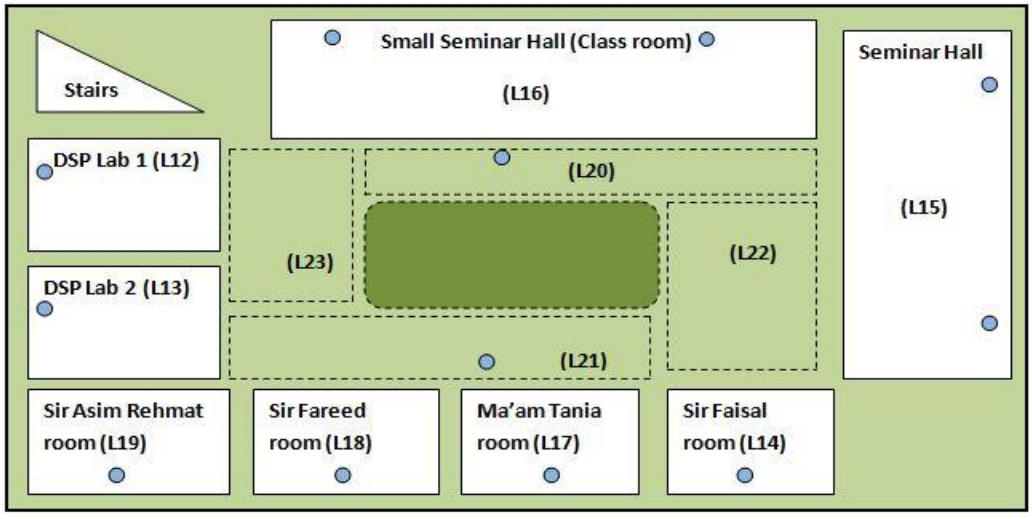


Figure 3

We have collected the data from all the locations except 5 locations. The total collected samples are 6506. The detail of number of samples that we have collected per location is shown in the following figure.

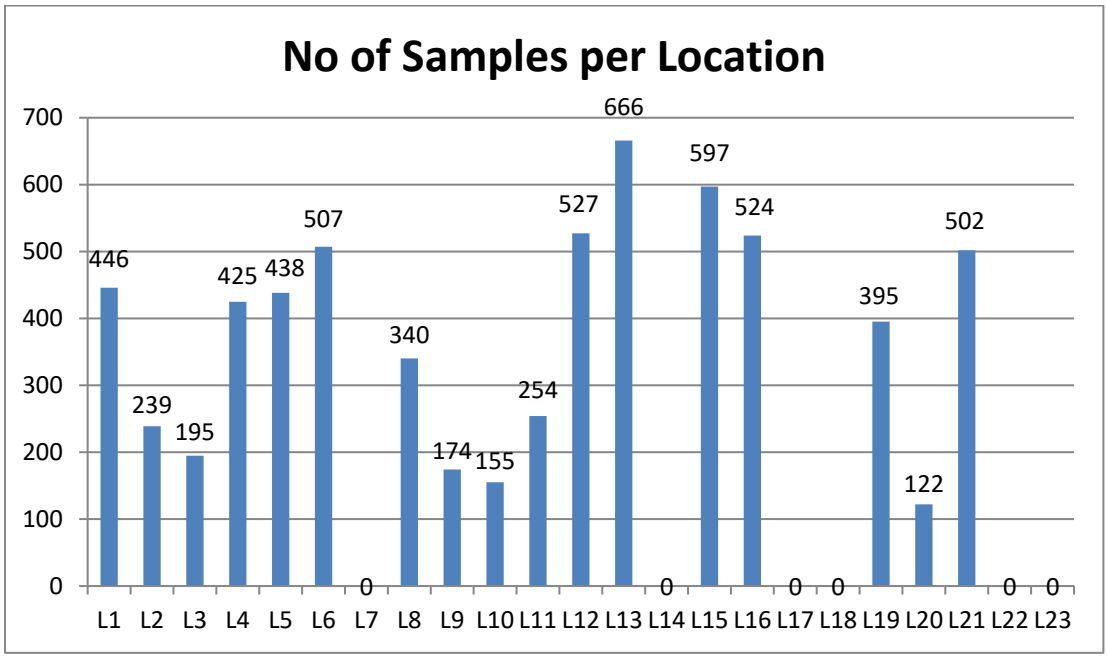



Figure 4

# Collected data set:

## Preparation of data set:

When we captured the data, numerous .csv files generated in which each .csv file contain one sample. We used 5 different cell phones for data capturing. Then we compile all the generated .csv files into single .csv file in specific format that we have shown in Figure 1. We discard all those RSSI values of surrounding BLE enable devices which are not from our BLE beacons.

## Pre-processing of data:

We have done pre-processing on our collected data set. In pre-processing, we have dropped all those rows that have null values in all of their input columns. During data collection some access points are visible, some not hence there were lot of missing values in data set. So, we

replace missing values with -100dbm. The reason for replacing the missing values with -100dbm is because it shows extremely weak signal means it is far off from access point. We have not done normalization on our data set because we didn't find it useful for our collected data set.

# Analysis:

## Performance Measures criterion for the comparison of algorithms:

**Accuracy** is the measurement of model performance. It is calculated as the no of samples correctly classified/ total no of samples.

**Precision** tells about how accurate your model is out of those predicted positive, how many of them are true positive. Precision is a good measure to determine, when the costs of False Positive is high.

**Recall** actually calculates how many of the Actual Positives our model captures through labeling it as Positive (True Positive).

**For calculation of precision and recall for multi-classification**, we can calculate the precision and recall separately for each class. For example, for class X, we can calculate its precision like that TP_X/Total Predicted X and recall for X class is calculated as TP_X/Total Actually Labeled X. Then we calculate the average precision and recall of all classes and hence we will get the approximate calculation of precision and recall for them.

Here are the formulas of accuracy, precision and recall for binary-classification:

$$\text{Accuracy}(\%) = \frac{\text{Tp} + \text{Tn}}{\text{Tp} + \text{Tn} + \text{Ep} + \text{Fn}}$$

$$\text{Precision} = \frac{\text{Tp}}{\text{Tp} + \text{Fp}}$$

$$\text{Recall} = \frac{\text{Tp}}{\text{Tp} + \text{Fn}}$$

Figure 5

## Data Trainings by using different methods:

## k-Nearest Neighbors classifier (k-NN):

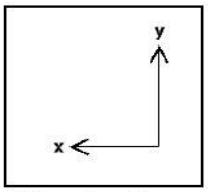
k-NN is the simplest and supervised machine learning algorithm. Here we used it for multi-classification. k-NN uses data and classify new data points based on majority vote of the nearby points. Before the implementation of k-NN, we have to tell the values of the hyper parameters of k-NN. The basic parameters for k-NN algorithm are:

- **Number of nearest neighbors:**
  We have to choose number of nearest neighbors in order to find the class of new data point based on the majority votes from these nearest neighbors.
- **Distance measure:**
  There are two major distance measure Euclidean and Manhattan. We have to choose one of them as a distance measure criterion. Figure 6 shows the Euclidean distance measure and Figure 7 shows Manhattan distance measure.

$$d(x, y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

Euclidean distance formula can be used to calculate the distance between two data points in a plane.


Figure 6




**Manhattan**

Distance *d* will be calculated using an ***absolute sum of difference*** between its cartesian co-ordinates as below:

$$d = \sum_{i=1}^{n} |x_i - y_i|$$

Figure 7

- **Weights assigned to neighbors:**

  The weight parameter has two choices: 'distance' and 'uniform'. The 'uniform' weight means that each of the k neighbors has equal vote, whatever its distance from target point. The 'distance' measure means that in the k neighbors, the neighbor who is close to target point has given more weight than the neighbor who is far from the target point.

## Hyper-parameters tuning:

Before we implement k-NN, we need to find the best values for parameters at which accuracy is maximum. For this purpose, we use GridsearchCV function to test for all possible combinations of n_neighbors, weights, distance in order to find out best parameters at which accuracy is maximum. We take values of k neighbors from 1 to 26, weights = {'uniform', 'distance'} and distance measure = {'euclidean', 'manhattan'}, then we try to find the best possible combination. Figure 8 represents the impact of choosing the distance measure on accuracy score.

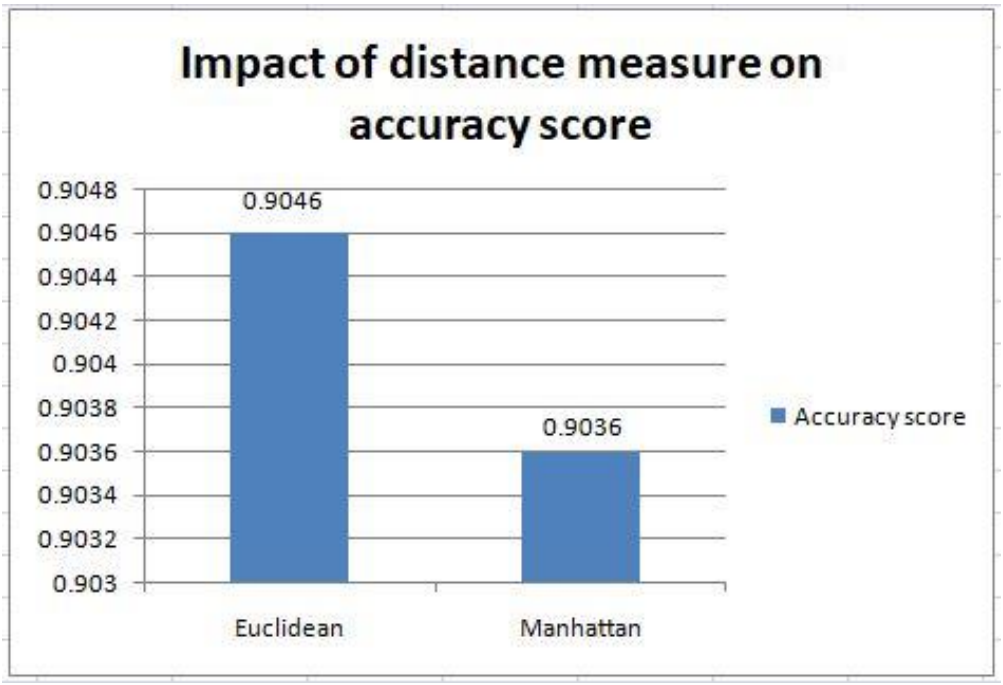



Figure 8

Figure 9 represents impact of choosing the weight assigning criterion on accuracy score.

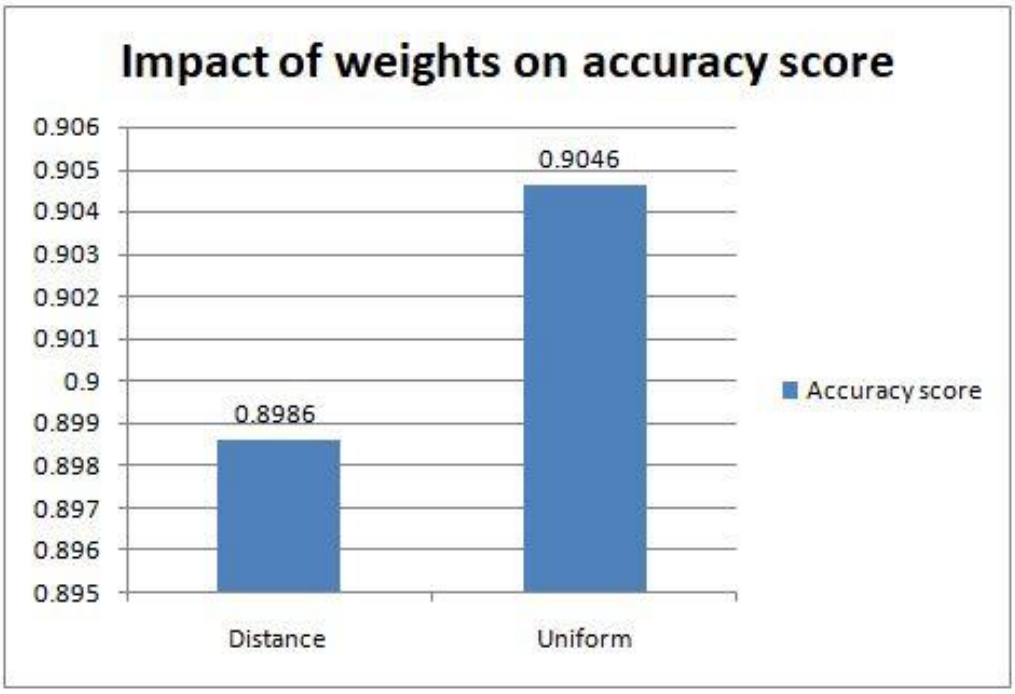



Figure 9

Figure 10 represents the impact of choosing the number of neighbors on accuracy score.

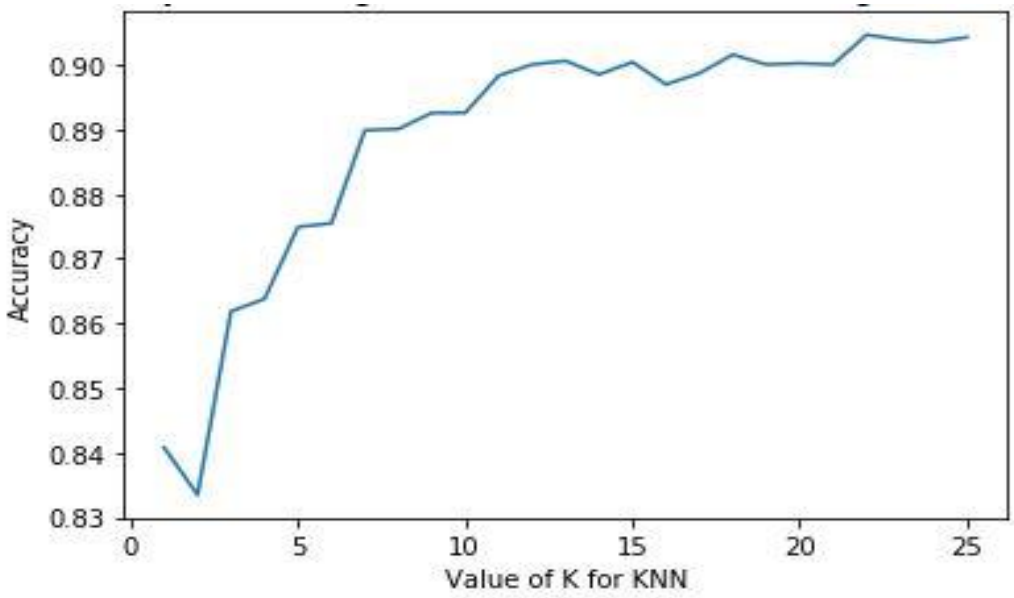



Figure 10

Hence, after checking the impact of choosing different values of parameters on accuracy score, we find out that best parameters are:

Distance measure: **'euclidean'**

Number of neighbors: **22**

Weights: **'uniform'**

## Methodology for data training:

1- Input: Data set and best parameters for k-NN.
2- Split the data into training set (75%) and testing set (25%). We use 'startify' so that the train and test sets have approximately the same percentage of samples of each target class as the complete set.
3- Built the k-NN classifier by feeding it with the training data and best parameters.
4- Train the model and store the model on the disk.
5- Plotting of confusion matrix.
6- Calculate accuracy, precision and recall for training and testing data.

## Results:

For the better evaluation of the performance of k-NN algorithm, we plot confusion matrix as shown in the Figure 11.

As it is a multi-classification problem, it is a little bit different from binary classification. For example: ‘L1’ row shows that out of total samples in test data where data is labeled as ‘L1’, 92 instances were correctly classify, while 4 samples were wrongly predicted as ‘L9’.

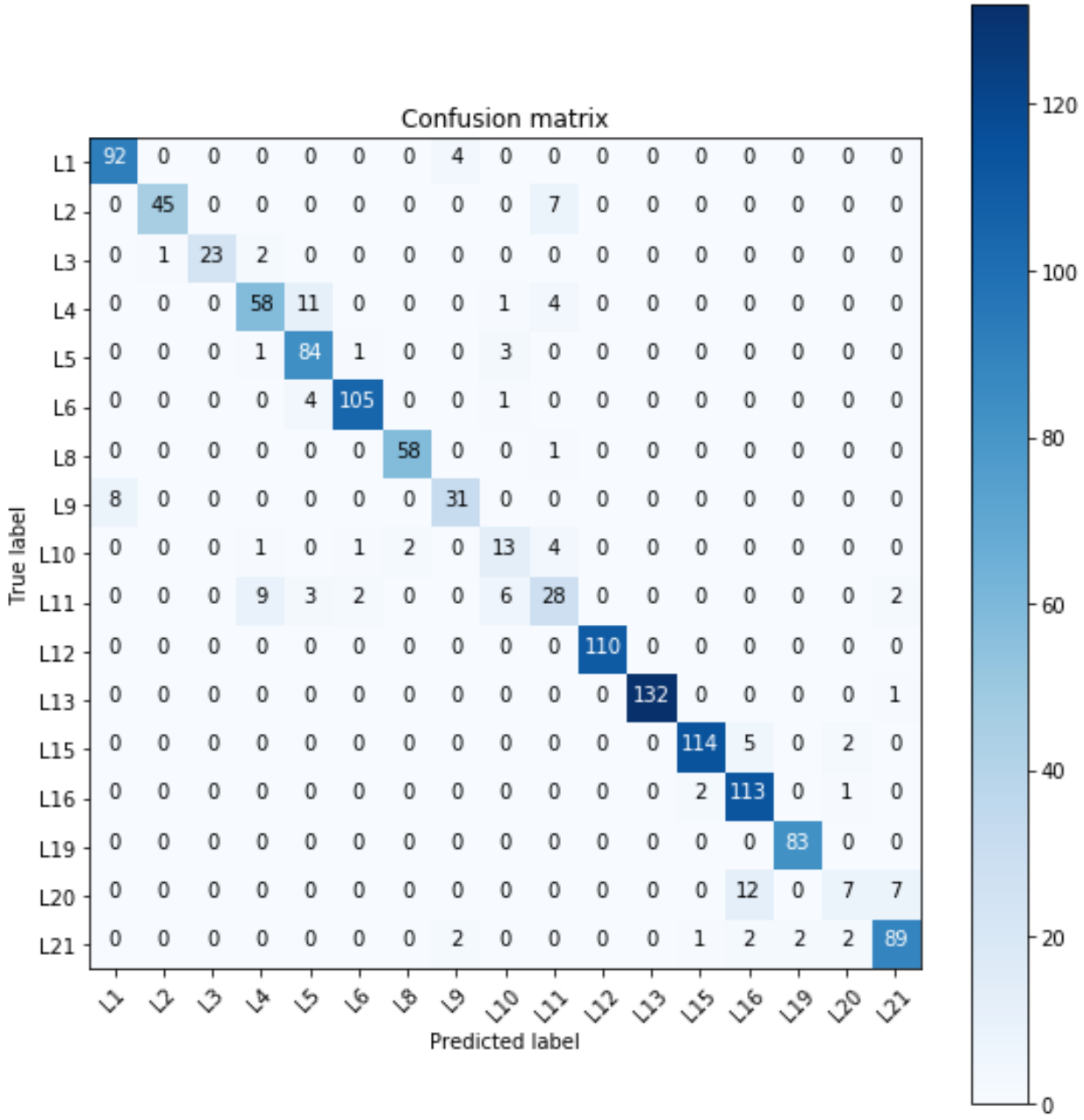



Figure 11

Here is the accuracy, precision and recall graph for k-NN.

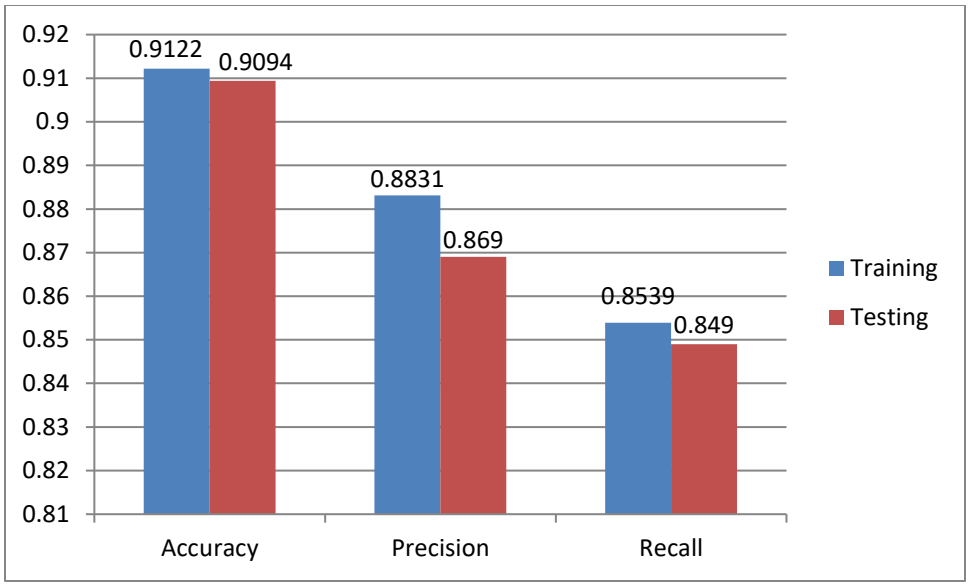



Figure 12

## Random Forest Classifier:

Random forest is one of the most powerful supervised machine learning algorithms. It is capable of performing both classification and regression but here we use this algorithm for multi-classification. For this purpose, it creates multiple decision trees in the form of forest by selecting random samples from dataset for each decision tree.

### Why we are using random forest?

- The over fitting problem will never come when we use the random forest algorithm in any classification problem.
- The random forest algorithm can be used to identify the most important features out of the available features from the training dataset. It means we don't have need to normalize the features.
- Tree models can handle both continuous and categorical data and can model nonlinear relationships

## Methodology:

1. Present a dataset containing number of training samples characterized by multiple features and target feature.
2. Random forest selects ‘m’ features from the total features randomly.
3. Select the root node using best split and split these nodes into daughter nodes.
4. Repeat above steps to form various decision trees until maximum number of nodes reached.
5. Create leaf nodes which are further used to predict new query instances.
6. Make and initialize a tensorflow session to train the model.
7. Calculate accuracy of training and testing model.
8. Predict the outcome using these decision trees.
9. Measure the outcome or target label predicted by each tree. The target with the highest vote is considered as the final prediction of the random forest algorithm.

**Training accuracy: 95.3%**

**Testing accuracy: 91.3%**

The reason of why training accuracy is greater than testing accuracy is that the model over fits the data.

## Description of Hyper-Parameters:

**n_estimators:** number of trees in the forest.

- Higher the number of trees, higher is the prediction accuracy.
- Value =10

**max_features:** maximum number of features considered for the best split.

- We took all the features in a single step.

**max_node:** maximum number of nodes in which dataset can split

- Value=1000

**Batch_size:** It indicates the number of samples per batch

- Vlaue=1024

**random_state:** defines random selection to compare between different models.

- Value = 1

Except these values, we also trained our model on different values of hyper parameters and plot their accuracy response in the form of graphs as shown below:

## Graphs:

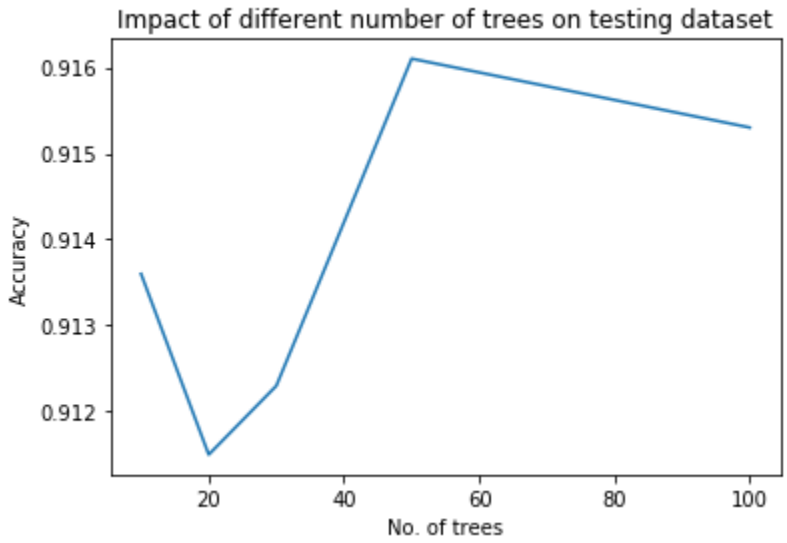



Figure 13

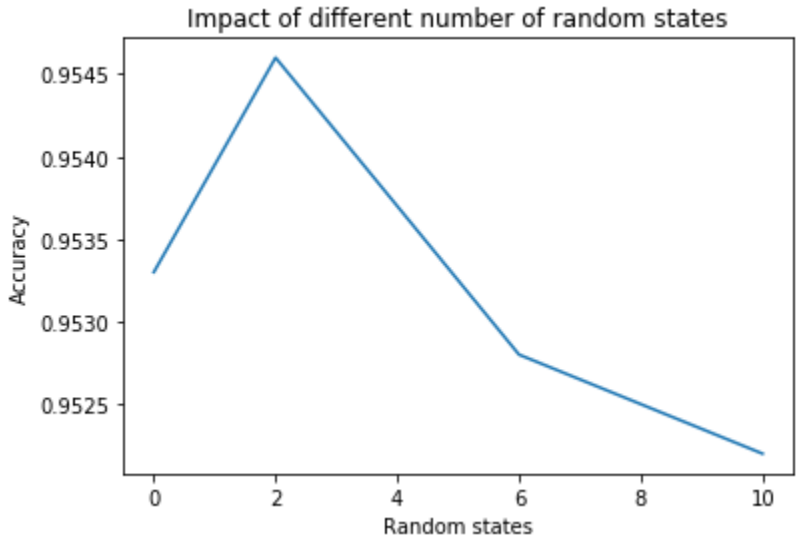



Figure 14

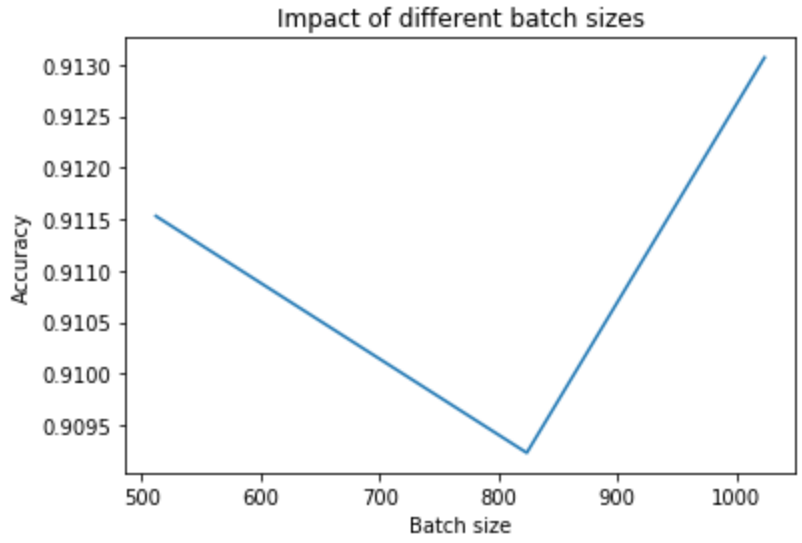



Figure 15

**Feature Importance graph**

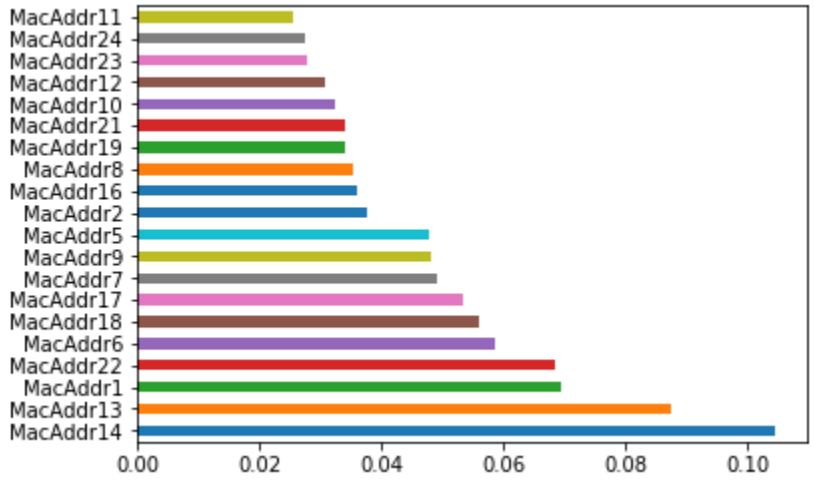



Figure 16

## Artificial Neural Network (ANN):

ANN is a powerful deep learning algorithm inspired by biological structure of neurons. Information flows in the hierarchical form. We have used 2-layered neural network for our multi-classification problem because in many internet sources, we find out that 2-layered network is sufficient. The input layer consists of input neurons. Those neurons transmit data to

the next layer, which in turn sends the output neurons to the output layer. Figure 17 shows the structure of Neural Network.

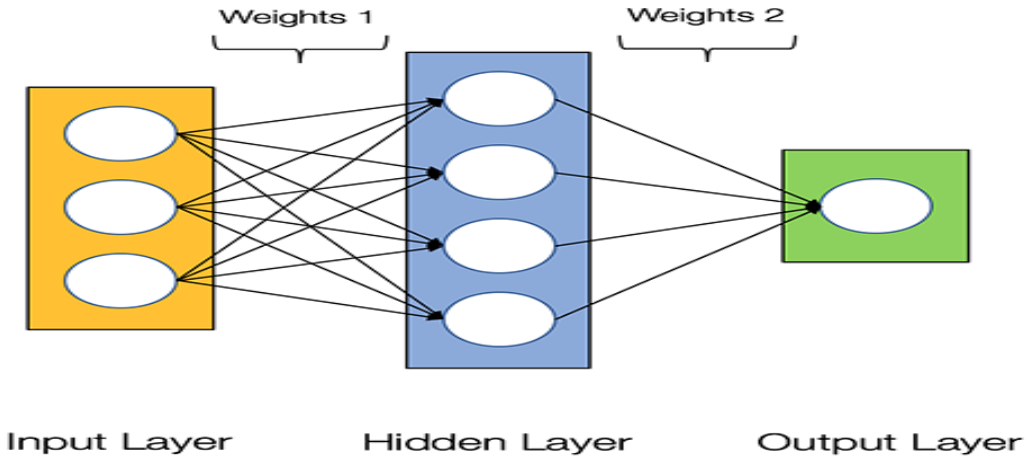



Figure 17

The basic parameters for ANN algorithm are:

1- Batch size and Epochs:
   Batch size tells that how many samples are used to calculate the error. Number of epochs controls the number of complete passes through training dataset. One epoch is comprised of feed forward and back propagation. In feed forward, we calculate the predicted outputs and then calculate the error using any error measure formula. Then, in back propagation, this error is used to update the weights.
2- Optimization Algorithms:
   There are many optimization algorithms like 'SGD', 'Adam', 'Nadam' etc. We can use any of the optimization algorithms and it is used for updating the weights.
3- Weight Initialization Methods:
   There are many weight initialization methods like 'zero', 'normal', 'uniform'. We can use any of the weight initialization methods and it is used for initializing the weight vector. 'zero' method initializes the weight vector with all zeros.
4- Activation Function:
   Each neuron has an activation function that gets the input from previous layer which combines with associated weights and then passes to the activation function that produce output and then this output pass to the next layer.
5- Number of neurons:

Each layer has many number of neurons as you can see in the Figure 17.

## Hyper-parameters tuning:

Before we implement ANN, we need to find the best values for parameters at which accuracy score is maximum. For this purpose, we use GridsearchCV function to test for all possible combinations. Here are the results:

This figure shows the impact of batch sizes on accuracy score.

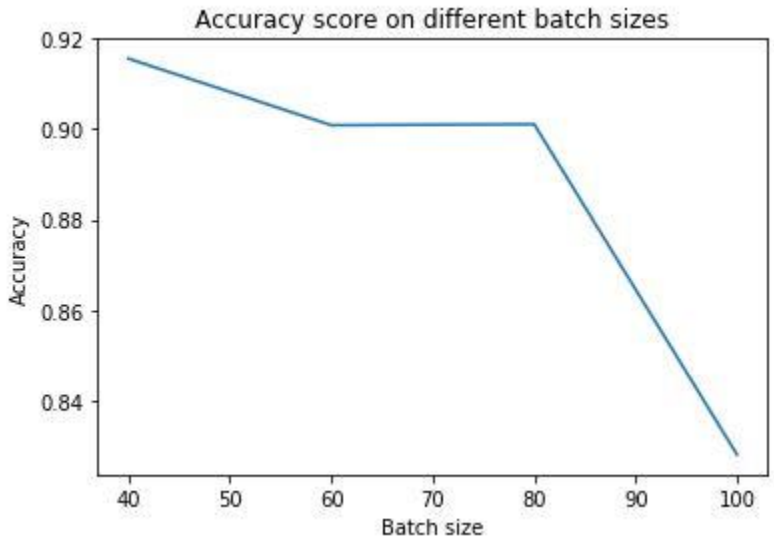



Figure 18

This figure shows the impact of epochs on accuracy score.

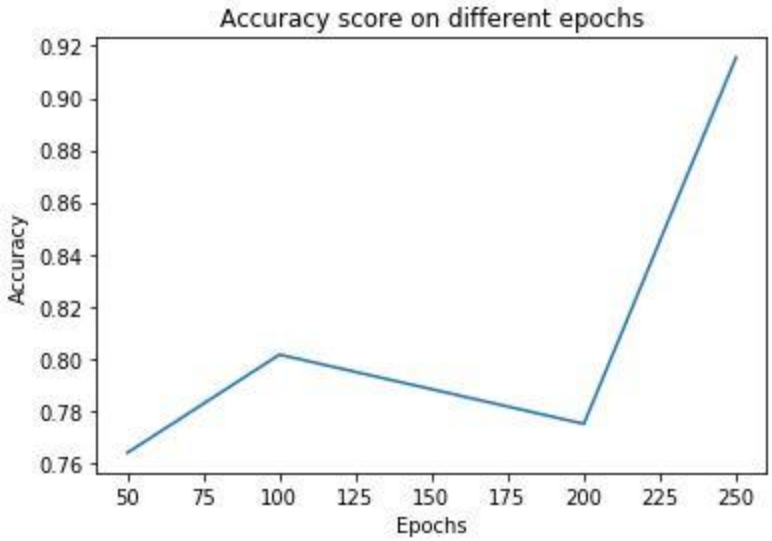



Figure 19

This figure shows the impact of optimizations algorithms on accuracy score.

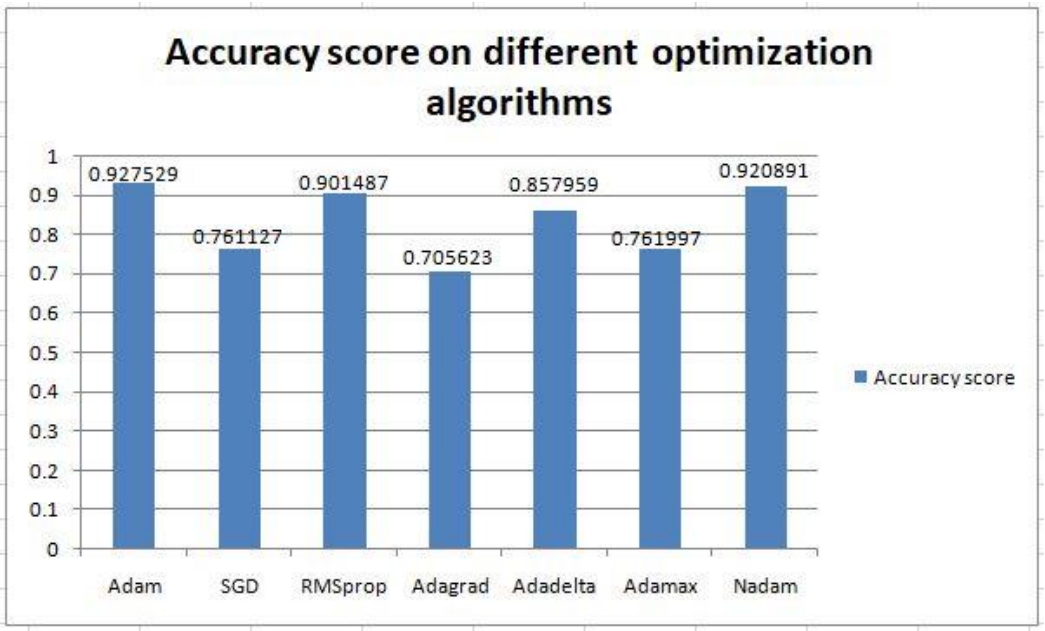



Figure 20

This figure shows the impact of weight initialization methods on accuracy score.

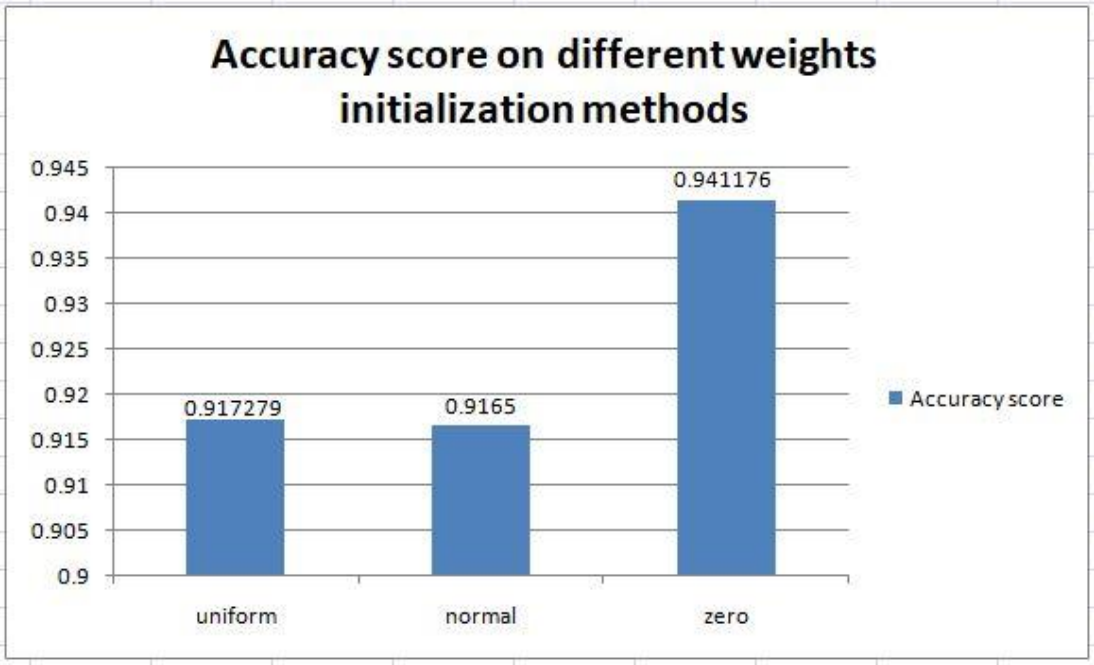



Figure 21

This figure shows the impact of number of neurons on accuracy score.

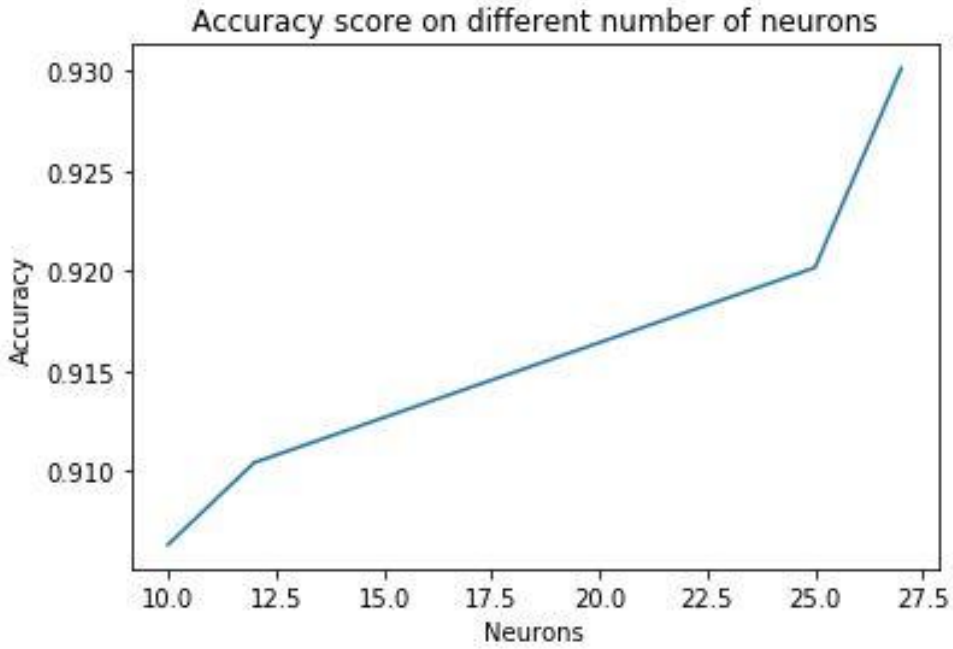



Figure 22

We also tune activation functions { 'relu', 'tanh', 'sigmoid', 'linear'}. All of them gave same accuracy score.

Hence, after checking the impact of choosing different values of parameters on accuracy score, we find out that best combination has following values:

Batch size: **40**

Epochs: **250**

Optimization Algorithm: **'Adam'**

Weight Initialization Methods: **'uniform'**

Activation Function: **'relu' for hidden layer and 'sigmoid' for output layer**

Number of Neurons for hidden layer: **28**

## Methodology for data training:

1- Input: Data set and best parameters for ANN.
2- Split the data into training set (75%) and testing set (25%). We use 'startify' so that the train and test sets have approximately the same percentage of samples of each target class as the complete set.
3- We apply label encoder to the output vector that encode classes that are in string format into integers. Then we apply to_categorical that convert a class vector into binary class matrix.
4- Built the 2-layered ANN classifier by feeding it with the training data and best parameters.
5- Train the model and store the model on the disk.
6- Plotting of confusion matrix.
7- Calculate accuracy, precision and recall for training and testing data.

## Results:

For the better evaluation of the performance of ANN algorithm, we plot confusion matrix as shown in the Figure 23.

As we use label encoder, so it encode classes ranges from 0 to 16. Here is the actual sequence, it means ‘L1’ encode to 0, ‘L10’ encode to 1 and so on:

['L1','L10','L11','L12','L13','L15','L16','L19','L2','L20','L21','L3','L4','L5','L6','L8','L9']

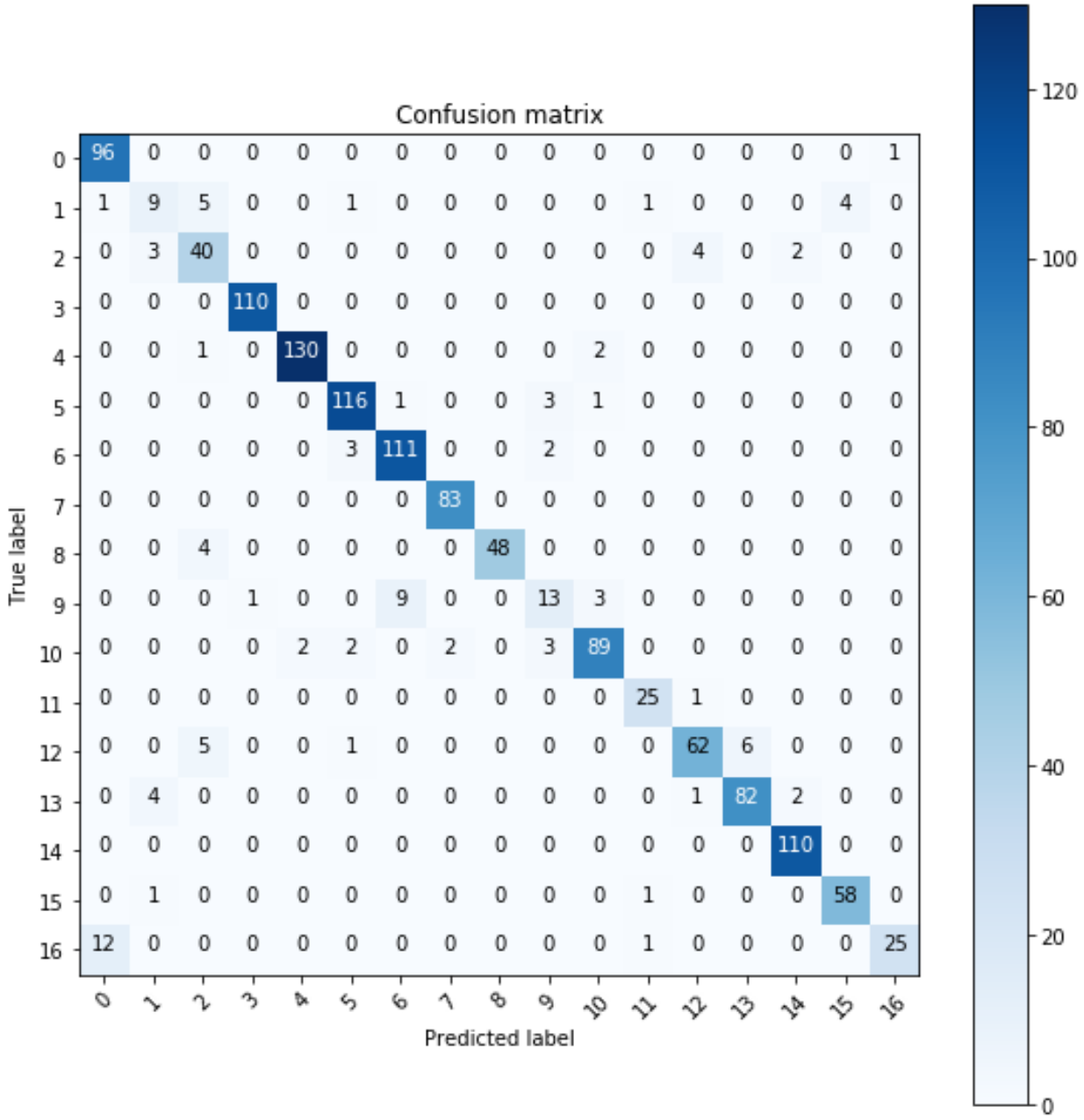



Figure 23

Here is the accuracy, precision and recall graph for ANN.

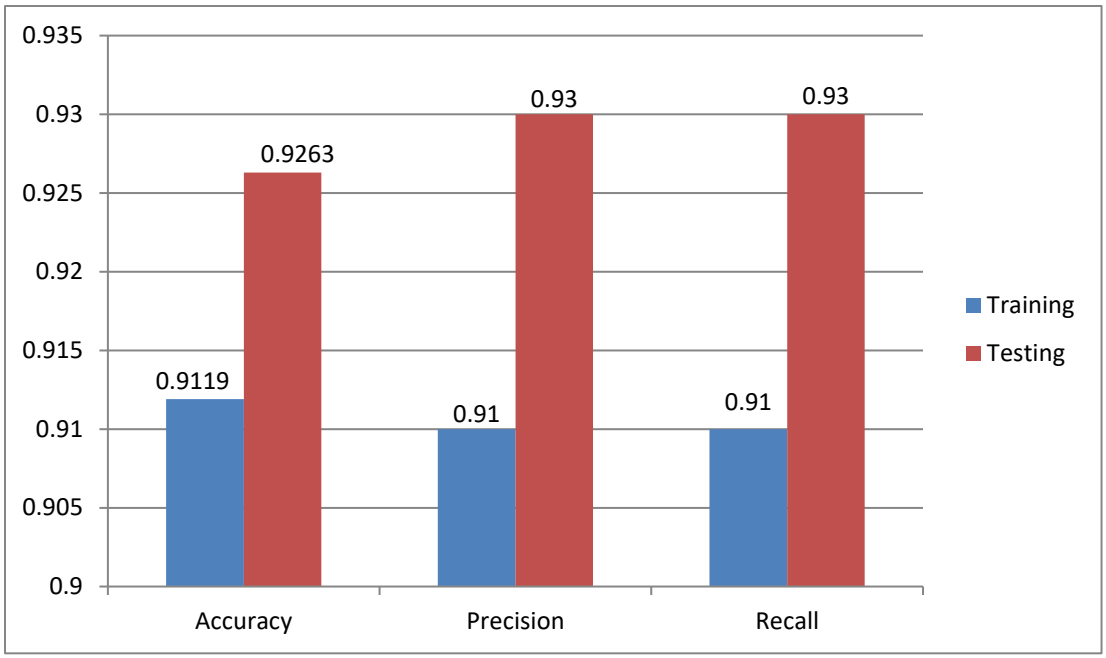



Figure 24

# Conclusion:

We conclude that on the basis of accuracy, precision and recall score for k-NN, Random forest and ANN. It is clear that ANN has highest accuracy **(92.63%)**, precision **(93%)** and recall **(93%)**. Hence**, ANN is our selected model.**