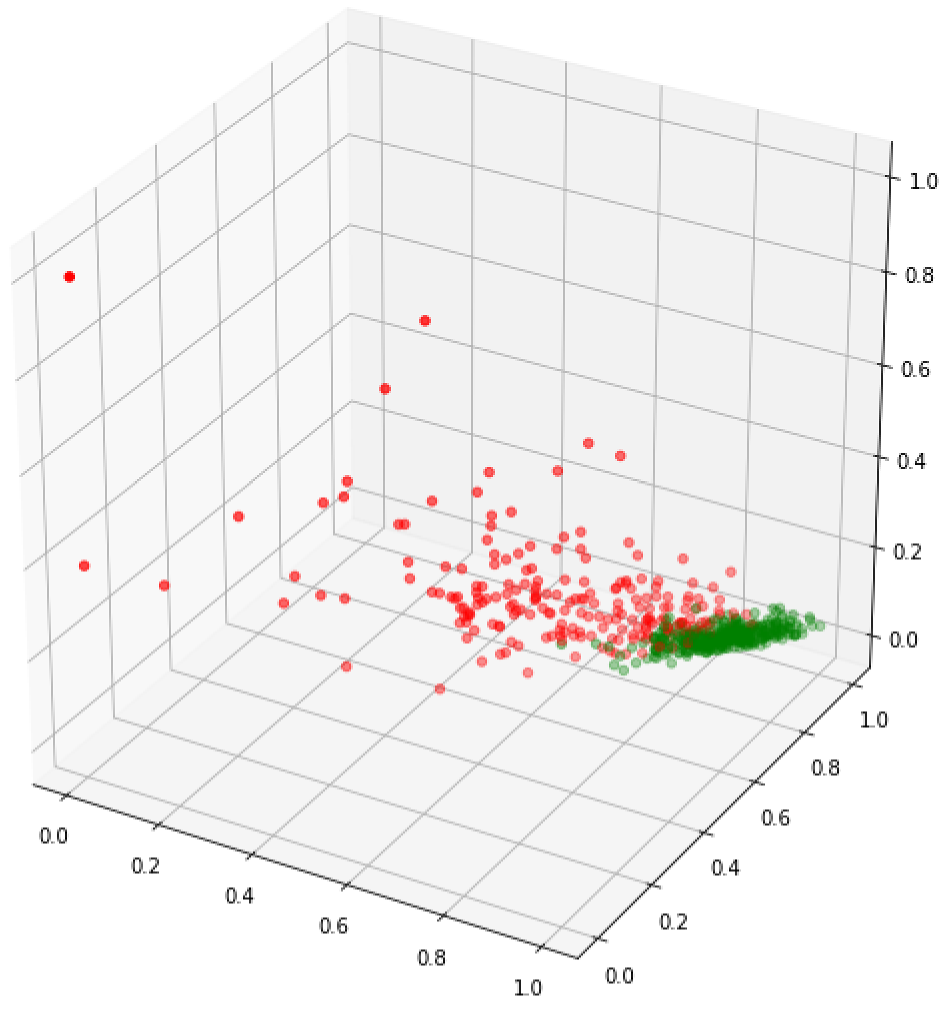
ML Practical Project

Shubhankar Poundrik - ufh6ft
Nakshatra Yalagach - jhj5dh
Anushruti - rba7cb

• Data Exploration

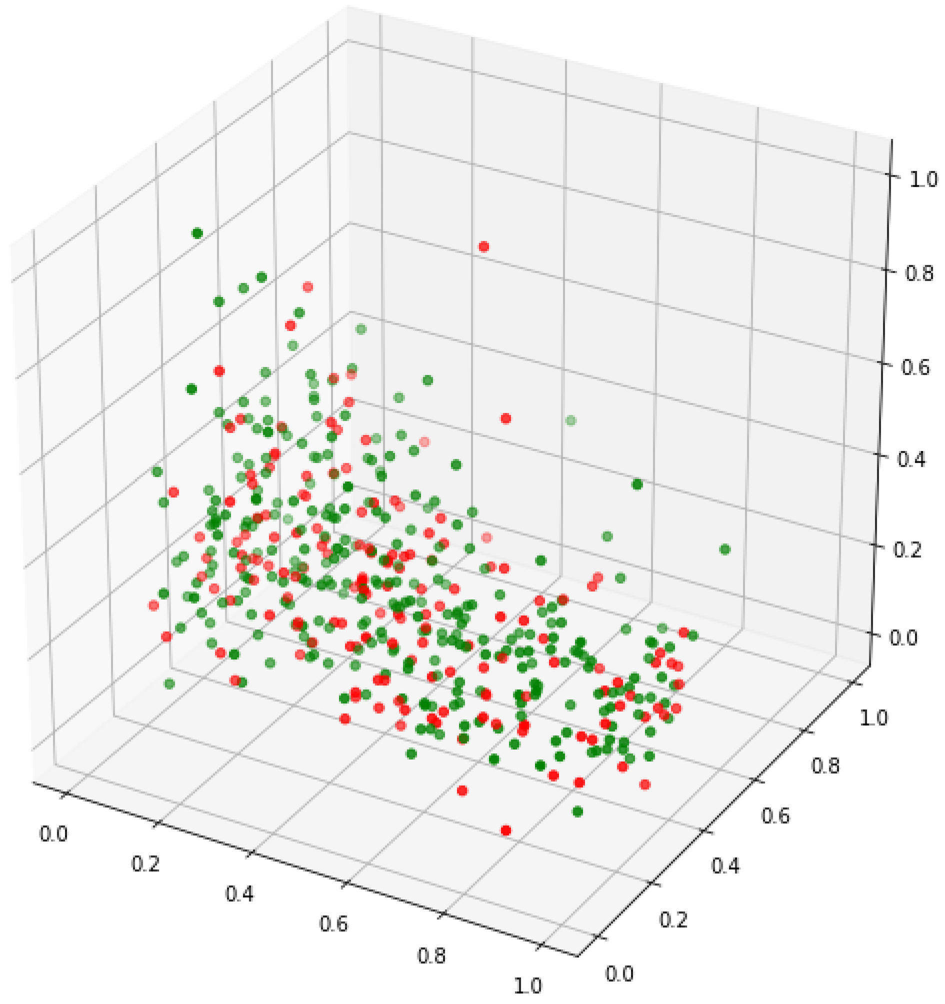
We refer to project3_dataset1 as dataset1 and project3_dataset2 as dataset2

PCA with 3 principal components for dataset1



We can see that the classes seem to be seperable and we can expect high accuracy from our models.

PCA with 3 principal components for dataset2



We can see that the classes do NOT seem to be seperable and we CANNOT expect high accuracy from our models.

● Logistic Regression

Logistic regression is a type of statistical model that is used to predict the probability of an outcome. It is often used in binary classification tasks, where the goal is to predict whether an individual belongs to one of two classes, such as "yes" or "no".

The algorithm works by using a mathematical function to map input data to a set of numerical values called weights. The algorithm fits the given data to a sigmoid / logistic function. These weights represent the strength of the relationship between the input data and the outcome. The algorithm then uses these weights to calculate the probability that an individual belongs to one of the two classes.

To train the algorithm, the input data and corresponding outcomes are provided to the algorithm. The algorithm then uses an optimization method, such as gradient descent, to adjust the weights in order to minimize the error between the predicted probabilities and the true outcomes. This process is repeated until the error is sufficiently small, at which point the algorithm is considered to be trained.

Once the algorithm is trained, it can be used to predict the probability of an outcome for new input data. This is done by applying the trained weights to the input data and calculating the probability of the individual belonging to one of the two classes.

Overall, the logistic regression algorithm is a powerful tool for predicting probabilities in binary classification tasks. It is widely used in a variety of applications, such as in medical research and marketing.

Logistic regression involves estimating a logistic model's parameters. In Logistic Regression, we try to fit a logistic curve (a sigmoid function) to the given data. This involves the estimation of a single parameter that has a dimensionality greater than the input data points by 1.

The equation for logistic regression can be written as,

$$p(x) = 1/(1 + e^{-\beta^T x'}), \quad where\ \ x' = [1 \quad x^T]^T$$

Here the parameter to estimate is beta.

The training to find the best parameter value is done using gradient descent.

For Logistic Regression, no hyperparameter tuning is required.

We use a 80%-20% train-test split.

10 fold Cross-validation is employed while training the Logistic Regression.

data1 shape = (569, 31)

```
data2 shape = (462, 10)

============ Dataset 1 ============

Distribution: Counter({0: 357, 1: 212})
X.shape, y.shape = (569, 31), (569,)
Fold: 0. Accuracy = 0.978, precision = 1.000, recall = 0.941, f1 = 0.970, AUC = 0.971
Fold: 1. Accuracy = 0.978, precision = 1.000, recall = 0.941, f1 = 0.970, AUC = 0.971
Fold: 2. Accuracy = 0.913, precision = 1.000, recall = 0.765, f1 = 0.867, AUC = 0.882
Fold: 3. Accuracy = 0.978, precision = 1.000, recall = 0.941, f1 = 0.970, AUC = 0.971
Fold: 4. Accuracy = 0.978, precision = 1.000, recall = 0.941, f1 = 0.970, AUC = 0.971
Fold: 5. Accuracy = 0.978, precision = 1.000, recall = 0.941, f1 = 0.970, AUC = 0.971
Fold: 6. Accuracy = 0.956, precision = 1.000, recall = 0.875, f1 = 0.933, AUC = 0.938
Fold: 7. Accuracy = 0.933, precision = 1.000, recall = 0.812, f1 = 0.897, AUC = 0.906
Fold: 8. Accuracy = 0.956, precision = 1.000, recall = 0.875, f1 = 0.933, AUC = 0.938
Fold: 9. Accuracy = 1.000, precision = 1.000, recall = 1.000, f1 = 1.000, AUC = 1.000
Average across folds:
Accuracy = 0.965, precision = 1.000, recall = 0.903, f1 = 0.948, AUC = 0.952
Test set results:
Accuracy = 0.956, precision = 0.956, recall = 0.935, f1 = 0.945, AUC = 0.953
==================================================

============ Dataset 2 ============

Distribution: Counter({0: 302, 1: 160})
X.shape, y.shape = (462, 10), (462,)
Fold: 0. Accuracy = 0.811, precision = 0.800, recall = 0.615, f1 = 0.696, AUC = 0.766
Fold: 1. Accuracy = 0.676, precision = 0.538, recall = 0.538, f1 = 0.538, AUC = 0.644
Fold: 2. Accuracy = 0.730, precision = 0.667, recall = 0.462, f1 = 0.545, AUC = 0.668
Fold: 3. Accuracy = 0.676, precision = 0.571, recall = 0.308, f1 = 0.400, AUC = 0.591
Fold: 4. Accuracy = 0.595, precision = 0.375, recall = 0.231, f1 = 0.286, AUC = 0.511
Fold: 5. Accuracy = 0.757, precision = 0.692, recall = 0.643, f1 = 0.667, AUC = 0.734
Fold: 6. Accuracy = 0.757, precision = 0.727, recall = 0.571, f1 = 0.640, AUC = 0.720
Fold: 7. Accuracy = 0.811, precision = 0.769, recall = 0.714, f1 = 0.741, AUC = 0.792
Fold: 8. Accuracy = 0.676, precision = 0.583, recall = 0.500, f1 = 0.538, AUC = 0.641
Fold: 9. Accuracy = 0.806, precision = 0.800, recall = 0.615, f1 = 0.696, AUC = 0.764
Average across folds:
Accuracy = 0.729, precision = 0.652, recall = 0.520, f1 = 0.575, AUC = 0.683
Test set results:
Accuracy = 0.785, precision = 0.615, recall = 0.615, f1 = 0.615, AUC = 0.733
```

● K Nearest Neighbors

K-nearest neighbors (KNN) is a classification algorithm that is used in supervised learning. It is a non-parametric and lazy learning algorithm, meaning it does not make any assumptions about the underlying data distribution and simply stores the training data. The KNN algorithm works by finding the K observations in the training data that are nearest to the point for which we want to make a prediction and taking a majority vote. It then uses the class labels of those K training points to make a prediction for the new point. The KNN algorithm is simple and easy to implement, but it can be computationally expensive because it requires a distance calculation for each point in the training data. Additionally, the algorithm can be sensitive to the choice of K, so it is important to choose an appropriate value for K.

============ Dataset 1 ============
Plotting K value Vs Accuracy

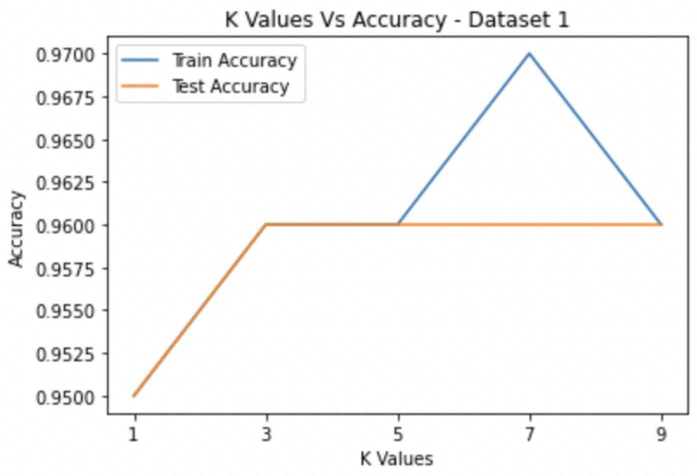
Sample output for 1 fold

```
X.shape, y.shape = (569, 30), (569,)
Fold: 0. Accuracy = 0.913, precision = 0.882, recall = 0.882, f1 = 0.882, AUC = 0.907
Fold: 1. Accuracy = 0.913, precision = 0.842, recall = 0.941, f1 = 0.889, AUC = 0.919
Fold: 2. Accuracy = 0.978, precision = 1.000, recall = 0.941, f1 = 0.970, AUC = 0.971
Fold: 3. Accuracy = 0.957, precision = 0.941, recall = 0.941, f1 = 0.941, AUC = 0.953
Fold: 4. Accuracy = 1.000, precision = 1.000, recall = 1.000, f1 = 1.000, AUC = 1.000
Fold: 5. Accuracy = 0.911, precision = 0.882, recall = 0.882, f1 = 0.882, AUC = 0.905
Fold: 6. Accuracy = 0.956, precision = 0.941, recall = 0.941, f1 = 0.941, AUC = 0.953
Fold: 7. Accuracy = 0.978, precision = 1.000, recall = 0.941, f1 = 0.970, AUC = 0.971
Fold: 8. Accuracy = 0.956, precision = 0.941, recall = 0.941, f1 = 0.941, AUC = 0.953
Fold: 9. Accuracy = 0.978, precision = 0.941, recall = 1.000, f1 = 0.970, AUC = 0.983
Accuracy = 0.954, precision = 0.937, recall = 0.941, f1 = 0.939, AUC = 0.951
```

Training Results:
K values: [1, 3, 5, 7, 9]
Train Accuracy: [0.95, 0.96, 0.96, 0.97, 0.96]

Testing Results:
K values: [1, 3, 5, 7, 9]
Test Accuracy: [0.95, 0.96, 0.96, 0.96, 0.96]

Best K value: 3,5,7,9 with test accuracy 0.96



From the above graph we can see that the train accuracy increases till k=7 and reduces after that. Test accuracy peaks at 3 and then saturates. We can also observe that the train accuracy is greater than test accuracy. Hence the model is overfitting.

Plotting Distance Metrics Vs Accuracy

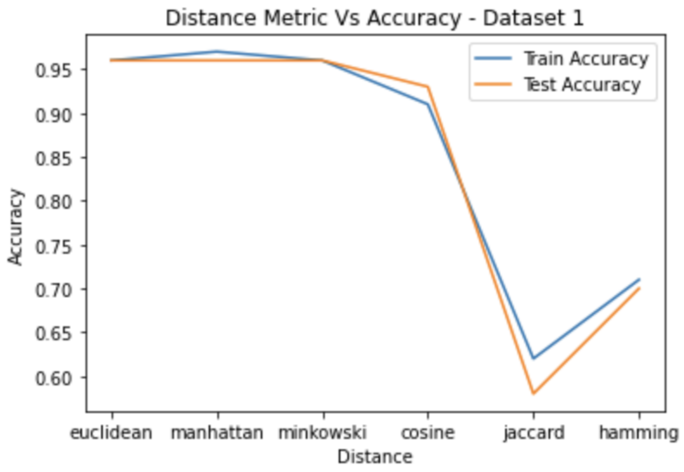
Sample output for 1 fold

```
X.shape, y.shape = (569, 30), (569,)
Fold: 0. Accuracy = 0.913, precision = 1.000, recall = 0.765, f1 = 0.867, AUC = 0.882
Fold: 1. Accuracy = 0.978, precision = 1.000, recall = 0.941, f1 = 0.970, AUC = 0.971
Fold: 2. Accuracy = 0.978, precision = 1.000, recall = 0.941, f1 = 0.970, AUC = 0.971
Fold: 3. Accuracy = 0.935, precision = 0.938, recall = 0.882, f1 = 0.909, AUC = 0.924
Fold: 4. Accuracy = 0.978, precision = 1.000, recall = 0.938, f1 = 0.968, AUC = 0.969
Fold: 5. Accuracy = 0.978, precision = 0.941, recall = 1.000, f1 = 0.970, AUC = 0.983
Fold: 6. Accuracy = 0.978, precision = 0.941, recall = 1.000, f1 = 0.970, AUC = 0.983
Fold: 7. Accuracy = 0.956, precision = 1.000, recall = 0.875, f1 = 0.933, AUC = 0.938
Fold: 8. Accuracy = 1.000, precision = 1.000, recall = 1.000, f1 = 1.000, AUC = 1.000
Fold: 9. Accuracy = 0.956, precision = 1.000, recall = 0.875, f1 = 0.933, AUC = 0.938
Accuracy = 0.965, precision = 0.982, recall = 0.922, f1 = 0.949, AUC = 0.956
```

Training Results:
Distance Metric ['euclidean', 'manhattan', 'minkowski', 'cosine', 'jaccard', 'hamming']
Train Accuracy: [0.96, 0.97, 0.96, 0.91, 0.62, 0.71]

Testing Results:
Distance Metric: ['euclidean', 'manhattan', 'minkowski', 'cosine', 'jaccard', 'hamming']
Test Accuracy: [0.96, 0.96, 0.96, 0.93, 0.58, 0.7]

Best Distance Metric: Euclidean, Manhattan, Minkowski with test accuracy of 0.96



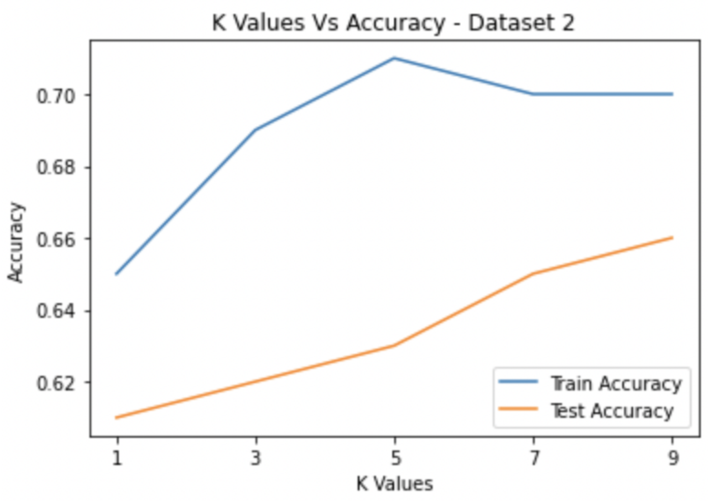
From the above graph we can observe that both the training and testing results follow the same trend. We can also notice that the train accuracy peaks for manhattan metric and test accuracy peaks for euclidean, manhattan and minikowski

===

============ Dataset 2 ============
Plotting K value Vs Accuracy

Sample output for 1 fold

```
X.shape, y.shape = (462, 9), (462,)
Fold: 0. Accuracy = 0.649, precision = 0.500, recall = 0.462, f1 = 0.480, AUC = 0.606
Fold: 1. Accuracy = 0.730, precision = 0.615, recall = 0.615, f1 = 0.615, AUC = 0.704
Fold: 2. Accuracy = 0.649, precision = 0.500, recall = 0.538, f1 = 0.519, AUC = 0.623
Fold: 3. Accuracy = 0.730, precision = 0.636, recall = 0.538, f1 = 0.583, AUC = 0.686
Fold: 4. Accuracy = 0.486, precision = 0.250, recall = 0.231, f1 = 0.240, AUC = 0.428
Fold: 5. Accuracy = 0.622, precision = 0.462, recall = 0.462, f1 = 0.462, AUC = 0.585
Fold: 6. Accuracy = 0.730, precision = 0.600, recall = 0.692, f1 = 0.643, AUC = 0.721
Fold: 7. Accuracy = 0.622, precision = 0.455, recall = 0.385, f1 = 0.417, AUC = 0.567
Fold: 8. Accuracy = 0.595, precision = 0.429, recall = 0.462, f1 = 0.444, AUC = 0.564
Fold: 9. Accuracy = 0.722, precision = 0.600, recall = 0.500, f1 = 0.545, AUC = 0.667
Accuracy = 0.653, precision = 0.505, recall = 0.488, f1 = 0.495, AUC = 0.615

Training Results:
K values: [1, 3, 5, 7, 9]
Train Accuracy: [0.65, 0.69, 0.71, 0.7, 0.7]

Testing Results:
K values: [1, 3, 5, 7, 9]
Test Accuracy: [0.61, 0.62, 0.63, 0.65, 0.66]

Best K value: 9 with test accuracy 0.66
```



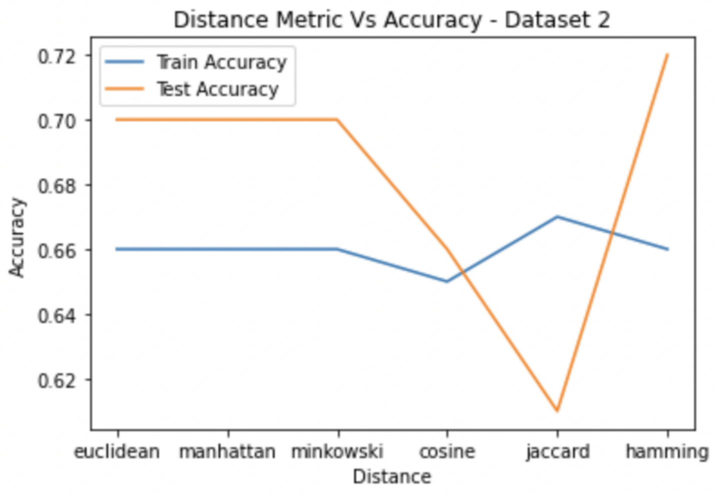
From the above graph we can observe that the train accuracy is higher than the test accuracy. Hence the model is overfitting. The train accuracy peaks at K=5 and the test accuracy peaks at k=9

Plotting Distance Metrics Vs Accuracy

Sample output for 1 fold

```
X.shape, y.shape = (462, 9), (462,)
Fold: 0. Accuracy = 0.595, precision = 0.375, recall = 0.231, f1 = 0.286, AUC = 0.511
Fold: 1. Accuracy = 0.595, precision = 0.400, recall = 0.308, f1 = 0.348, AUC = 0.529
Fold: 2. Accuracy = 0.676, precision = 0.571, recall = 0.308, f1 = 0.400, AUC = 0.591
Fold: 3. Accuracy = 0.595, precision = 0.000, recall = 0.000, f1 = 0.000, AUC = 0.458
Fold: 4. Accuracy = 0.649, precision = 0.500, recall = 0.385, f1 = 0.435, AUC = 0.588
Fold: 5. Accuracy = 0.784, precision = 0.727, recall = 0.615, f1 = 0.667, AUC = 0.745
Fold: 6. Accuracy = 0.676, precision = 0.667, recall = 0.154, f1 = 0.250, AUC = 0.556
Fold: 7. Accuracy = 0.730, precision = 0.636, recall = 0.538, f1 = 0.583, AUC = 0.686
Fold: 8. Accuracy = 0.676, precision = 0.556, recall = 0.385, f1 = 0.455, AUC = 0.609
Fold: 9. Accuracy = 0.667, precision = 0.500, recall = 0.250, f1 = 0.333, AUC = 0.562
Accuracy = 0.664, precision = 0.493, recall = 0.317, f1 = 0.376, AUC = 0.584

Training Results:
Distance Metric ['euclidean', 'manhattan', 'minkowski', 'cosine', 'jaccard', 'hamming']
Train Accuracy: [0.66, 0.66, 0.66, 0.65, 0.67, 0.66]

Testing Results:
Distance Metric: ['euclidean', 'manhattan', 'minkowski', 'cosine', 'jaccard', 'hamming']
Test Accuracy: [0.7, 0.7, 0.7, 0.66, 0.61, 0.72]
```

Best Distance Metric: Hamming with test accuracy of 0.72



From the above graph we can observe that test accuracy higher than the train accuracy for some metrics. Hence the model is underfitting for some metrics. The train results are maximum for jaccard metric while the test results are highest for hamming metric.

● Boosting

Boosting is a type of ensemble learning algorithm that involves training a sequence of weak models in a way that each model tries to correct the mistakes of the previous model. A new dataset is created from the previous dataset to train each model and the data points previously misclassified are given more weight in the new dataset. This way, the next model learns to correctly predict the data points wrongly classified by the previous model. The goal is to combine the weak models into a single strong model that is able to make accurate predictions on new data. Boosting algorithms typically use decision trees as the weak models, but other types of models can also be used.

============ Dataset 1 ============
Plotting Max Depth Vs Accuracy

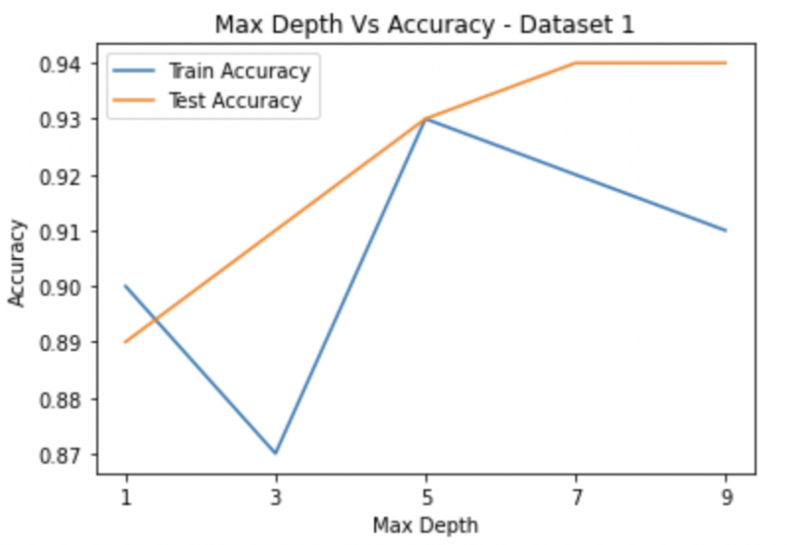
Sample output for 1 fold

```
X.shape, y.shape = (569, 30), (569,)
Train accuracy = 1.0
Test accuracy = 0.9347826086956522
Fold: 0. Accuracy = 0.935, precision = 0.941, recall = 0.889, f1 = 0.914, AUC = 0.927
Train accuracy = 1.0
Test accuracy = 0.8913043478260869
Fold: 1. Accuracy = 0.891, precision = 0.882, recall = 0.833, f1 = 0.857, AUC = 0.881
Train accuracy = 1.0
Test accuracy = 0.8913043478260869
Fold: 2. Accuracy = 0.891, precision = 0.882, recall = 0.833, f1 = 0.857, AUC = 0.881
Train accuracy = 1.0
Test accuracy = 0.9347826086956522
Fold: 3. Accuracy = 0.935, precision = 0.938, recall = 0.882, f1 = 0.909, AUC = 0.924
Train accuracy = 1.0
Test accuracy = 0.8913043478260869
Fold: 4. Accuracy = 0.891, precision = 0.929, recall = 0.765, f1 = 0.839, AUC = 0.865
Train accuracy = 1.0
Test accuracy = 0.8666666666666667
Fold: 5. Accuracy = 0.867, precision = 0.867, recall = 0.765, f1 = 0.812, AUC = 0.847
Train accuracy = 1.0
Test accuracy = 0.9555555555555556
Fold: 6. Accuracy = 0.956, precision = 1.000, recall = 0.882, f1 = 0.938, AUC = 0.941
Train accuracy = 1.0
Test accuracy = 0.8888888888888888
Fold: 7. Accuracy = 0.889, precision = 0.875, recall = 0.824, f1 = 0.848, AUC = 0.876
Train accuracy = 1.0
Test accuracy = 0.8888888888888888
Fold: 8. Accuracy = 0.889, precision = 0.875, recall = 0.824, f1 = 0.848, AUC = 0.876
Train accuracy = 1.0
Test accuracy = 0.8888888888888888
Fold: 9. Accuracy = 0.889, precision = 0.875, recall = 0.824, f1 = 0.848, AUC = 0.876
Accuracy = 0.903, precision = 0.906, recall = 0.832, f1 = 0.867, AUC = 0.889
```

Training Results:
Max Depth: [1, 3, 5, 7, 9]
Accuracy: [0.9, 0.87, 0.93, 0.92, 0.91]

Testing Results:
Max Depth: [1, 3, 5, 7, 9]
Test Accuracy: [0.89, 0.91, 0.93, 0.94, 0.94]

Best Max Depth: 7 an 9 with a test accuracy of 0.94



From the above graph we can observe that the test accuracy is higher than train accuracy. Hence the model is undrfitting. Train accuracy is highest for max depth = 5 and the test accuracy is highest for max depth = 7 and 9

============ Dataset 2 ============
Plotting Max Depth Vs Accuracy

Sample output for 1 fold

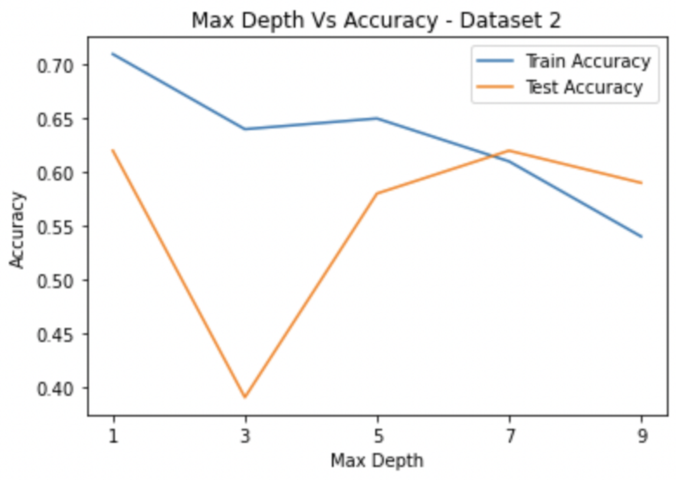
```
X.shape, y.shape = (462, 9), (462,)
Train accuracy = 1.0
Test accuracy = 0.8108108108108109
Fold: 0. Accuracy = 0.811, precision = 0.778, recall = 0.583, f1 = 0.667, AUC = 0.752
Train accuracy = 1.0
Test accuracy = 0.6756756756756757
Fold: 1. Accuracy = 0.676, precision = 0.500, recall = 0.333, f1 = 0.400, AUC = 0.587
Train accuracy = 1.0
Test accuracy = 0.8648648648648649
Fold: 2. Accuracy = 0.865, precision = 0.818, recall = 0.750, f1 = 0.783, AUC = 0.835
Train accuracy = 1.0
```

```
Test accuracy = 0.6756756756756757
Fold: 3. Accuracy = 0.676, precision = 0.500, recall = 0.583, f1 = 0.538, AUC = 0.652
Train accuracy = 1.0
Test accuracy = 0.5945945945945946
Fold: 4. Accuracy = 0.595, precision = 0.421, recall = 0.667, f1 = 0.516, AUC = 0.613
Train accuracy = 1.0
Test accuracy = 0.7837837837837838
Fold: 5. Accuracy = 0.784, precision = 0.667, recall = 0.769, f1 = 0.714, AUC = 0.780
Train accuracy = 1.0
Test accuracy = 0.7297297297297297
Fold: 6. Accuracy = 0.730, precision = 0.600, recall = 0.692, f1 = 0.643, AUC = 0.721
Train accuracy = 1.0
Test accuracy = 0.7567567567567568
Fold: 7. Accuracy = 0.757, precision = 0.611, recall = 0.846, f1 = 0.710, AUC = 0.777
Train accuracy = 1.0
Test accuracy = 0.6756756756756757
Fold: 8. Accuracy = 0.676, precision = 0.526, recall = 0.769, f1 = 0.625, AUC = 0.697
Train accuracy = 1.0
Test accuracy = 0.5
Fold: 9. Accuracy = 0.500, precision = 0.250, recall = 0.250, f1 = 0.250, AUC = 0.438
Accuracy = 0.707, precision = 0.567, recall = 0.624, f1 = 0.585, AUC = 0.685
```

Training Results:
Max Depth: [1, 3, 5, 7, 9]
Accuracy: [0.71, 0.64, 0.65, 0.61, 0.54]

Testing Results:
Max Depth: [1, 3, 5, 7, 9]
Test Accuracy: [0.62, 0.39, 0.58, 0.62, 0.59]

Best Max Depth: 1 and 7 with a test accuracy of 0.62



From the above graph we can observe that the train accuracy is higher than the test accuracy. Hence the model is overfitting. Both train and test accuracy peak at max depth =1.

Decision Tree

Decision trees come under the concept of supervised learning. In this, the next step is decided based on the response of the previous step. For decision trees, the dataset has to be characterized. Decision trees mimic how humans actually think and make decisions.

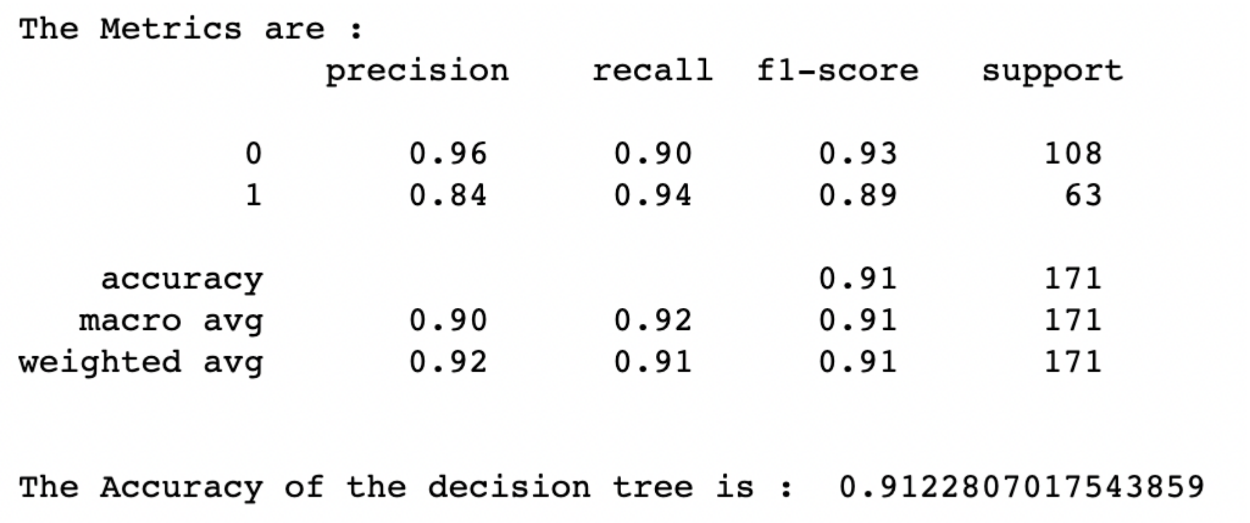
The algorithm can be explained in 5 steps :

Step-1: Start building the tree from the root node, let's say S, which holds the entire dataset.
Step-2: Utilize Attribute Selection Measure(ASM) to identify the dataset's top
Step-3: Divide the S into subsets that contain possible values for the best attributes.
Step-4: Generate the decision tree node, which contains the best attribute.
Step-5: Recursively make new decision trees using the subsets of the dataset created in step -3. Continue this process until a stage is reached where you cannot further classify the nodes and called the final node as a leaf node.

In the decision trees, we have used the gini index to evaluate the splits in the dataset.

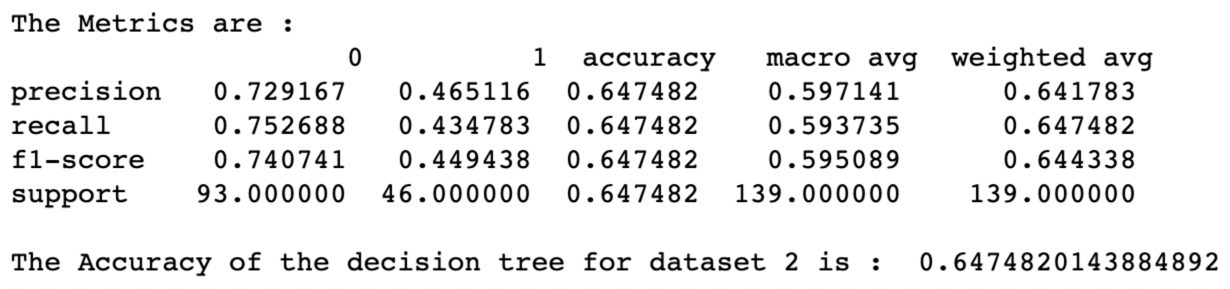
The accuracy obtained on the dataset 1 is : 91.2%
The accuracy obtained on the dataset 2 is : 64.7%

More details on the metrics are :



```
The Metrics are :
              precision    recall  f1-score   support

           0       0.96      0.90      0.93       108
           1       0.84      0.94      0.89        63

    accuracy                           0.91       171
   macro avg       0.90      0.92      0.91       171
weighted avg       0.92      0.91      0.91       171


The Accuracy of the decision tree is :  0.9122807017543859
```

For dataset 2 :



```
The Metrics are :
                   0          1  accuracy   macro avg  weighted avg
precision   0.729167   0.465116  0.647482    0.597141      0.641783
recall      0.752688   0.434783  0.647482    0.593735      0.647482
f1-score    0.740741   0.449438  0.647482    0.595089      0.644338
support    93.000000  46.000000  0.647482  139.000000    139.000000

The Accuracy of the decision tree for dataset 2 is :  0.6474820143884892
```

Hyperparameter Tuning & K-Fold Cross Validation:

In order not to overfit the data, we used hyperparameter tuning with parameters like to find the best parameter value for fitting the data. We have used parameters the below mentioned parameters for tuning :

- Criterion
- Max_depth
- Min_samples_split
- Min_samples_leaf

We have used GridSearchCV and provided the value 10 for parameter cv which will do 10-Fold cross validation internally.

GridSearchCV Explanation :

We use GridSearchCV to find the optimal parameters for the given model. The CV stands for cross validation evaluating the performance of each combination of the parameter mentioned in the grid. For the hyperparameter tuning, we split the dataset into test and train. On this training set, we test the hyperparameters. In GridsearchCV, for every parameter set, k-fold cross-validation takes place internally. When the function is called, it takes the parameter cv which specifies the number of folds. Based on this value, on every iteration, k-fold cross validation will take place.

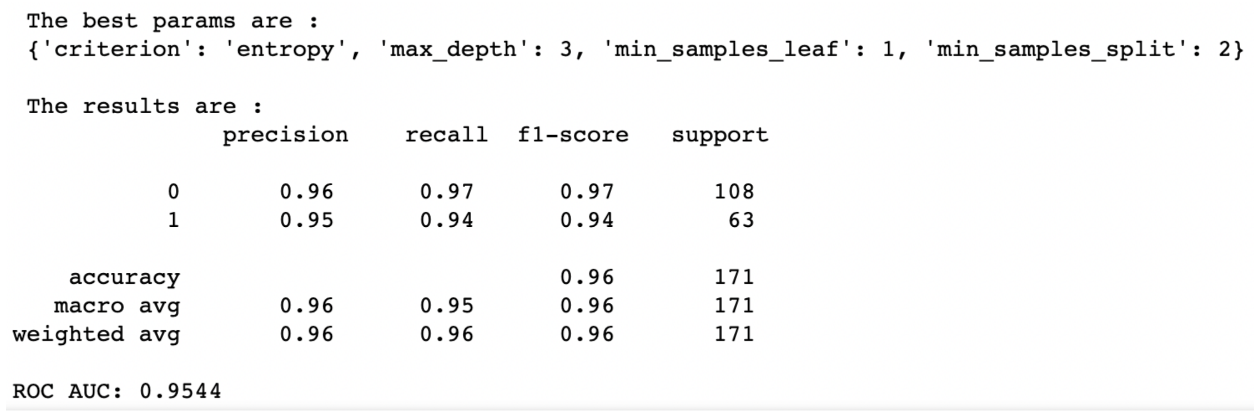
Some of the cross validation test results are :

```
[CV 1/10] END criterion=entropy, max_depth=5, min_samples_leaf=8, min_samples_split=3; AUC: (test=1.000) Accuracy: (test=0.975) F1: (test=0.966) Precision: (test=1.000) Recall: (test=0.933) total time=   0.0s
[CV 2/10] END criterion=entropy, max_depth=5, min_samples_leaf=8, min_samples_split=3; AUC: (test=0.965) Accuracy: (test=0.875) F1: (test=0.828) Precision: (test=0.857) Recall: (test=0.800) total time=   0.0s
[CV 3/10] END criterion=entropy, max_depth=5, min_samples_leaf=8, min_samples_split=3; AUC: (test=0.949) Accuracy: (test=0.875) F1: (test=0.839) Precision: (test=0.812) Recall: (test=0.867) total time=   0.0s
[CV 4/10] END criterion=entropy, max_depth=5, min_samples_leaf=8, min_samples_split=3; AUC: (test=0.961) Accuracy: (test=0.850) F1: (test=0.769) Precision: (test=0.909) Recall: (test=0.667) total time=   0.0s
[CV 5/10] END criterion=entropy, max_depth=5, min_samples_leaf=8, min_samples_split=3; AUC: (test=0.895) Accuracy: (test=0.900) F1: (test=0.857) Precision: (test=0.923) Recall: (test=0.800) total time=   0.0s
[CV 6/10] END criterion=entropy, max_depth=5, min_samples_leaf=8, min_samples_split=3; AUC: (test=0.943) Accuracy: (test=0.925) F1: (test=0.903) Precision: (test=0.875) Recall: (test=0.933) total time=   0.0s
[CV 7/10] END criterion=entropy, max_depth=5, min_samples_leaf=8, min_samples_split=3; AUC: (test=0.921) Accuracy: (test=0.925) F1: (test=0.903) Precision: (test=0.875) Recall: (test=0.933) total time=   0.0s
[CV 8/10] END criterion=entropy, max_depth=5, min_samples_leaf=8, min_samples_split=3; AUC: (test=1.000) Accuracy: (test=1.000) F1: (test=1.000) Precision: (test=1.000) Recall: (test=1.000) total time=   0.0s
[CV 9/10] END criterion=entropy, max_depth=5, min_samples_leaf=8, min_samples_split=3; AUC: (test=1.000) Accuracy: (test=1.000) F1: (test=1.000) Precision: (test=1.000) Recall: (test=1.000) total time=   0.0s
[CV 10/10] END criterion=entropy, max_depth=5, min_samples_leaf=8, min_samples_split=3; AUC: (test=0.929) Accuracy: (test=0.821) F1: (test=0.759) Precision: (test=0.786) Recall: (test=0.733) total time=   0.0s
```

From this, we were able to get the best parameters for dataset 1 as :

- 'criterion': entropy
- 'max_depth': 3
- 'min_samples_leaf': 1
- 'min_samples_split': 2
-

The maximum accuracy obtained from using these best parameters is : 96%



```
 The best params are :
 {'criterion': 'entropy', 'max_depth': 3, 'min_samples_leaf': 1, 'min_samples_split': 2}

 The results are :
               precision    recall  f1-score   support

           0       0.96      0.97      0.97       108
           1       0.95      0.94      0.94        63

    accuracy                           0.96       171
   macro avg       0.96      0.95      0.96       171
weighted avg       0.96      0.96      0.96       171

ROC AUC: 0.9544
```

The sample results from cross validation for hyperparameter for dataset 2 is :

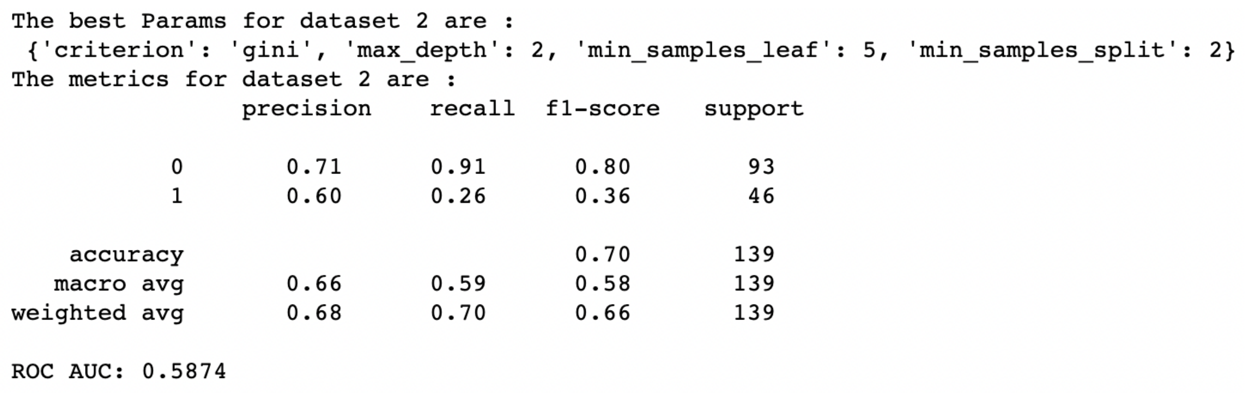
[CV 1/10] END criterion=entropy, max_depth=5, min_samples_leaf=8, min_samples_split=5; AUC: (test=0.734) Accuracy: (test=0.788) F1: (test=0.667) Precision: (test=0.778) Recall: (test=0.583) total time= 0.0s
[CV 2/10] END criterion=entropy, max_depth=5, min_samples_leaf=8, min_samples_split=5; AUC: (test=0.637) Accuracy: (test=0.636) F1: (test=0.500) Precision: (test=0.500) Recall: (test=0.500) total time= 0.0s
[CV 3/10] END criterion=entropy, max_depth=5, min_samples_leaf=8, min_samples_split=5; AUC: (test=0.802) Accuracy: (test=0.727) F1: (test=0.667) Precision: (test=0.600) Recall: (test=0.750) total time= 0.0s
[CV 4/10] END criterion=entropy, max_depth=5, min_samples_leaf=8, min_samples_split=5; AUC: (test=0.548) Accuracy: (test=0.594) F1: (test=0.381) Precision: (test=0.444) Recall: (test=0.333) total time= 0.0s
[CV 5/10] END criterion=entropy, max_depth=5, min_samples_leaf=8, min_samples_split=5; AUC: (test=0.803) Accuracy: (test=0.719) F1: (test=0.308) Precision: (test=1.000) Recall: (test=0.182) total time= 0.0s
[CV 6/10] END criterion=entropy, max_depth=5, min_samples_leaf=8, min_samples_split=5; AUC: (test=0.528) Accuracy: (test=0.562) F1: (test=0.417) Precision: (test=0.385) Recall: (test=0.455) total time= 0.0s

[CV 7/10] END criterion=entropy, max_depth=5, min_samples_leaf=8, min_samples_split=5; AUC: (test=0.610) Accuracy: (test=0.656) F1: (test=0.353) Precision: (test=0.500) Recall: (test=0.273) total time= 0.0s
[CV 8/10] END criterion=entropy, max_depth=5, min_samples_leaf=8, min_samples_split=5; AUC: (test=0.682) Accuracy: (test=0.750) F1: (test=0.600) Precision: (test=0.667) Recall: (test=0.545) total time= 0.0s
[CV 9/10] END criterion=entropy, max_depth=5, min_samples_leaf=8, min_samples_split=5; AUC: (test=0.799) Accuracy: (test=0.750) F1: (test=0.667) Precision: (test=0.615) Recall: (test=0.727) total time= 0.0s
[CV 10/10] END criterion=entropy, max_depth=5, min_samples_leaf=8, min_samples_split=5; AUC: (test=0.673) Accuracy: (test=0.625) F1: (test=0.333) Precision: (test=0.429) Recall: (test=0.273) total time= 0.0s

For the dataset 2 , the best parameters were obtained as :

- 'criterion': 'gini'
- 'max_depth': 2
- 'min_samples_leaf': 5
- 'min_samples_split': 2

The accuracy obtained from these best parameters is 70.0%



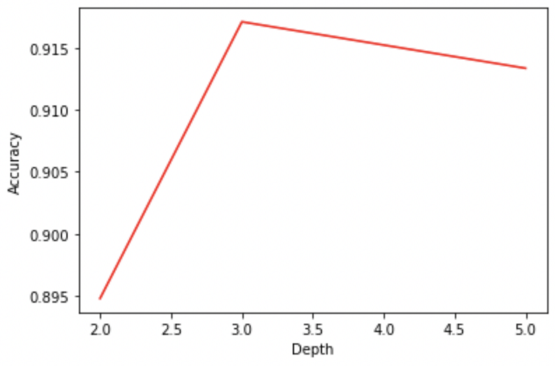
```
The best Params for dataset 2 are :
 {'criterion': 'gini', 'max_depth': 2, 'min_samples_leaf': 5, 'min_samples_split': 2}
The metrics for dataset 2 are :
               precision    recall  f1-score   support

           0       0.71      0.91      0.80        93
           1       0.60      0.26      0.36        46

    accuracy                           0.70       139
   macro avg       0.66      0.59      0.58       139
weighted avg       0.68      0.70      0.66       139

ROC AUC: 0.5874
```

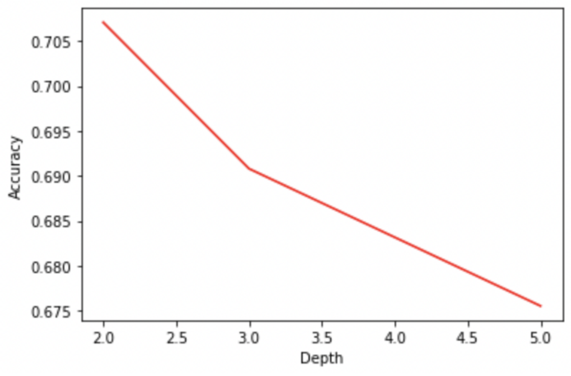
Discussion on bias/Variance tradeoff :

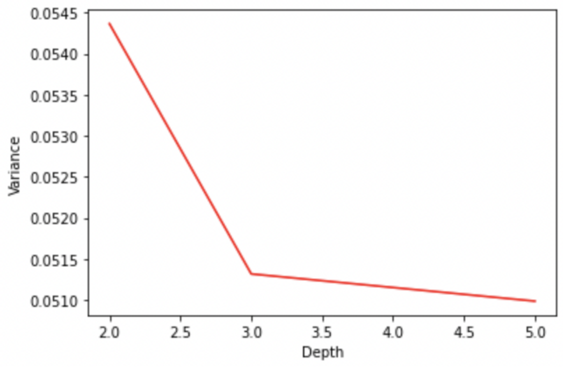
(Column 1 - Dataset 1 Column 2 - Dataset 2 graphs)

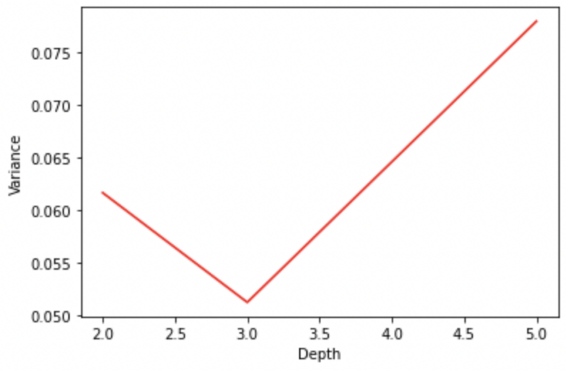
For decision trees, the model tends to move towards overfitting if we go on increasing the depth of the trees. From the hyperparameter tuning, plotting a graph against the depth of the trees and Test and train accuracy, we see that we get the best accuracy for the mid value(3 in case of dataset 1 and 2 in case of dataset 2). If we further,

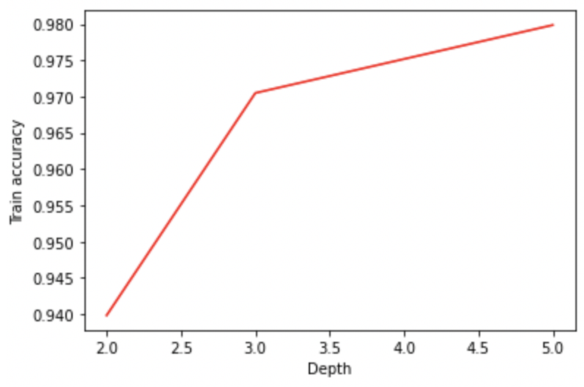
increase the depth, the test accuracy reduces. The first graph shows the test accuracy on y axis and depth on x axis(For dataset 1- The trends were similar for dataset 2). The second graph shows variance against the depth. We see that as the depth increase, the variance goes on increasing moving the model towards overfitting.
We find the optimal point at point 3 where we find the bias to be low(higher accuracy) and low variance.
Coming to the bias, we see that in both cases the train accuracy increases but the test accuracy decreases at the point 3 and that is where the variance is low. Hence that is the optimal point.

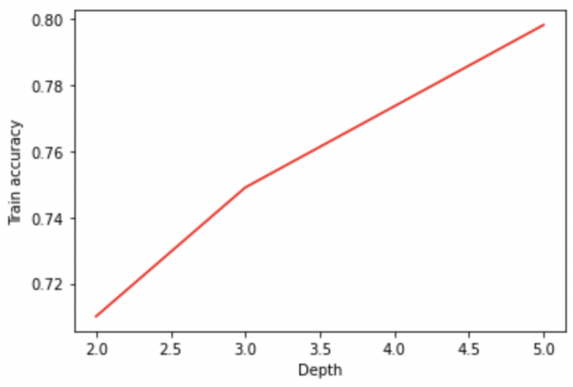












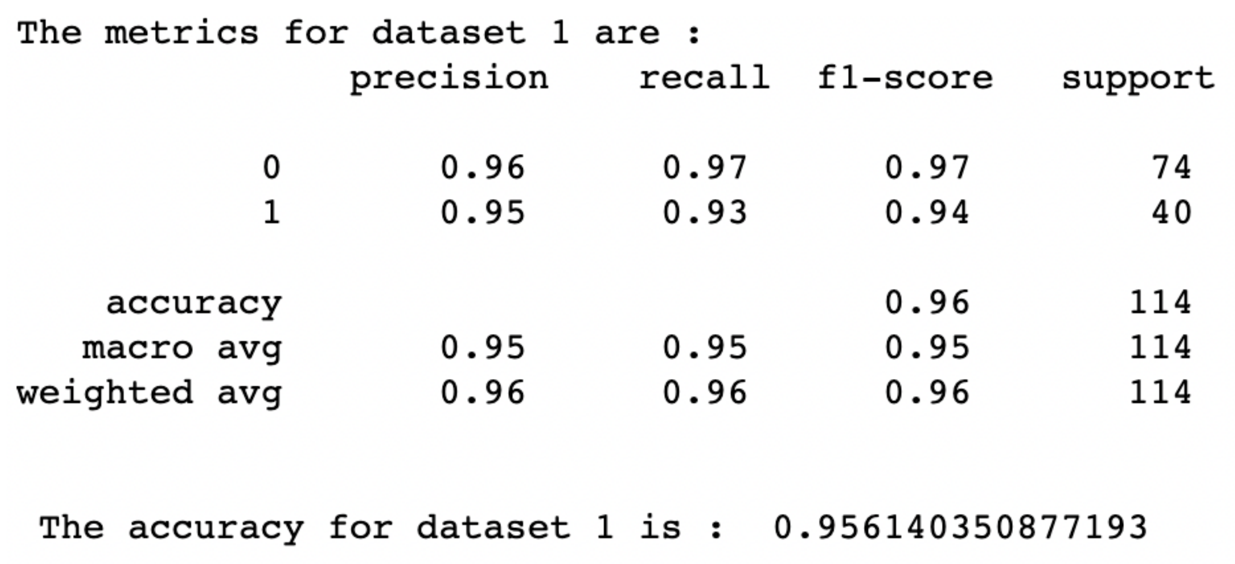
Support Vector Machine

Support vector machines are a class of supervised learning which are employed to find the hyperplane that would best divide the dataset into 2 classes. The points that are closest to this hyperplane are called as support vectors.
The SVM algorithm chooses a hyperplane as the best when the distance between the hyperplane and the support vectors are to be maximum.

To implement SVM , we have used svm.SVC from the sklearn package. We have split the dataset into train and test sets.

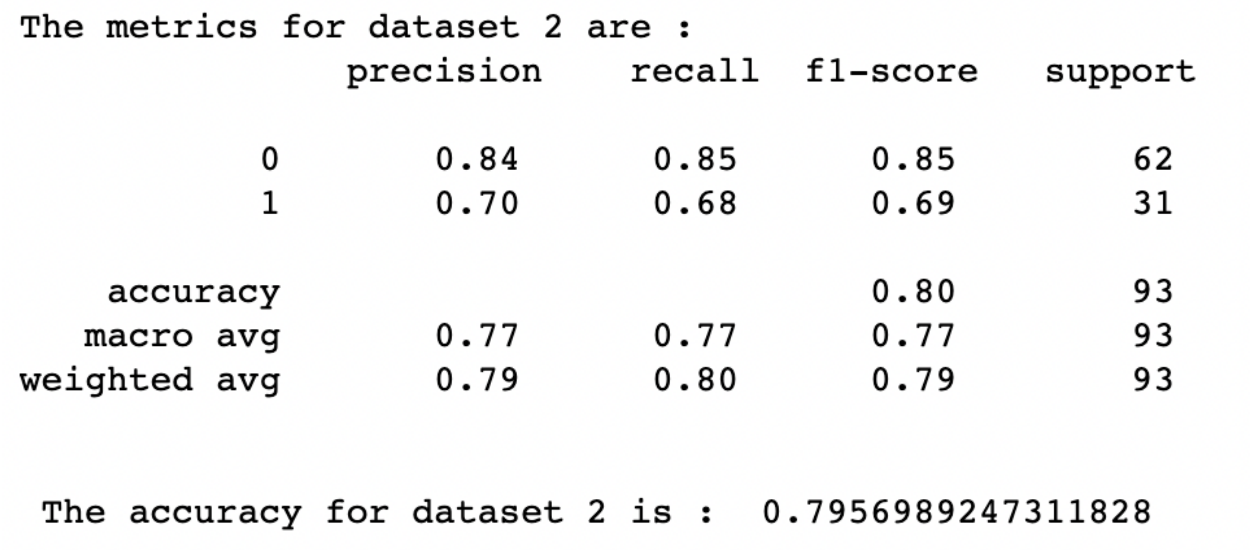
The accuracy on the test set for dataset 1 was found to be : 95.6%
The accuracy on the test set for dataset 2 was found to be : 79.56%.

The other metrics are :



```
The metrics for dataset 1 are :
              precision    recall  f1-score   support

           0       0.96      0.97      0.97        74
           1       0.95      0.93      0.94        40

    accuracy                           0.96       114
   macro avg       0.95      0.95      0.95       114
weighted avg       0.96      0.96      0.96       114


 The accuracy for dataset 1 is :  0.956140350877193
```

For dataset 2 :



```
The metrics for dataset 2 are :
              precision    recall  f1-score   support

           0       0.84      0.85      0.85        62
           1       0.70      0.68      0.69        31

    accuracy                           0.80        93
   macro avg       0.77      0.77      0.77        93
weighted avg       0.79      0.80      0.79        93


 The accuracy for dataset 2 is :  0.7956989247311828
```

Hyperparameter Tuning & K-Fold Cross Validation:

The explanation to GridsearchCV is provided in the decision tree section. To find the best parameters, we employed hyperparameter tuning with parameters :
'C': [0.1, 1, 10, 100, 1000]
'gamma': [1, 0.1, 0.01, 0.001, 0.0001]
'kernel': ['linear']

Sample results from hyperparameter tuning and 10-fold cross validation :

[CV 10/10] END C=10, gamma=0.1, kernel=linear; AUC: (test=1.000) Accuracy: (test=0.956) F1: (test=0.938) Precision: (test=1.000) Recall: (test=0.882) total time= 0.8s
[CV 1/10] END C=10, gamma=0.01, kernel=linear; AUC: (test=1.000) Accuracy: (test=0.978) F1: (test=0.973) Precision: (test=0.947) Recall: (test=1.000) total time= 1.3s
[CV 2/10] END C=10, gamma=0.01, kernel=linear; AUC: (test=0.996) Accuracy: (test=0.935) F1: (test=0.914) Precision: (test=0.941) Recall: (test=0.889) total time= 1.4s
[CV 3/10] END C=10, gamma=0.01, kernel=linear; AUC: (test=1.000) Accuracy: (test=1.000) F1: (test=1.000) Precision: (test=1.000) Recall: (test=1.000) total time= 1.1s
[CV 4/10] END C=10, gamma=0.01, kernel=linear; AUC: (test=0.990) Accuracy: (test=0.957) F1: (test=0.938) Precision: (test=1.000) Recall: (test=0.882) total time= 1.5s
[CV 5/10] END C=10, gamma=0.01, kernel=linear; AUC: (test=1.000) Accuracy: (test=0.978) F1: (test=0.971) Precision: (test=0.944) Recall: (test=1.000) total time= 1.2s
[CV 6/10] END C=10, gamma=0.01, kernel=linear; AUC: (test=0.996) Accuracy: (test=0.978) F1: (test=0.970) Precision: (test=1.000) Recall: (test=0.941) total time= 1.6s
[CV 7/10] END C=10, gamma=0.01, kernel=linear; AUC: (test=0.977) Accuracy: (test=0.933) F1: (test=0.903) Precision: (test=1.000) Recall: (test=0.824) total time= 1.6s
[CV 8/10] END C=10, gamma=0.01, kernel=linear; AUC: (test=0.996) Accuracy: (test=0.978) F1: (test=0.971) Precision: (test=0.944) Recall: (test=1.000) total time= 2.5s

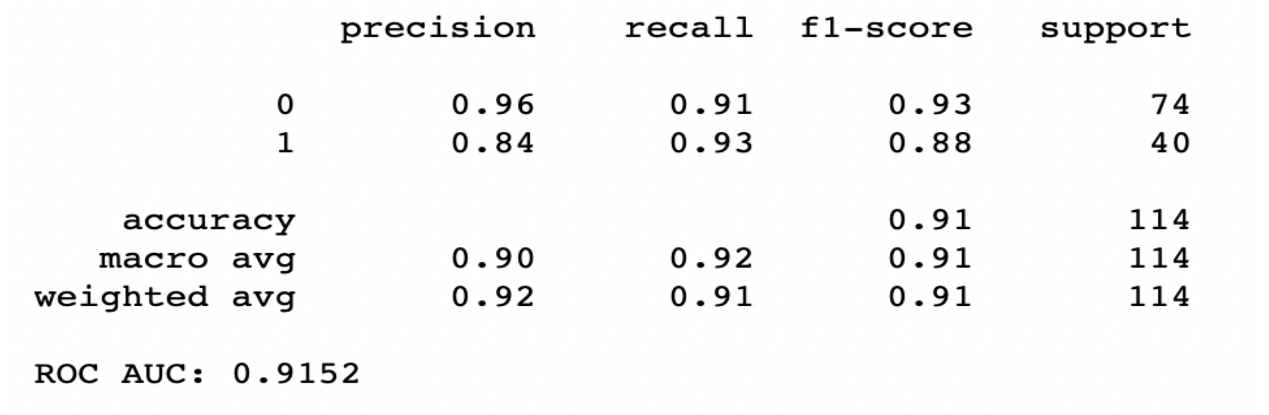
[CV 9/10] END C=10, gamma=0.01, kernel=linear; AUC: (test=0.977) Accuracy: (test=0.933) F1: (test=0.909) Precision: (test=0.938) Recall: (test=0.882) total time= 3.6s
[CV 10/10] END C=10, gamma=0.01, kernel=linear; AUC: (test=1.000) Accuracy: (test=0.956) F1: (test=0.938) Precision: (test=1.000) Recall: (test=0.882) total time= 0.8s

The best params for dataset1 were found to be :

- 'C': 1
- 'gamma': 1

The accuracy when the dataset was fit with these best parameters is : 97.0%

The metrics with the best params was :



```
              precision    recall  f1-score   support

           0       0.96      0.91      0.93        74
           1       0.84      0.93      0.88        40

    accuracy                           0.91       114
   macro avg       0.90      0.92      0.91       114
weighted avg       0.92      0.91      0.91       114

ROC AUC: 0.9152
```

Sample results from hyperparameter tuning and 10-fold cross validation :

[CV 1/10] END C=1000, gamma=1, kernel=linear; AUC: (test=0.747) Accuracy: (test=0.784) F1: (test=0.636) Precision: (test=0.778) Recall: (test=0.538) total time= 6.8s
[CV 2/10] END C=1000, gamma=1, kernel=linear; AUC: (test=0.638) Accuracy: (test=0.676) F1: (test=0.455) Precision: (test=0.556) Recall: (test=0.385) total time= 8.5s
[CV 3/10] END C=1000, gamma=1, kernel=linear; AUC: (test=0.651) Accuracy: (test=0.676) F1: (test=0.333) Precision: (test=0.600) Recall: (test=0.231) total time= 8.3s
[CV 4/10] END C=1000, gamma=1, kernel=linear; AUC: (test=0.817) Accuracy: (test=0.811) F1: (test=0.667) Precision: (test=0.875) Recall: (test=0.538) total time= 12.6s
[CV 5/10] END C=1000, gamma=1, kernel=linear; AUC: (test=0.599) Accuracy: (test=0.595) F1: (test=0.348) Precision: (test=0.400) Recall: (test=0.308) total time= 8.2s
[CV 6/10] END C=1000, gamma=1, kernel=linear; AUC: (test=0.718) Accuracy: (test=0.622) F1: (test=0.417) Precision: (test=0.455) Recall: (test=0.385) total time= 16.8s
[CV 7/10] END C=1000, gamma=1, kernel=linear; AUC: (test=0.728) Accuracy: (test=0.703) F1: (test=0.522) Precision: (test=0.600) Recall: (test=0.462) total time= 13.9s
[CV 8/10] END C=1000, gamma=1, kernel=linear; AUC: (test=0.837) Accuracy: (test=0.757) F1: (test=0.526) Precision: (test=0.833) Recall: (test=0.385) total time= 11.3s
[CV 9/10] END C=1000, gamma=1, kernel=linear; AUC: (test=0.872) Accuracy: (test=0.838) F1: (test=0.750) Precision: (test=0.818) Recall: (test=0.692) total time= 14.3s

[CV 10/10] END C=1000, gamma=1, kernel=linear; AUC: (test=0.799) Accuracy: (test=0.778) F1: (test=0.636) Precision: (test=0.700) Recall: (test=0.583) total time= 10.2s

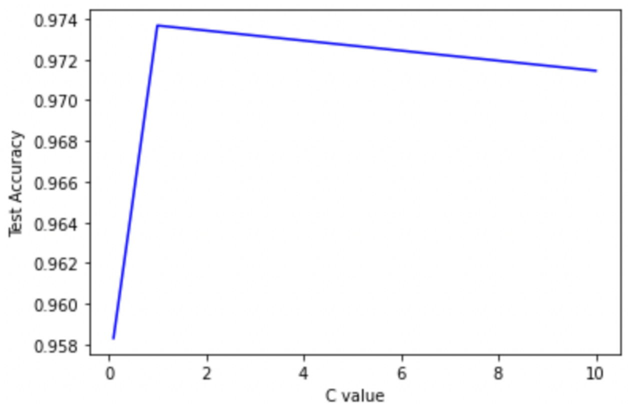
The best params for dataset2 were found to be :

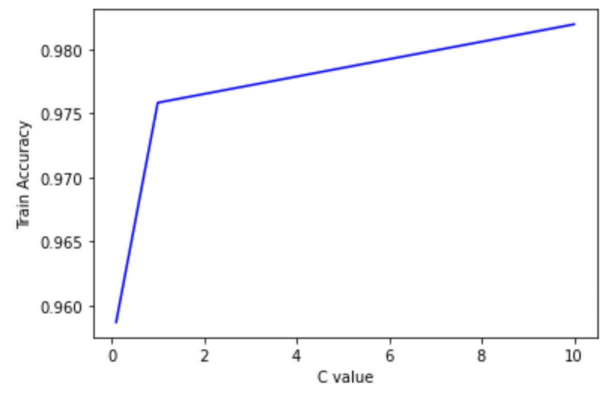
- 'C': 10
- 'gamma': 1

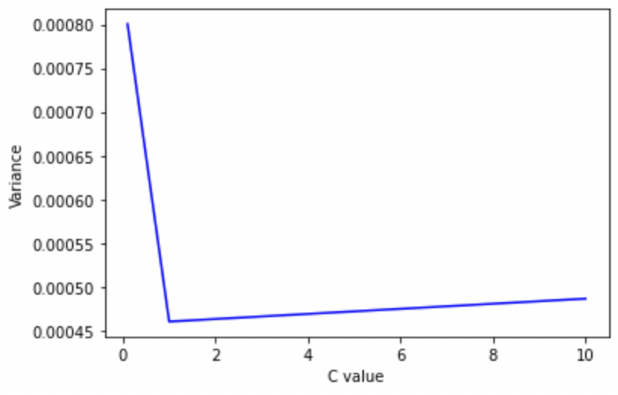
The accuracy obtained from these best parameters are : 73.0%.
The other metrics with best params are :

```
              precision    recall  f1-score   support

           0       0.79      0.90      0.84        62
           1       0.73      0.52      0.60        31

    accuracy                           0.77        93
   macro avg       0.76      0.71      0.72        93
weighted avg       0.77      0.77      0.76        93

ROC AUC: 0.7097
```

Discussion on Bias-Variance Tradeoff :







Taking the example of dataset 1 to discuss the bias-variance tradeoff(since the trends for dataset 2 are similar). Based on the plotted graphs for C value we can see that, **at point 0.1, the test accuracy and the test accuracy is extremely low with high variance making the model underfit.** As we increase the value of C, the model reaches an optimal point at C=1. where test accuracy is high and variance is low. **As we increase the value further, the test accuracy drops, moving the model to underfitting.**

Random Forest

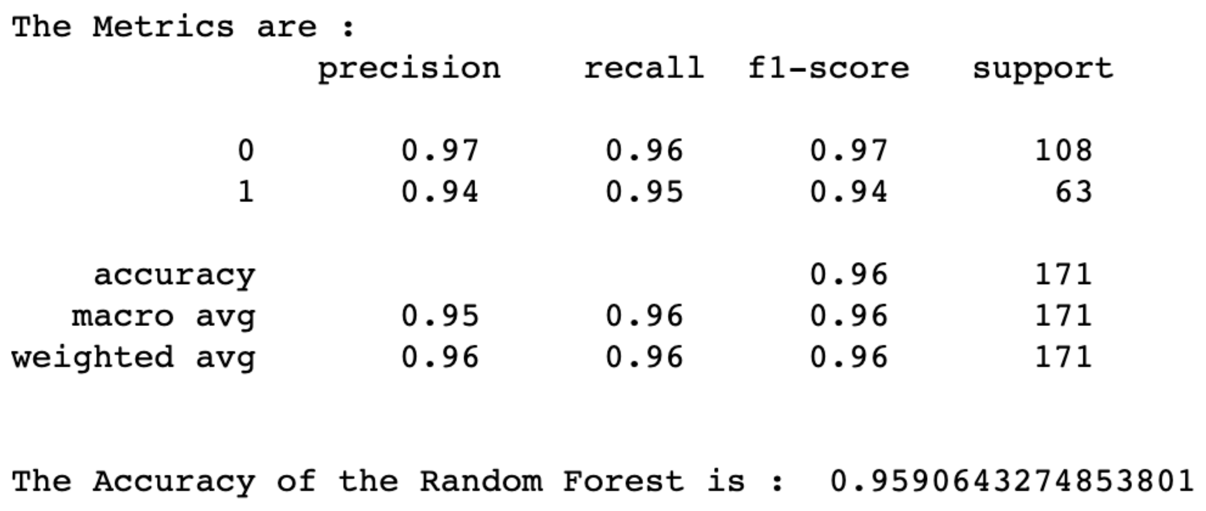
Random Forest is the group of decision trees with very low correlation between them. It is also a class of supervised machine learning algorithm. How the random forest works on a high level is that, each decision tree in the random forest is given a classification. Random forest works on the Bagging principle.

The steps in the Random forest are as follows :
Step 1 : In Random forest n number of random records are taken from the data set having k number of records.
Step 2 : Individual decision trees are constructed for each sample.
Step 3 : Each decision tree will generate an output.
Step 4 : Final output is considered based on Majority Voting or Averaging for Classification and regression respectively

While implementing Random Forest , we have used to RandomForestClassifier from the sklearn.ensemble package.

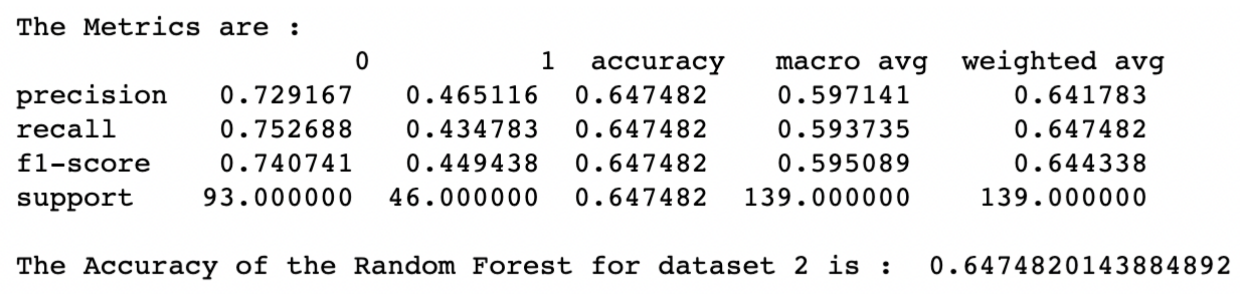
The accuracy we obtained for the dataset 1 is : 95.9%
The accuracy obtained for the dataset 2 is : 64.74%

The other metrics were :



```
The Metrics are :
              precision    recall  f1-score   support

           0       0.97      0.96      0.97       108
           1       0.94      0.95      0.94        63

    accuracy                           0.96       171
   macro avg       0.95      0.96      0.96       171
weighted avg       0.96      0.96      0.96       171


The Accuracy of the Random Forest is :  0.9590643274853801
```

Dataset 2 :



```
The Metrics are :
                   0          1  accuracy   macro avg  weighted avg
precision   0.729167   0.465116  0.647482    0.597141      0.641783
recall      0.752688   0.434783  0.647482    0.593735      0.647482
f1-score    0.740741   0.449438  0.647482    0.595089      0.644338
support    93.000000  46.000000  0.647482  139.000000    139.000000

The Accuracy of the Random Forest for dataset 2 is :  0.6474820143884892
```

Hyperparameter Tuning & K-Fold Cross Validation:

The explanation to GridsearchCV is provided in the decision tree section. In order to not overfit the data, we have employed hyperparameter tuning with 10 fold cross validation to find the best params for both the datasets.

The Best parameters for dataset 1 are :

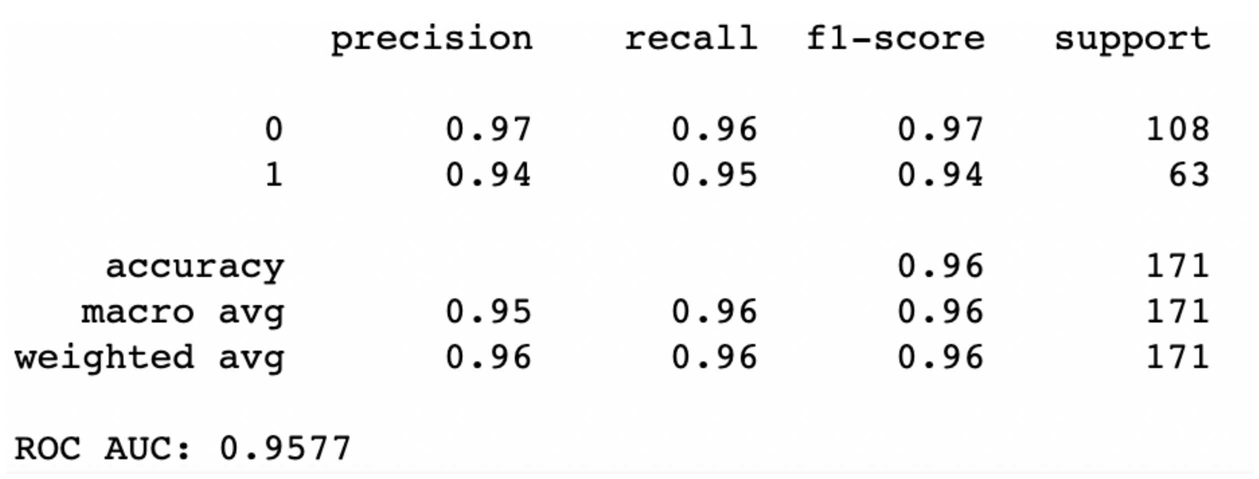
- 'max_depth': 10
- 'max_features': 'auto'
- 'min_samples_leaf': 1
- 'min_samples_split': 2
- 'n_estimators': 200

Sample 10-fold cross validation output for one of the hyperparameter combination :

[CV 1/10] END max_depth=30, max_features=sqrt, min_samples_leaf=4, min_samples_split=10, n_estimators=600; AUC: (test=1.000) Accuracy: (test=1.000) F1: (test=1.000) Precision: (test=1.000) Recall: (test=1.000) total time=   0.5s
[CV 2/10] END max_depth=30, max_features=sqrt, min_samples_leaf=4, min_samples_split=10, n_estimators=600; AUC: (test=0.997) Accuracy: (test=0.975) F1: (test=0.968) Precision: (test=0.938) Recall: (test=1.000) total time=   0.5s
[CV 3/10] END max_depth=30, max_features=sqrt, min_samples_leaf=4, min_samples_split=10, n_estimators=600; AUC: (test=0.981) Accuracy: (test=0.900) F1: (test=0.867) Precision: (test=0.867) Recall: (test=0.867) total time=   0.5s
[CV 4/10] END max_depth=30, max_features=sqrt, min_samples_leaf=4, min_samples_split=10, n_estimators=600; AUC: (test=0.992) Accuracy: (test=0.925) F1: (test=0.897) Precision: (test=0.929) Recall: (test=0.867) total time=   0.5s
[CV 5/10] END max_depth=30, max_features=sqrt, min_samples_leaf=4, min_samples_split=10, n_estimators=600; AUC: (test=0.936) Accuracy: (test=0.900) F1: (test=0.846) Precision: (test=1.000) Recall: (test=0.733) total time=   0.5s

```
[CV 6/10] END max_depth=30, max_features=sqrt, min_samples_leaf=4, min_samples_split=10,
n_estimators=600; AUC: (test=0.989) Accuracy: (test=0.975) F1: (test=0.966) Precision: (test=1.000)
Recall: (test=0.933) total time=   0.5s
[CV 7/10] END max_depth=30, max_features=sqrt, min_samples_leaf=4, min_samples_split=10,
n_estimators=600; AUC: (test=0.987) Accuracy: (test=0.900) F1: (test=0.875) Precision: (test=0.824)
Recall: (test=0.933) total time=   0.5s
[CV 8/10] END max_depth=30, max_features=sqrt, min_samples_leaf=4, min_samples_split=10,
n_estimators=600; AUC: (test=1.000) Accuracy: (test=0.975) F1: (test=0.966) Precision: (test=1.000)
Recall: (test=0.933) total time=   0.5s
[CV 9/10] END max_depth=30, max_features=sqrt, min_samples_leaf=4, min_samples_split=10,
n_estimators=600; AUC: (test=1.000) Accuracy: (test=1.000) F1: (test=1.000) Precision: (test=1.000)
Recall: (test=1.000) total time=   0.5s
[CV 10/10] END max_depth=30, max_features=sqrt, min_samples_leaf=4, min_samples_split=10,
n_estimators=600; AUC: (test=0.994) Accuracy: (test=0.949) F1: (test=0.933) Precision: (test=0.933)
Recall: (test=0.933) total time=   0.5s
```

The accuracy obtained from the best params is 96% and the AUC is 0.9577



```
              precision    recall  f1-score   support

           0       0.97      0.96      0.97       108
           1       0.94      0.95      0.94        63

    accuracy                           0.96       171
   macro avg       0.95      0.96      0.96       171
weighted avg       0.96      0.96      0.96       171

ROC AUC: 0.9577
```

Dataset 2 :

The best params for dataset 2 were found to be :

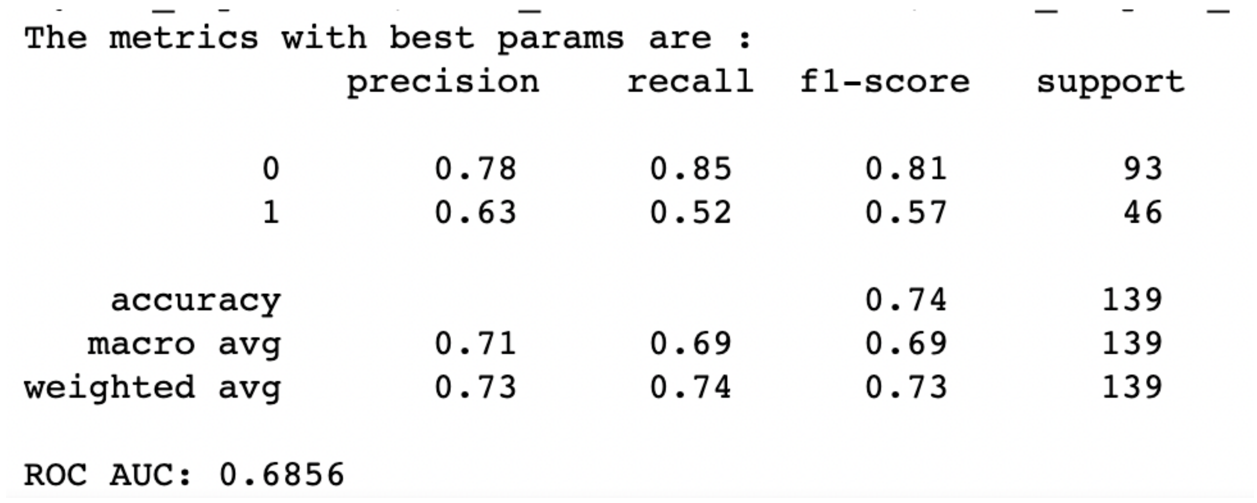
- 'max_depth': 20
- 'max_features': 'auto'
- 'min_samples_leaf': 1
- 'min_samples_split': 5
- 'n_estimators': 300

Sample 10-fold cross validation output for one of the hyperparameter combination :

```
Fitting 10 folds for each of 162 candidates, totalling 1620 fits
```

```
[CV 1/10] END max_depth=10, max_features=auto, min_samples_leaf=1, min_samples_split=2,
n_estimators=200; AUC: (test=0.627) Accuracy: (test=0.697) F1: (test=0.583) Precision: (test=0.583)
Recall: (test=0.583) total time=   0.2s
[CV 2/10] END max_depth=10, max_features=auto, min_samples_leaf=1, min_samples_split=2,
n_estimators=200; AUC: (test=0.667) Accuracy: (test=0.606) F1: (test=0.235) Precision: (test=0.400)
Recall: (test=0.167) total time=   0.1s
[CV 3/10] END max_depth=10, max_features=auto, min_samples_leaf=1, min_samples_split=2,
n_estimators=200; AUC: (test=0.917) Accuracy: (test=0.818) F1: (test=0.727) Precision: (test=0.800)
Recall: (test=0.667) total time=   0.1s
[CV 4/10] END max_depth=10, max_features=auto, min_samples_leaf=1, min_samples_split=2,
n_estimators=200; AUC: (test=0.617) Accuracy: (test=0.594) F1: (test=0.381) Precision: (test=0.444)
Recall: (test=0.333) total time=   0.1s
[CV 5/10] END max_depth=10, max_features=auto, min_samples_leaf=1, min_samples_split=2,
n_estimators=200; AUC: (test=0.861) Accuracy: (test=0.688) F1: (test=0.286) Precision: (test=0.667)
Recall: (test=0.182) total time=   0.1s
[CV 6/10] END max_depth=10, max_features=auto, min_samples_leaf=1, min_samples_split=2,
n_estimators=200; AUC: (test=0.697) Accuracy: (test=0.688) F1: (test=0.375) Precision: (test=0.600)
Recall: (test=0.273) total time=   0.1s
[CV 7/10] END max_depth=10, max_features=auto, min_samples_leaf=1, min_samples_split=2,
n_estimators=200; AUC: (test=0.589) Accuracy: (test=0.688) F1: (test=0.444) Precision: (test=0.571)
Recall: (test=0.364) total time=   0.1s
[CV 8/10] END max_depth=10, max_features=auto, min_samples_leaf=1, min_samples_split=2,
n_estimators=200; AUC: (test=0.779) Accuracy: (test=0.719) F1: (test=0.571) Precision: (test=0.600)
Recall: (test=0.545) total time=   0.1s
[CV 9/10] END max_depth=10, max_features=auto, min_samples_leaf=1, min_samples_split=2,
n_estimators=200; AUC: (test=0.779) Accuracy: (test=0.781) F1: (test=0.667) Precision: (test=0.700)
Recall: (test=0.636) total time=   0.1s
[CV 10/10] END max_depth=10, max_features=auto, min_samples_leaf=1, min_samples_split=2,
n_estimators=200; AUC: (test=0.610) Accuracy: (test=0.719) F1: (test=0.471) Precision: (test=0.667)
Recall: (test=0.364) total time=   0.1s
```

The other metrics are :

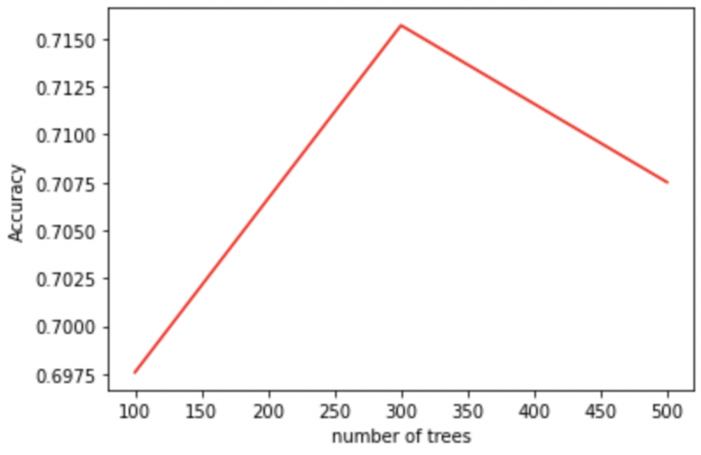


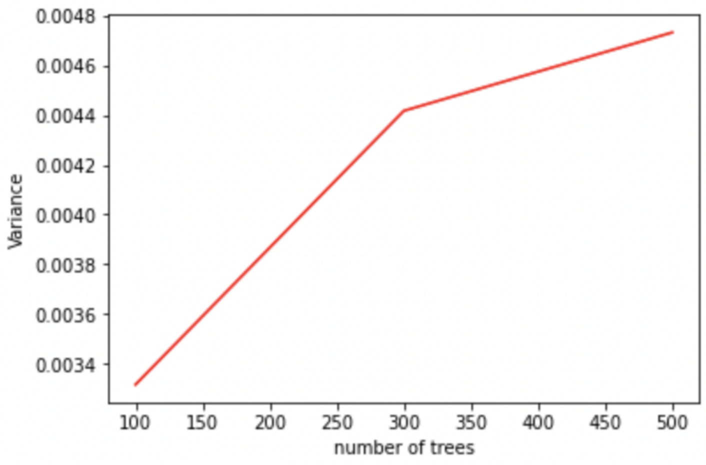
```
The metrics with best params are :
              precision    recall  f1-score   support

           0       0.78      0.85      0.81        93
           1       0.63      0.52      0.57        46

    accuracy                           0.74       139
   macro avg       0.71      0.69      0.69       139
weighted avg       0.73      0.74      0.73       139

ROC AUC: 0.6856
```

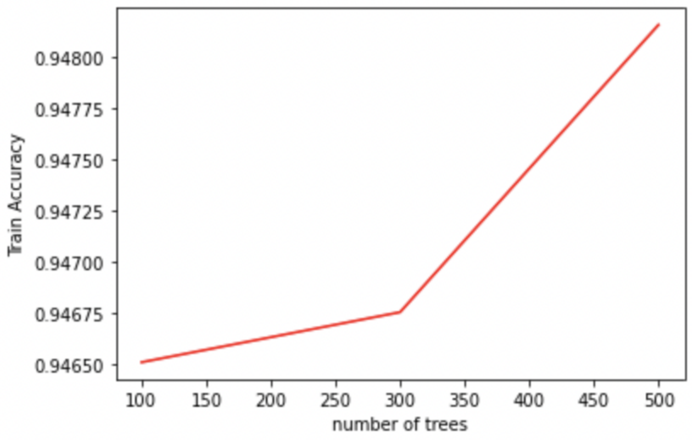
The accuracy was found to be 74% on using the best parameters with AUC to be 0.6856

Bias-Variance Tradeoff :

For Random forest, we use bagging in order to remove the overfitting. The bias and variance in Random forest increase or decrease based on the number of trees. So, we plotted a graph as shown below, Fig 1- Number of trees vs Test Accuracy. Fig 2-Number of trees vs variance. From the graphs, we can see that the optimal value for number of trees is 300 for dataset 1 since it has both low bias(higher test accuracy) and low variance. Whereas **for 100, we can see that bias is low and the variance appears to be low. But the test accuracy is extremely low. Hence the model is underfitting.** Hence as we increase number of trees, we can see the trend of increasing variance means the model moves towards overfitting. Fig 3 shows the train accuracy and **if we compare train vs test, we see that train accuracy keeps increasing but the accuracy post 300 drops for test moving the model towards overfitting**




0.0048
0.0046
0.0044
0.0042
0.0040
0.0038
0.0036
0.0034
Variance
100
150
200
250
300
350
400
450
500
number of trees


0.94800
0.94775
0.94750
0.94725
0.94700
0.94675
0.94650
Train Accuracy
100
150
200
250
300
350
400
450
500
number of trees

- **Neural Network**

Neural networks are a type of machine learning algorithm that is inspired by the structure and function of the human brain. They are composed of many interconnected units called neurons, which process and transmit information.

Neural networks are often used for tasks such as image and speech recognition, natural language processing, and predictive modeling. They are able to learn and adapt to new data, making them powerful tools for solving complex problems.

One of the key algorithms used in neural networks is called backpropagation. This algorithm is used to train the network by adjusting the connections between neurons in order to minimize the error between the predicted outputs of the network and the true outputs.

To do this, the algorithm uses a process called gradient descent, which involves calculating the gradient of the error with respect to the weights of the connections between neurons. This gradient is then used to update the weights in a way that reduces the error. This process is repeated for many iterations, allowing the network to gradually learn and improve its performance.

Loss used: Cross Entropy Loss

Bias variance tradeoff and discussion

Hidden units:

Hidden nodes tested: `[100, 10], [1000, 100], [20_000, 1000]`
Both the train and test accuracies first increase with the increase in the number of hidden units. The test accuracy is best at [1000, 100] hidden units. Then the accuracy (train & test) begins to decrease.

This is because for a lower number of hidden units the model suffers from underfitting as the number of hidden units is not enough to capture the trends in the data. This leads to a situation where there is high bias and low variance.

When there is a very large number of hidden units, the model test accuracy decrease as the model suffers from overfitting. The training accuracy also decreases because there is insufficient training data to fit the large model. This is a case of low bias - high variance.

Learning Rate:

Learning rate tested: `0.01, 0.001, 0.0001`
The accuracy first increases with the learning rate as initially, the learning rate is too high, and instead of converging to the minima the weights get updated too severely and the weights oscillate around the optimal value. For the intermediate learning rate (0.001), the accuracy is the

best. For learning rate that is too low (0.0001), the accuracy is not good as the model is not able to converge to the minima in the given number of epochs. This is a case of underfitting.

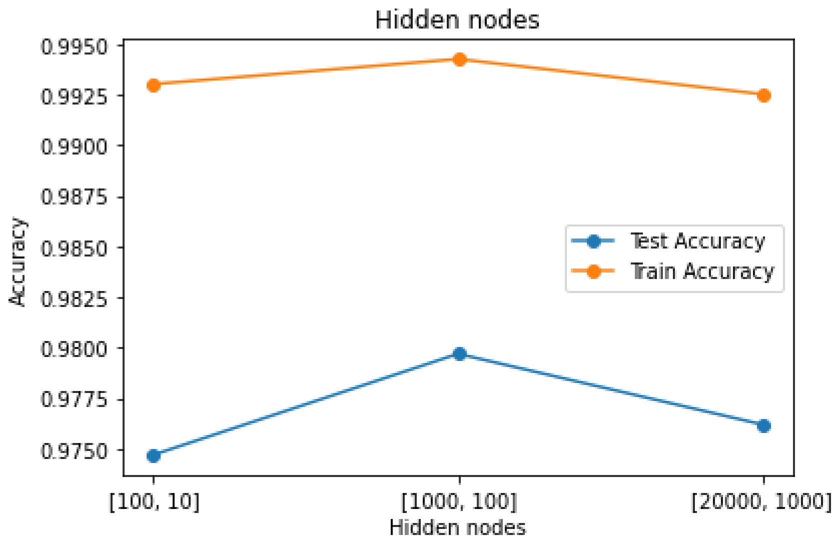
Weight initialization:

Weight initialization tested: `HeNormal(), RandomNormal(mean=0.0, stddev=0.05, seed=42), Constant(0.1)`

The best weight initialization is RandomNormal. Getting weights from a normal distribution is more effective than having constant weights as it helps explore the search space more efficiently.

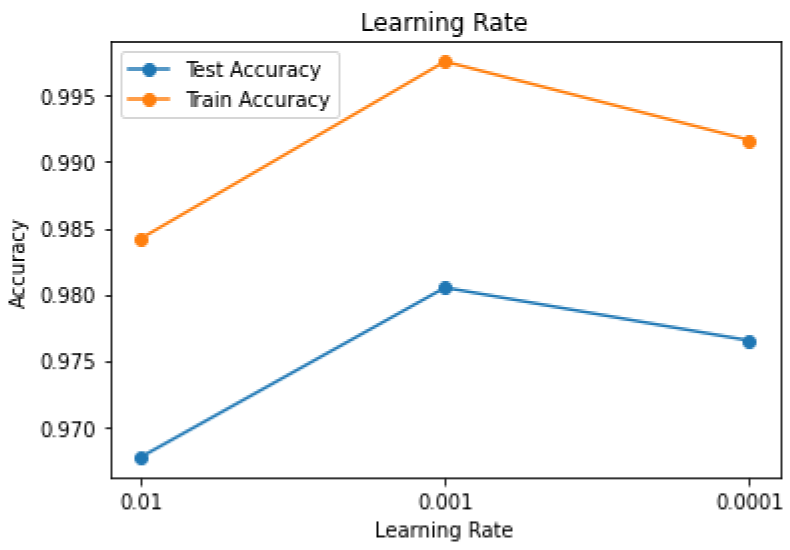
Results

```
--------------------------------------------------
Hidden nodes  =>  [100, 10]
--------------------------------------------------
Test Loss           = 0.0870
Test Accuracy         = 0.9747
Test Precision        = 0.9745
Test Recall          = 0.9745
Test F1 score         = 0.9744
Test AUC             = 0.9994
Train Loss           = 0.0265
Train Accuracy         = 0.9930
Train Precision        = 0.9930
Train Recall          = 0.9930
Train F1 score         = 0.9930
Train AUC             = 0.9999

--------------------------------------------------
Hidden nodes  =>  [1000, 100]
--------------------------------------------------
Test Loss           = 0.0837
Test Accuracy         = 0.9797
Test Precision        = 0.9800
Test Recall          = 0.9794
Test F1 score         = 0.9796
Test AUC             = 0.9997
Train Loss           = 0.0190
Train Accuracy         = 0.9943
Train Precision        = 0.9944
Train Recall          = 0.9942
Train F1 score        = 0.9942
Train AUC            = 1.0000

--------------------------------------------------
Hidden nodes  =>  [20000, 1000]
--------------------------------------------------
Test Loss           = 0.1120
Test Accuracy        = 0.9762
Test Precision       = 0.9765
Test Recall         = 0.9758
Test F1 score        = 0.9760
Test AUC            = 0.9995
Train Loss          = 0.0227
Train Accuracy       = 0.9925
Train Precision      = 0.9926
Train Recall         = 0.9924
Train F1 score       = 0.9925
Train AUC           = 1.0000
```

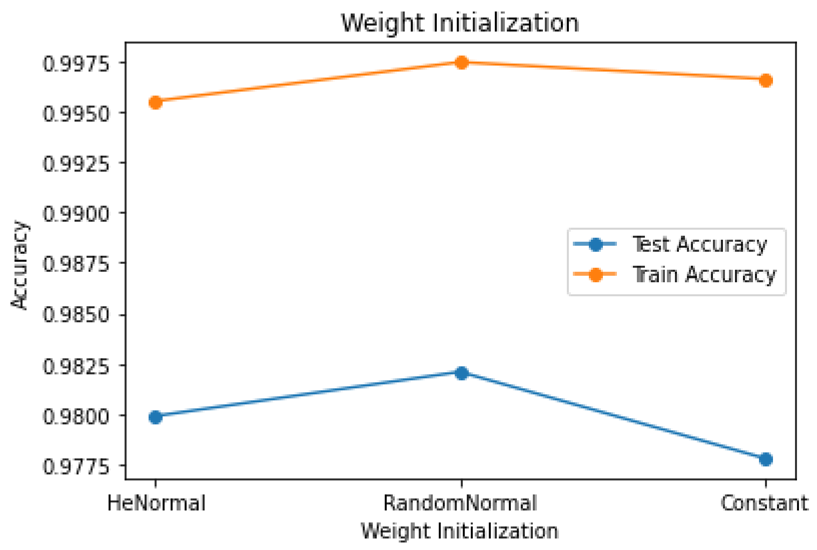


```
Best Hidden nodes = [1000, 100], with Test Accuracy = 0.9797


--------------------------------------------------
Learning Rate  =>  0.01
--------------------------------------------------
Test Loss           = 0.1502
Test Accuracy        = 0.9677
Test Precision       = 0.9679
Test Recall         = 0.9676
Test F1 score        = 0.9675
Test AUC            = 0.9989
```

```
Train Loss           = 0.0583
Train Accuracy       = 0.9842
Train Precision      = 0.9844
Train Recall         = 0.9843
Train F1 score       = 0.9842
Train AUC            = 0.9998

--------------------------------------------------
Learning Rate  =>  0.001
--------------------------------------------------
Test Loss            = 0.0814
Test Accuracy        = 0.9805
Test Precision       = 0.9805
Test Recall          = 0.9803
Test F1 score        = 0.9803
Test AUC             = 0.9997
Train Loss           = 0.0076
Train Accuracy       = 0.9976
Train Precision      = 0.9976
Train Recall         = 0.9975
Train F1 score       = 0.9976
Train AUC            = 1.0000

--------------------------------------------------
Learning Rate  =>  0.0001
--------------------------------------------------
Test Loss            = 0.0737
Test Accuracy        = 0.9765
Test Precision       = 0.9762
Test Recall          = 0.9764
Test F1 score        = 0.9763
Test AUC             = 0.9996
Train Loss           = 0.0362
Train Accuracy       = 0.9917
Train Precision      = 0.9916
Train Recall         = 0.9918
Train F1 score       = 0.9916
Train AUC            = 0.9999
```



```
Best Learning Rate = 0.001, with Test Accuracy = 0.9805



--------------------------------------------------
Weight Initialization  =>  <keras.initializers.initializers_v2.HeNormal object at 0x7fc29e4a2550>
--------------------------------------------------
Test Loss           = 0.0812
Test Accuracy         = 0.9799
Test Precision        = 0.9797
Test Recall          = 0.9798
Test F1 score         = 0.9797
Test AUC            = 0.9997
Train Loss           = 0.0132
Train Accuracy         = 0.9955
Train Precision        = 0.9955
Train Recall          = 0.9955
Train F1 score         = 0.9955
Train AUC            = 1.0000

--------------------------------------------------
Weight Initialization  =>  <keras.initializers.initializers_v2.RandomNormal object at 0x7fc29e4a2e80>
--------------------------------------------------
Test Loss           = 0.0826
Test Accuracy         = 0.9821
Test Precision        = 0.9820
Test Recall          = 0.9820
Test F1 score         = 0.9820
Test AUC            = 0.9996
Train Loss           = 0.0079
Train Accuracy         = 0.9975
Train Precision        = 0.9974
Train Recall          = 0.9975
Train F1 score        = 0.9975
Train AUC            = 1.0000

--------------------------------------------------
Weight Initialization  =>  <keras.initializers.initializers_v2.Constant object at 0x7fc29e4a25b0>
--------------------------------------------------
Test Loss            = 0.0977
Test Accuracy         = 0.9778
Test Precision        = 0.9777
Test Recall          = 0.9775
Test F1 score         = 0.9776
Test AUC             = 0.9995
Train Loss           = 0.0104
Train Accuracy        = 0.9966
Train Precision       = 0.9966
Train Recall         = 0.9966
Train F1 score        = 0.9966
Train AUC            = 1.0000
```



```
Best Weight Initialization = <keras.initializers.initializers_v2.RandomNormal object at
0x7fc29e4a2e80>, with Test Accuracy = 0.9821



Best hyperparameters: Hidden units = [1000, 100], learning rate = 0.001, Weight initialization =
<keras.initializers.initializers_v2.RandomNormal object at 0x7fc29e4a2e80>
```

Discussion on the best algorithm for the given datasets

To verify which algorithm suits best for given datasets, we compared the accuracies we got after using the optimal parameters. From that, we could conclude that the best algorithm can vary based on the dataset. So, for the dataset 1, we see that KNN provides the best accuracy for 99.1% and for dataset 2, logistic regression provides 78.5%.

Autoencoder for MNIST Image Denoising (Bonus)

Convolutional Autoencoder

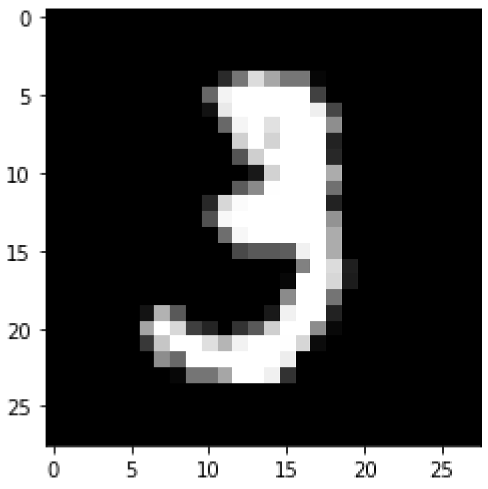
A convolutional Autoencoder is a type of deep learning model that can be used for image denoising. Image denoising refers to the process of removing noise, such as random pixel values or artifacts, from an image in order to improve its quality.

A convolutional Autoencoder is composed of two main parts: an encoder and a decoder. The encoder part of the model uses convolutional layers, which are a type of neural network layer that is designed to operate on data with a grid-like structure, such as an image. The convolutional layers are trained to compress the input image into a lower-dimensional representation, called a latent representation or code, that captures the most important features of the image.

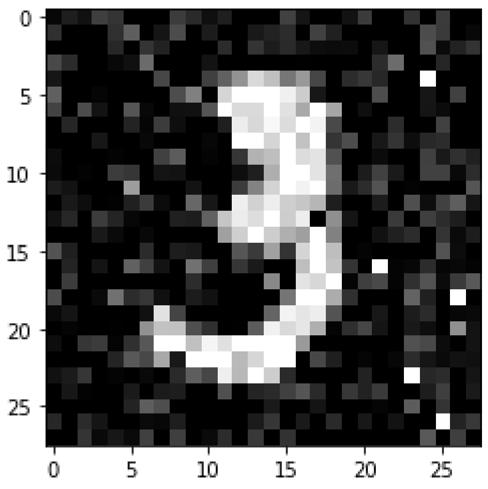
The decoder part of the model then takes this latent representation and uses it to reconstruct the denoised input image. When the model is used for image denoising, the input image is noisy, and the goal is to use the convolutional Autoencoder to remove the noise and produce a denoised version

First, we have used convolutional layers as they are better suited for image data than fully connected layers. We use MSE loss.

Original Image



Noisy Image



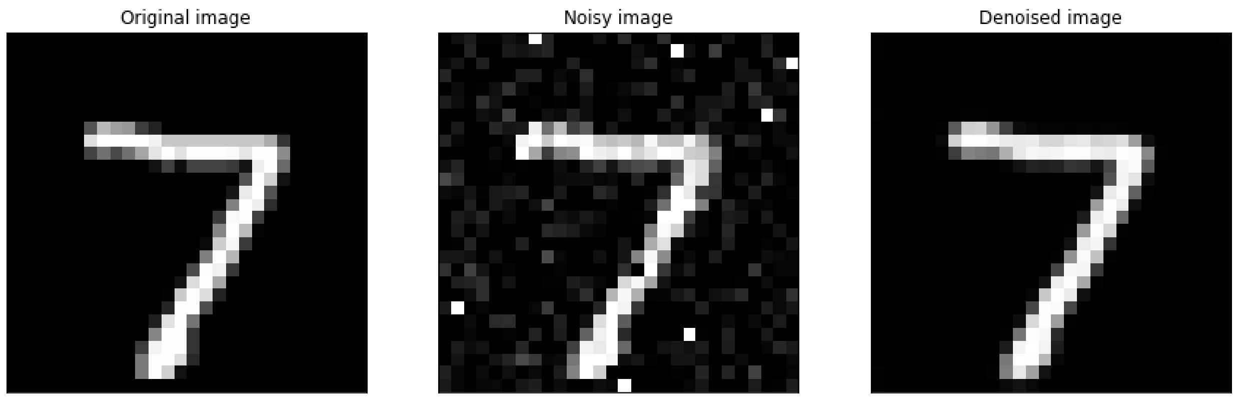
Model configuration

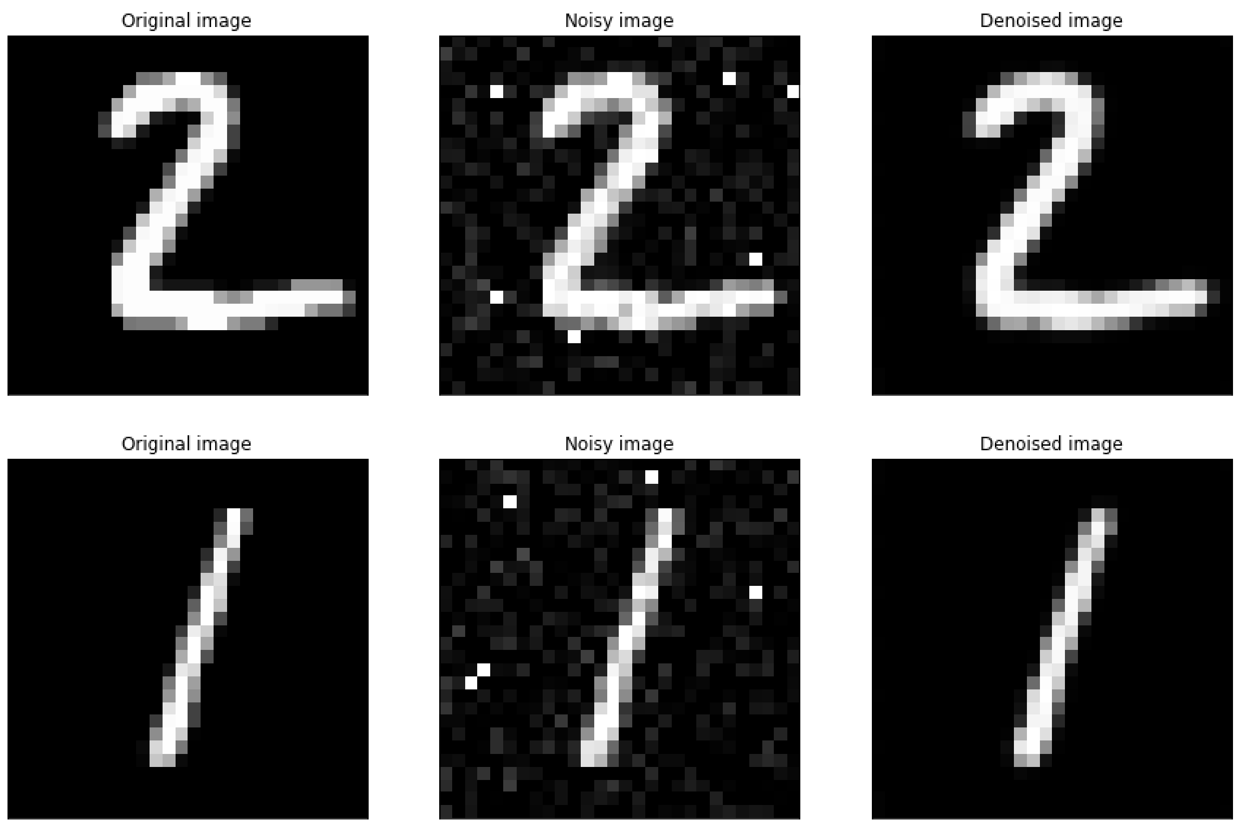
```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 conv2d_12 (Conv2D)           (None, 14, 14, 16)        160
 conv2d_13 (Conv2D)          (None, 7, 7, 8)           1160

 conv2d_transpose_8 (Conv2DT  (None, 14, 14, 8)        584
 ranspose)

 conv2d_transpose_9 (Conv2DT  (None, 28, 28, 16)       1168
 ranspose)

 conv2d_14 (Conv2D)          (None, 28, 28, 1)         145

=================================================================
Total params: 3,217
Trainable params: 3,217
Non-trainable params: 0
_________________________________________________________________
```

Results

```
Testing model before training
313/313 [==============================] - 1s 3ms/step - loss: 0.2328 - mse: 0.2328

Training evaluation:
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0029 - mse: 0.0029
Testing evaluation:
313/313 [==============================] - 1s 3ms/step - loss: 0.0029 - mse: 0.0029
```

Examples





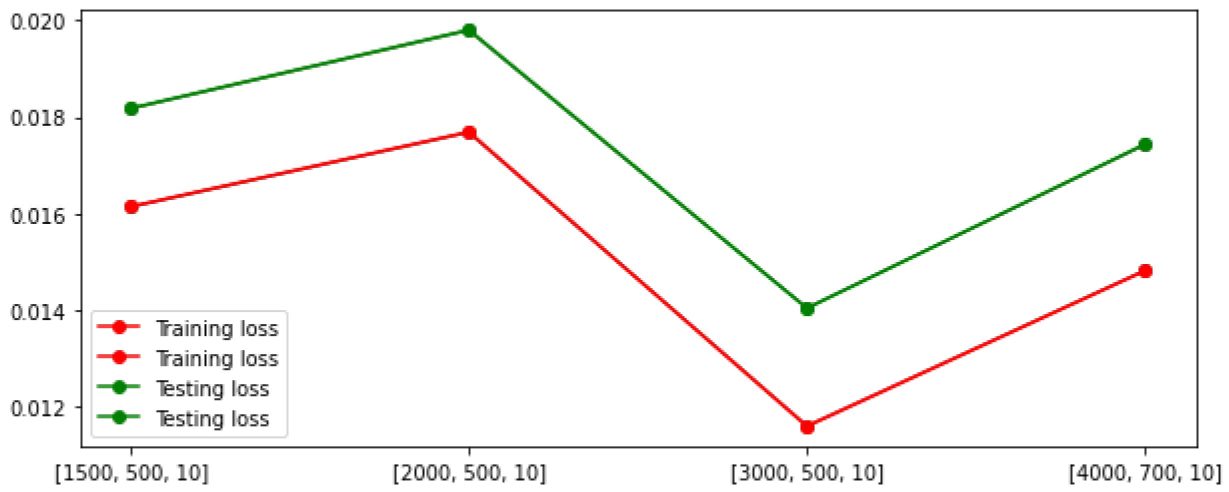
Fully Connected Layer Autoencoder

Note: Here when model hidden layer shape is referred to as (x, y, z) that is the shape of the encoder. The decoder will have a shape that is an exact mirror image of encoder shape, ie, (z, y, x).

We aim to compare the results of Convolutional Autoencoder to those of Fully Connected Layer Autoencoder.

We use MSE loss.

```
Model hidden layers = [3000, 500, 10]
Testing model before training
313/313 [==============================] - 1s 3ms/step - loss: 0.2311 - mse: 0.2311

Training evaluation:
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0116 - mse: 0.0116
Testing evaluation:
313/313 [==============================] - 1s 3ms/step - loss: 0.0140 - mse: 0.0140
==========================================================================
=================================
1875/1875 [==============================] - 3s 2ms/step
313/313 [==============================] - 1s 2ms/step
predTrain.shape = (60000, 28, 28), predTest.shape = (10000, 28, 28)
predTrain.shape = (60000, 28, 28), predTest.shape = (10000, 28, 28)


Model hidden layers = [4000, 700, 10]
Testing model before training
313/313 [==============================] - 1s 3ms/step - loss: 0.2311 - mse: 0.2311

Training evaluation:
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0148 - mse: 0.0148
Testing evaluation:
313/313 [==============================] - 1s 3ms/step - loss: 0.0174 - mse: 0.0174
==========================================================================
=================================
1875/1875 [==============================] - 3s 2ms/step
313/313 [==============================] - 1s 2ms/step
predTrain.shape = (60000, 28, 28), predTest.shape = (10000, 28, 28)
predTrain.shape = (60000, 28, 28), predTest.shape = (10000, 28, 28)
```



```
Models = [[1500, 500, 10], [2000, 500, 10], [3000, 500, 10], [4000, 700, 10]]. Loss test =
[[0.018176065757870674, 0.018176063895225525], [0.019795380532741547,
0.019795387983322144], [0.014047828502953053, 0.014047837816178799],
[0.017437027767300606, 0.01743701845407486]]. Best index = 2. Best Model = [3000, 500, 10]
```

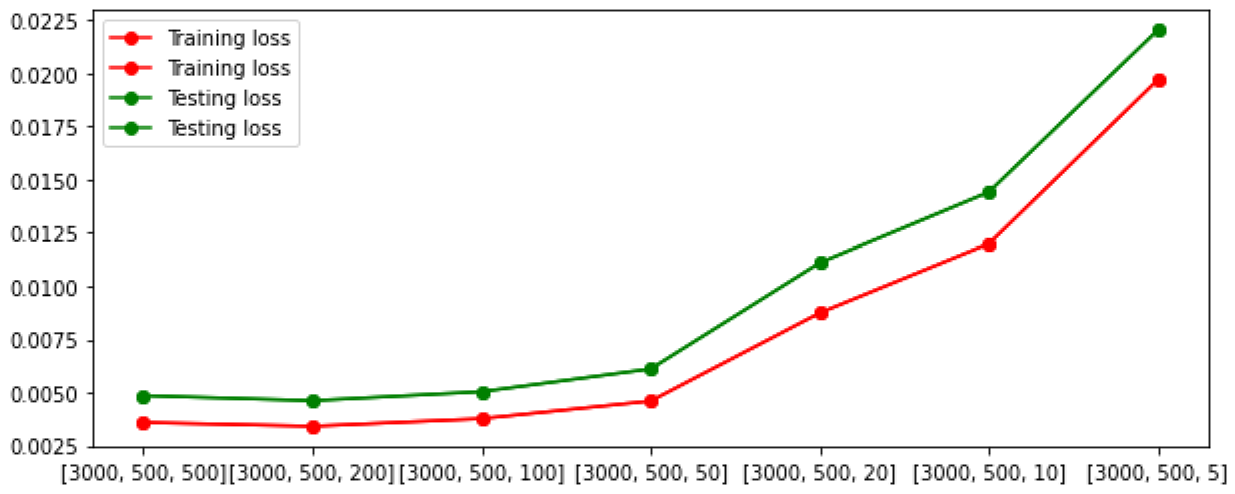
Best model out of above = [3000, 500, 10]

The best results may be seen for the encoder shape (3000, 500) and decoder shape (500, 3000) with the width of the narrowest layer being fixed to 10. For values smaller than this (eg-(1500, 500)) the loss is higher due to underfitting. For values bigger than this (eg-(4000, 700)) the loss is higher due to overfitting.

```
Model hidden layers = [3000, 500, 500]
Testing model before training
313/313 [==============================] - 1s 3ms/step - loss: 0.2310 - mse: 0.2310

Training evaluation:
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0036 - mse: 0.0036
Testing evaluation:
313/313 [==============================] - 1s 3ms/step - loss: 0.0049 - mse: 0.0049
==========================================================================================
==================================
1875/1875 [==============================] - 3s 2ms/step
313/313 [==============================] - 1s 2ms/step
predTrain.shape = (60000, 28, 28), predTest.shape = (10000, 28, 28)
predTrain.shape = (60000, 28, 28), predTest.shape = (10000, 28, 28)


Model hidden layers = [3000, 500, 200]
Testing model before training
313/313 [==============================] - 1s 3ms/step - loss: 0.2310 - mse: 0.2310

Training evaluation:
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0034 - mse: 0.0034
Testing evaluation:
313/313 [==============================] - 1s 3ms/step - loss: 0.0046 - mse: 0.0046
==========================================================================================
==================================
1875/1875 [==============================] - 3s 2ms/step
313/313 [==============================] - 1s 2ms/step
predTrain.shape = (60000, 28, 28), predTest.shape = (10000, 28, 28)
predTrain.shape = (60000, 28, 28), predTest.shape = (10000, 28, 28)


Model hidden layers = [3000, 500, 100]
Testing model before training
313/313 [==============================] - 1s 3ms/step - loss: 0.2312 - mse: 0.2312

Training evaluation:
```

```
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0038 - mse: 0.0038
Testing evaluation:
313/313 [==============================] - 1s 3ms/step - loss: 0.0050 - mse: 0.0050
===========================================================================
==================================
1875/1875 [==============================] - 3s 2ms/step
313/313 [==============================] - 1s 2ms/step
predTrain.shape = (60000, 28, 28), predTest.shape = (10000, 28, 28)
predTrain.shape = (60000, 28, 28), predTest.shape = (10000, 28, 28)


Model hidden layers = [3000, 500, 50]
Testing model before training
313/313 [==============================] - 1s 3ms/step - loss: 0.2309 - mse: 0.2309

Training evaluation:
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0046 - mse: 0.0046
Testing evaluation:
313/313 [==============================] - 1s 3ms/step - loss: 0.0061 - mse: 0.0061
===========================================================================
==================================
1875/1875 [==============================] - 3s 2ms/step
313/313 [==============================] - 1s 2ms/step
predTrain.shape = (60000, 28, 28), predTest.shape = (10000, 28, 28)
predTrain.shape = (60000, 28, 28), predTest.shape = (10000, 28, 28)


Model hidden layers = [3000, 500, 20]
Testing model before training
313/313 [==============================] - 1s 3ms/step - loss: 0.2310 - mse: 0.2310

Training evaluation:
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0087 - mse: 0.0087
Testing evaluation:
313/313 [==============================] - 1s 3ms/step - loss: 0.0111 - mse: 0.0111
===========================================================================
==================================
1875/1875 [==============================] - 3s 2ms/step
313/313 [==============================] - 1s 2ms/step
predTrain.shape = (60000, 28, 28), predTest.shape = (10000, 28, 28)
predTrain.shape = (60000, 28, 28), predTest.shape = (10000, 28, 28)


Model hidden layers = [3000, 500, 10]
Testing model before training
313/313 [==============================] - 1s 3ms/step - loss: 0.2310 - mse: 0.2310

Training evaluation:
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0120 - mse: 0.0120
```

```
Testing evaluation:
313/313 [==============================] - 1s 3ms/step - loss: 0.0144 - mse: 0.0144
============================================================================
==================================
1875/1875 [==============================] - 3s 2ms/step
313/313 [==============================] - 1s 2ms/step
predTrain.shape = (60000, 28, 28), predTest.shape = (10000, 28, 28)
predTrain.shape = (60000, 28, 28), predTest.shape = (10000, 28, 28)


Model hidden layers = [3000, 500, 5]
Testing model before training
313/313 [==============================] - 1s 3ms/step - loss: 0.2311 - mse: 0.2311

Training evaluation:
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0197 - mse: 0.0197
Testing evaluation:
313/313 [==============================] - 1s 3ms/step - loss: 0.0220 - mse: 0.0220
============================================================================
==================================
1875/1875 [==============================] - 3s 2ms/step
313/313 [==============================] - 1s 2ms/step
predTrain.shape = (60000, 28, 28), predTest.shape = (10000, 28, 28)
predTrain.shape = (60000, 28, 28), predTest.shape = (10000, 28, 28)
```



The loss increases as we make the narrowest layer smaller in size. This is because very few neurons cannot encode the original data without significant loss of information.