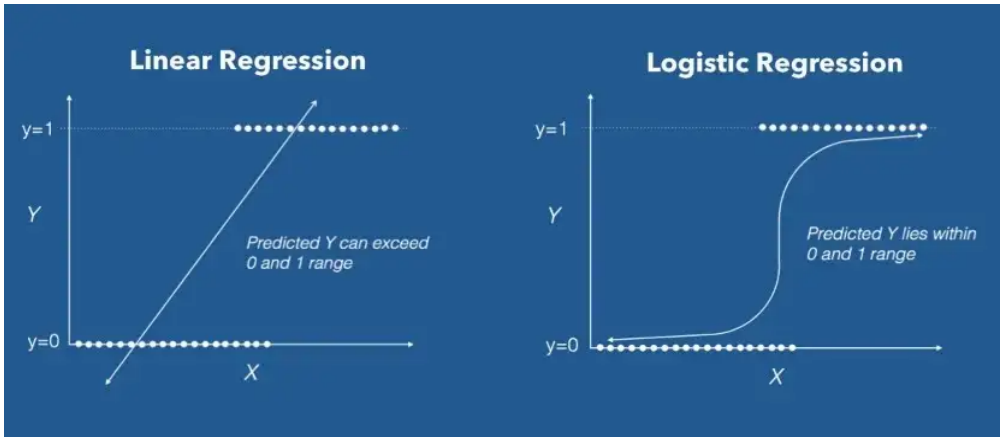
CS6316 Machine Learning Practical Project Report

Keshara Weerasinghe (cjh9fw), Rishabh Jain (tpe3sj), Jiho Lee (qxz6hb)

---

1. Logistic Regression

Similar to linear regression, logistic regression estimates the probability of an event occurring. Given a dataset, the algorithm will be able to provide an output bounded between 0 and 1. This algorithm can be used when the target variable is categorical. This algorithm can be extended from binary classification to a multinomial classification as well.



Linear Regression VS Logistic Regression Graph | Source: Data Camp and Introduction to Logistic Regression by A. Pant

Sigmoid function is used to map the predicted values to probabilities as it can map any real value to a value between 0 and 1.

$$\sigma(a) = \frac{1}{1 + \exp(-a)}$$

Similar to linear regression, logistic regression uses a cost function to optimize the objective in order to create an accurate model with minimum error. By using **Gradient Descent** algorithm, logistic regression will reduce the cost to the minimum it can.

Results and Evaluation

We preprocess the data by **standardizing** the dataset as they might behave badly if the data does not look like a normally distributed data. We use libraries from Sklearn to do the preprocessing.

Initially we use the default hyperparameters defined in the library to train our classifier with 10 fold cross validation. Figure 1 shows the evaluation metrics for both datasets. We were able to achieve an **Accuracy** of **0.96** for dataset 1 and **0.72** for dataset 2. Dataset 2 does not fit well with the base classifier.

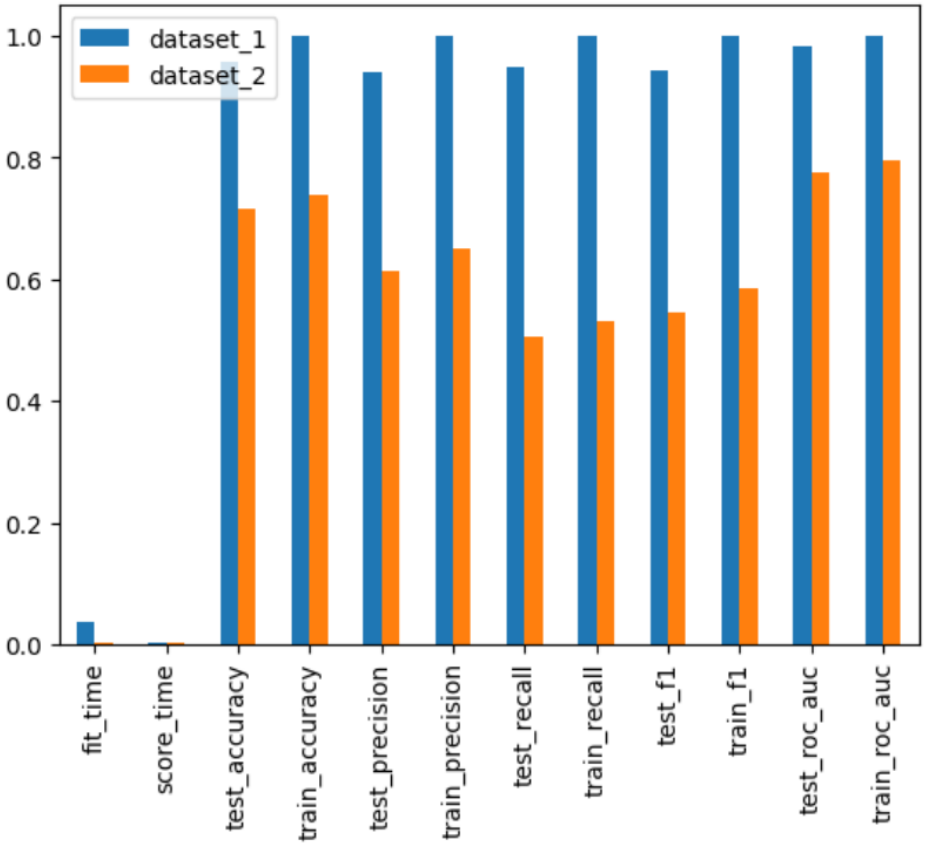


Figure 1: Results for Logistic Regression for Dataset 1 and 2 with default hyperparameters

Next, we use hyperparameter tuning to tune our logistic regression model to better fit the dataset. Our target is to improve accuracy and we use **GridsearchCV** function from Sklearn which performs an exhaustive search over the specified hyperparameters. In this algorithm, we tune **C** which controls the regularization but it is the inverse of regularization strength. We use 10-fold cross validation to optimize the hyper-parameter selection which can be defined in the **GridsearchCV** function itself.

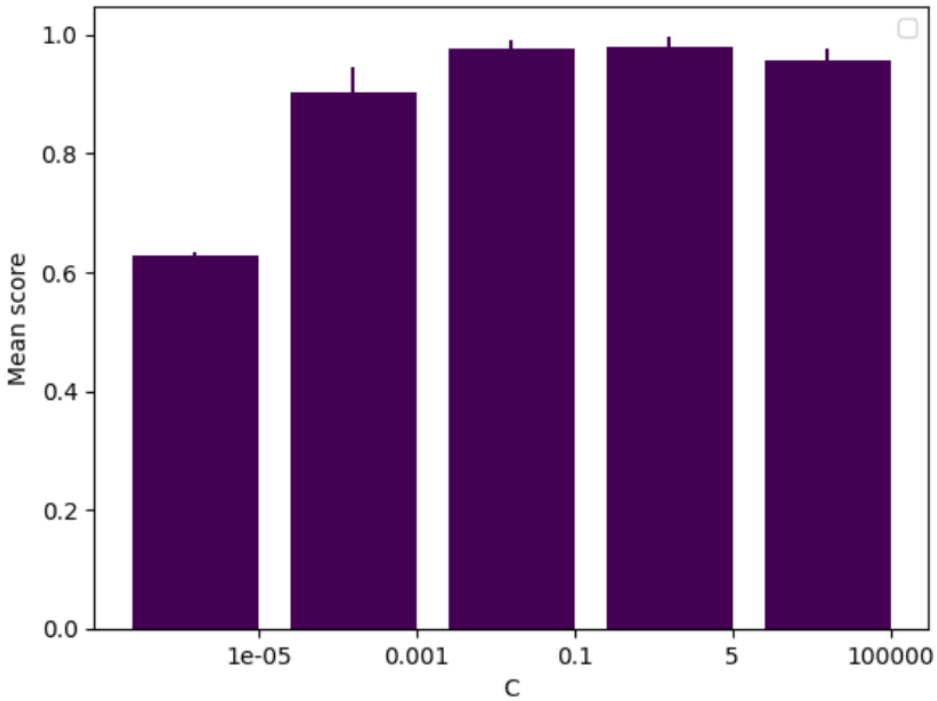


Figure 2: Test accuracy of Dataset 1 after hyperparameter tuning

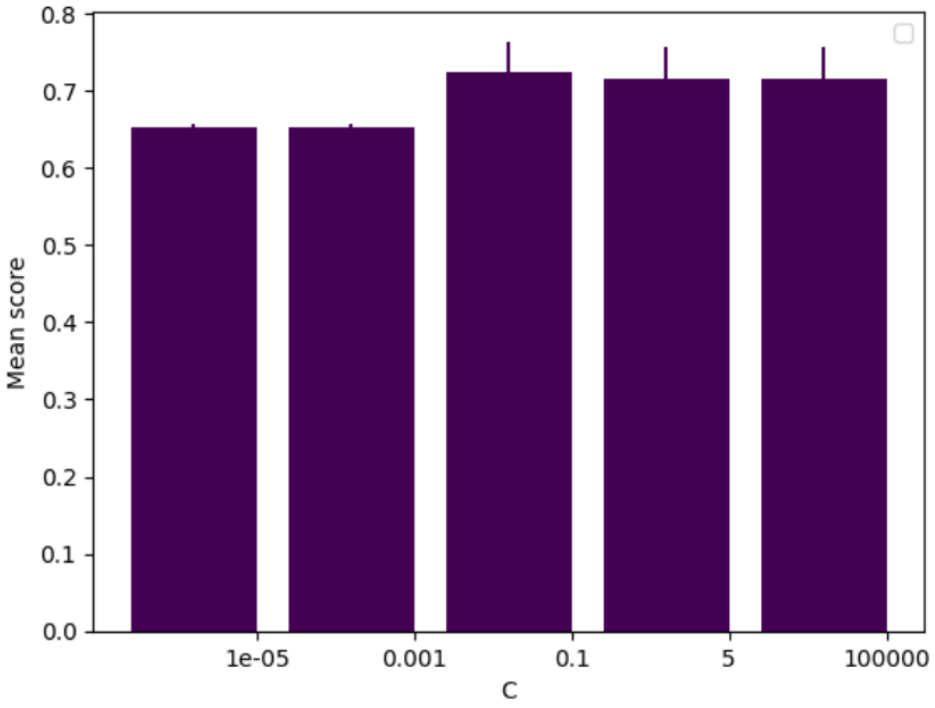


Figure 3: Test accuracy of Dataset 2 after hyperparameter tuning

Figure 2 and 3 shows that having a very small value for **C** which is having a very strong regularization lead to underfitting the data and having a very weak regularization will make the model overfit the training data. We see a slight dip in the accuracy when we decrease the regularization strength (increase **C**). We believe the understanding of bias-variance tradeoff is highly important to choose appropriate parameters for the dataset and that can improve the performance significantly.

---

2. KNN - K Nearest Neighbor

KNN predicts the outcome by calculating the distance between a certain data point and among all other data points and select a **K** number of neighbors and look at the majority class those neighbors belong to.

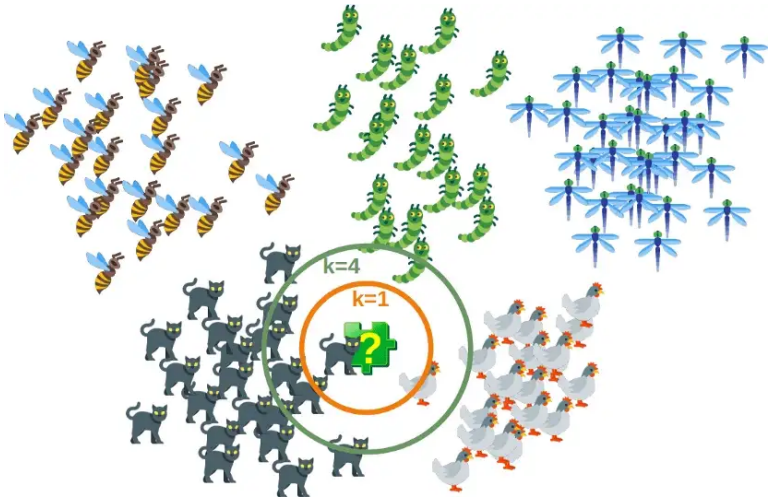


Figure 4: Intuition behind KNN | Source: K-Nearest Neighbor by Antony Christopher
https://medium.com/swlh/k-nearest-neighbor-ca2593d7a3c4

The algorithm behind KNN is quite simple and straightforward.

Step-1: Select **K;** the number of the neighbors
Step-2: Calculate the Euclidean distance of neighbors
Step-3: Take the **K** nearest neighbors as per the calculated Euclidean distance.
Step-4: Among these **K** neighbors, find the majority class they belong to
Step-5: Assign the output of the given data point to the majority class

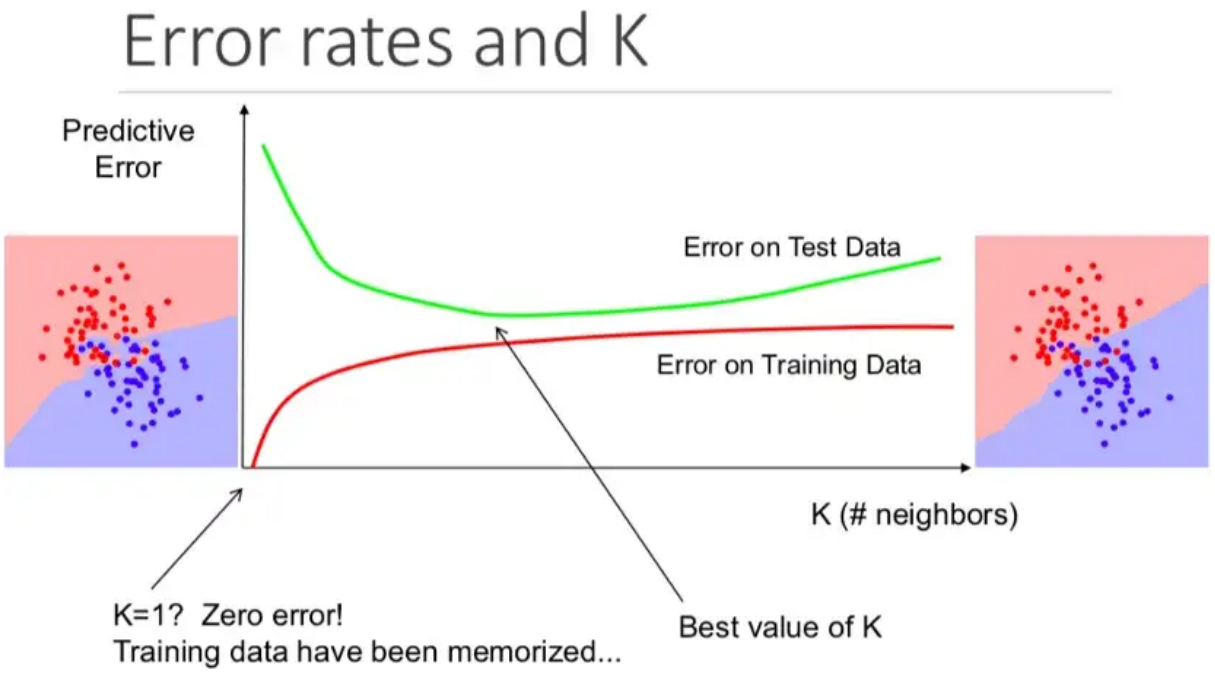


Figure 5: Error rate relationship with K Taken from K-Nearest Neighbors and Bias–Variance Tradeoff by Tzu-Chi Lin

Important thing to note is the selection of **K**. The number of neighbors will directly impact the bias-variance tradeoff. We will use Figure 6 to understand this. If we select **K** as 1 being the extreme case, the model remember the training points perfectly and will achieve perfect accuracy for the training dataset. However, we will have a very complex decision boundary and we will overfit the data. This will perform poorly on the Test data.
Moreover, having a very large **K** will lead to a very simple boundary as it will consider many neighbors and points from other classes will also fall into the same neighborhood. This will lead to underfitting the data.

Results and Evaluation

We preprocess the data by **standardizing** the dataset as they might behave badly if the data does not look like a normally distributed data. We use libraries from Sklearn to do the preprocessing.

Initially we use **K = 3** with 10 fold cross validation. Figure 6 shows the evaluation metrics for both datasets. We were able to achieve an **Accuracy** of **0.96** for dataset 1 and **0.67** for dataset 2. Dataset 2 does not fit well with the base classifier.

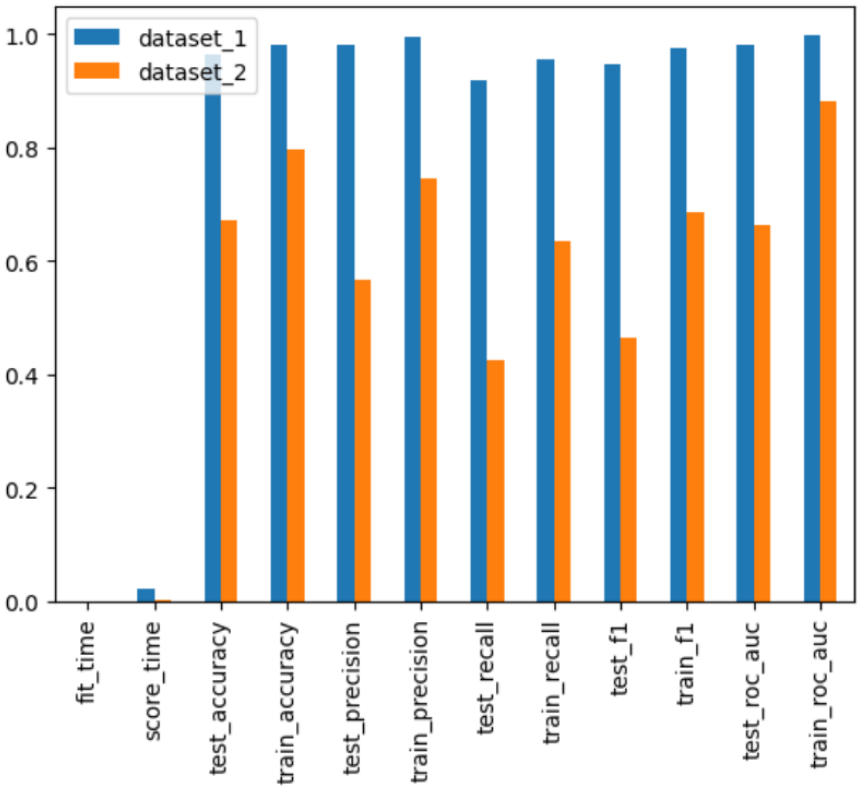


Figure 6: Results for KNN for Dataset 1 and 2 with with K=3

Next, we use hyperparameter tuning to tune the KNN model to better fit the dataset. We tune **K** by using the previously mentioned **GridsearchCV** function. We use 10-fold cross validation to optimize the hyper-parameter selection.

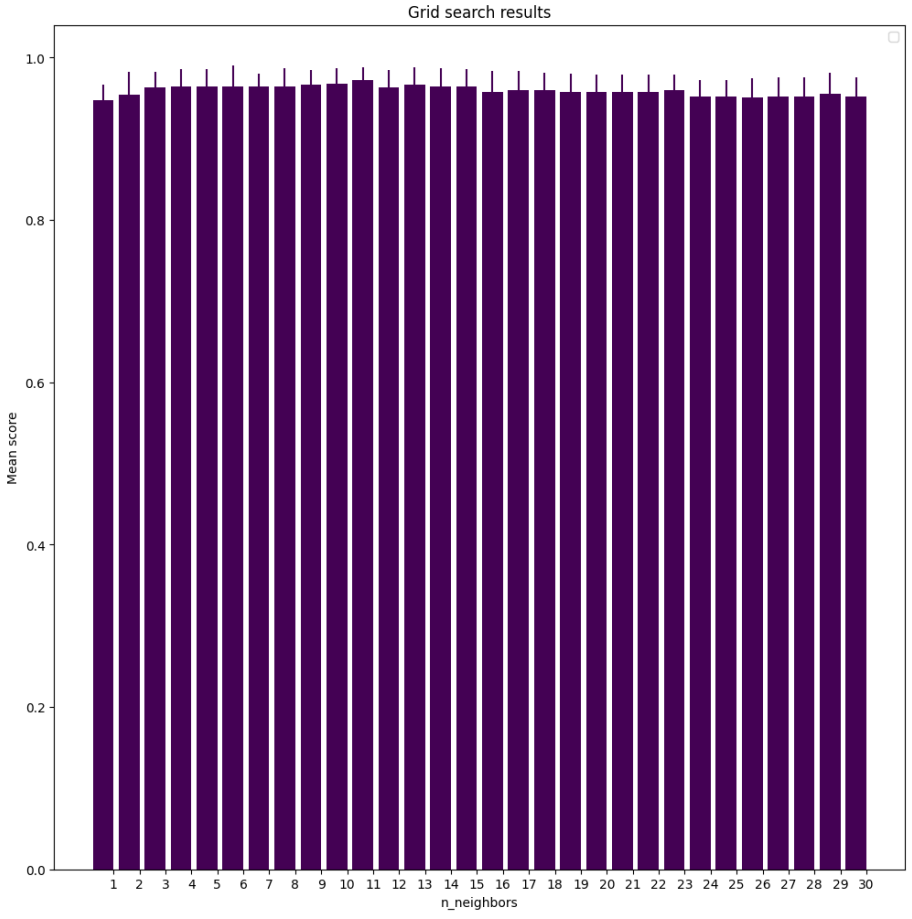


Figure 7: Test accuracy of Dataset 1 after hyperparameter tuning

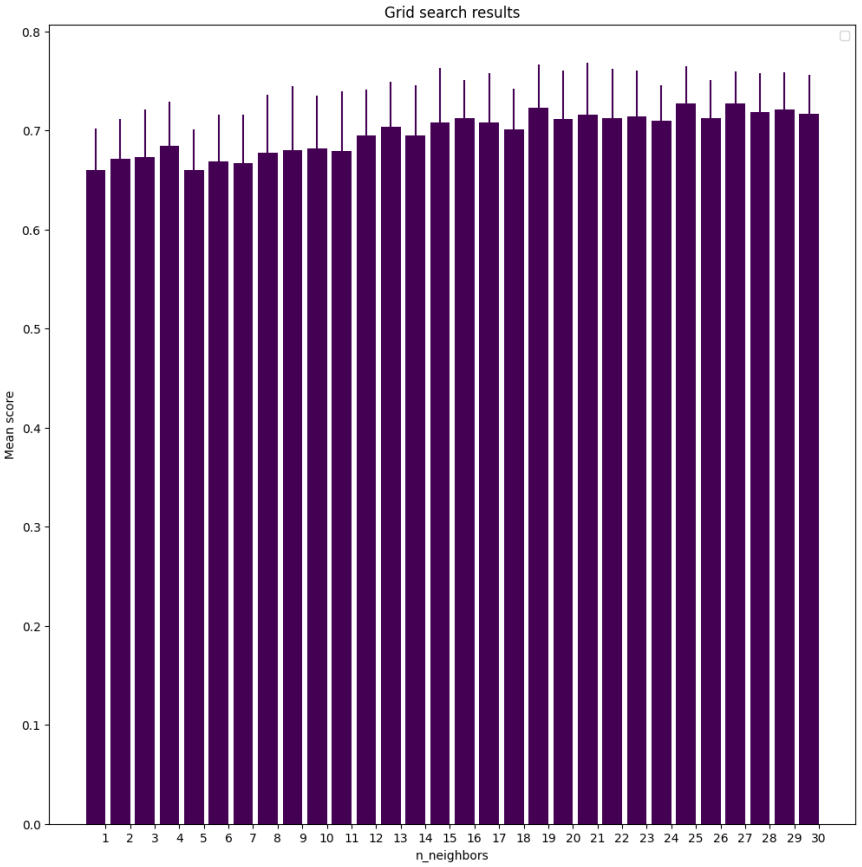


Figure 8: Test accuracy of Dataset 2 after hyperparameter tuning

	dataset_1	dataset_2
fit_time	0.000150	0.000352
score_time	0.016301	0.002198
test_accuracy	0.947274	0.660176
train_accuracy	1.000000	1.000000
test_precision	0.944926	0.559111
train_precision	1.000000	1.000000
test_recall	0.914719	0.437500
train_recall	1.000000	1.000000
test_f1	0.927124	0.453170
train_f1	1.000000	1.000000
test_roc_auc	0.940455	0.608266
train_roc_auc	1.000000	1.000000

Figure 9: Evaluation Metrics for Dataset 1 and 2 with K=1

Figure 9 shows that having **K = 1** will indeed give a perfect accuracy for training but perform worse for test data as it overfits.

Figure 7 and 8 shows the appropriate value for **K** after the hyperparameter tuning will increase the accuracy of our model. For dataset 1, **K = 11** gave the best results and for dataset 2, **K = 25** performed best.

Choosing an appropriate number of neighbors plays a key role in the accuracy of KNN classifier.

3. Decision Tree Classifier

This is another supervised learning algorithm which learns simple decision rules using a greedy method based on the training data and provide a decision for a given input. The decisioning process begins from the root of the tree and travels through its nodes.

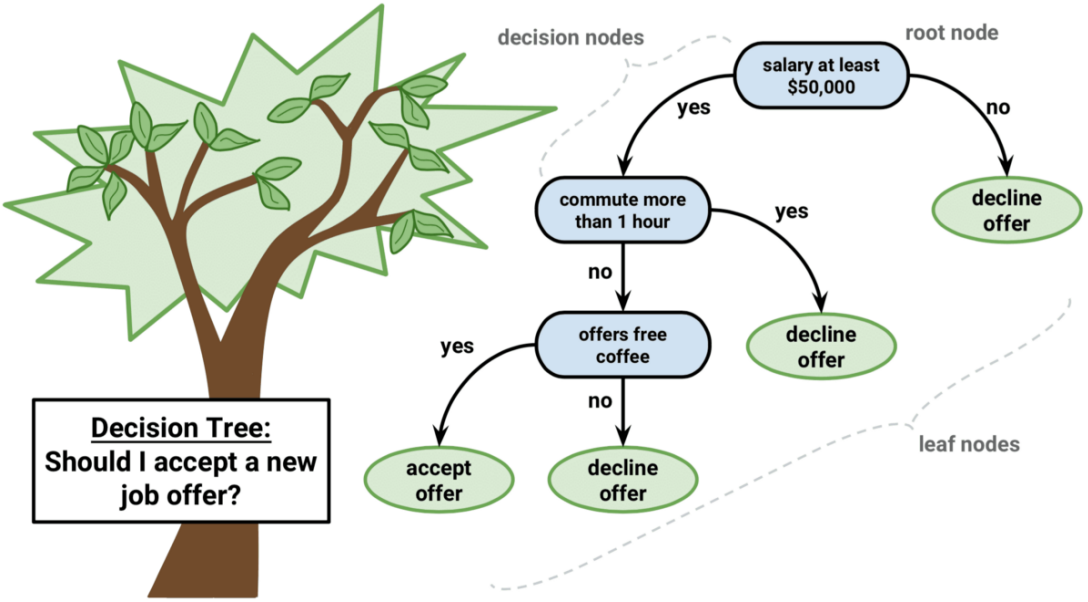


Figure 10: Intuition behind Decision Trees| Source: https://regenerativetoday.com/simple-explanation-on-how-decision-tree-algorithm-makes-decisions/

The algorithm behind building a Decision Tree is quite simple and straightforward. The tree is built in a top-down approach.

Step-1: Start with a single node containing all the data points
Repeat:
Step-2: Look at all current leaves and possible splits
Step-3: Select the split which most reduces the uncertainty in prediction (Gini, Entropy)

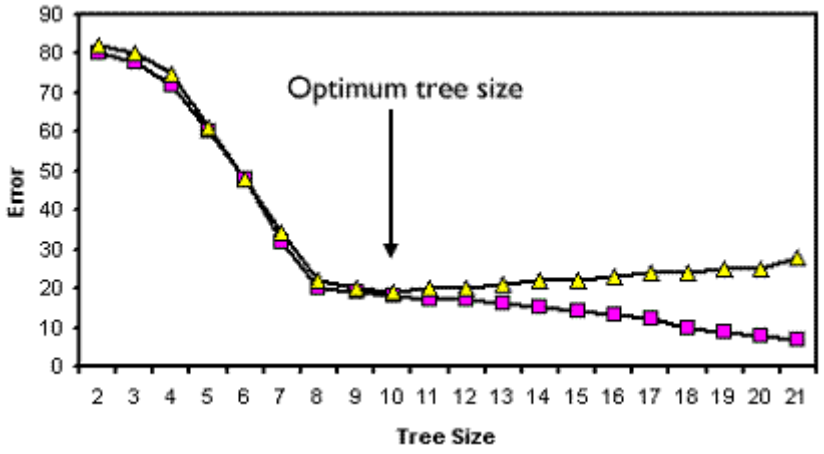


Figure 11: Error relationship with tree depth Taken from How Trees are Built and Pruned by Alan Fielding

Similar to KNN, selecting the tree depth (how large the tree is) directly impacts the bias-variance tradeoff. We will use Figure 11 to understand this. If we have a very small trees with very shallow depth, we do not

have enough decisioning nodes. That means, we will decide using very few information and our model will underfit the dataset. That is why we can see a high error for both training and test data in the above figure.

However, creating a very deep tree may also affect the performance of the model. As we go deeper, our decisioning process is highly complex and overfitted to the training dataset. We may have perfect accuracy for training but our tree wont generalize well..

Results and Evaluation

Similar to previous algorithms, we preprocess the data before feeding it to the classification model.

Initially we use the default hyperparameters for the decision tree function from Sklearn. We do not set the max depth and by default, the tree will expand until all leaves are pure or until all leaves contain less than a minimum number of samples required to split a node.. Figure 7 shows the evaluation metrics for both datasets. We were able to achieve an **Accuracy** of **0.93** for dataset 1 and **0.70** for dataset 2. Dataset 2t fit better with this classifier compared to previous classifiers.

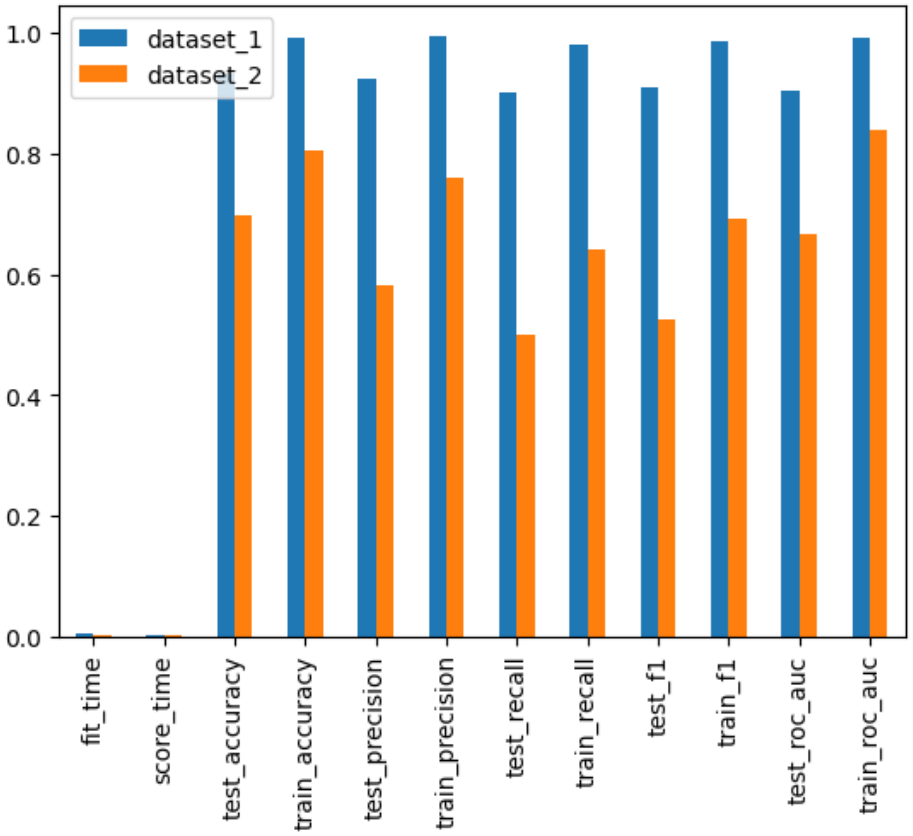


Figure 12: Results for Decision Tree Classifier for Dataset 1 and 2 with with default hyperparameters

Next, we use hyperparameter tuning to tune our Decision Tree Classifier to better fit the dataset. We tune 2 parameters; **max_depth** and **min_samples_split** by using the previously mentioned **GridsearchCV** function. We use 10-fold cross validation to optimize the hyper-parameter selection.

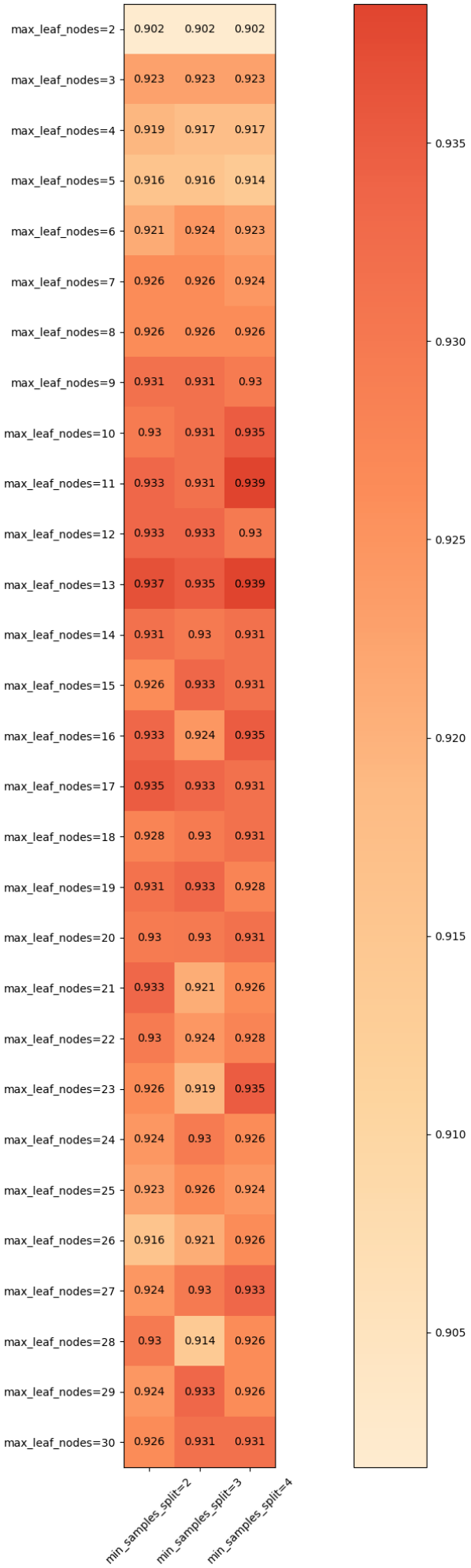


Figure 13: Test accuracy of Dataset 1 after hyperparameter tuning

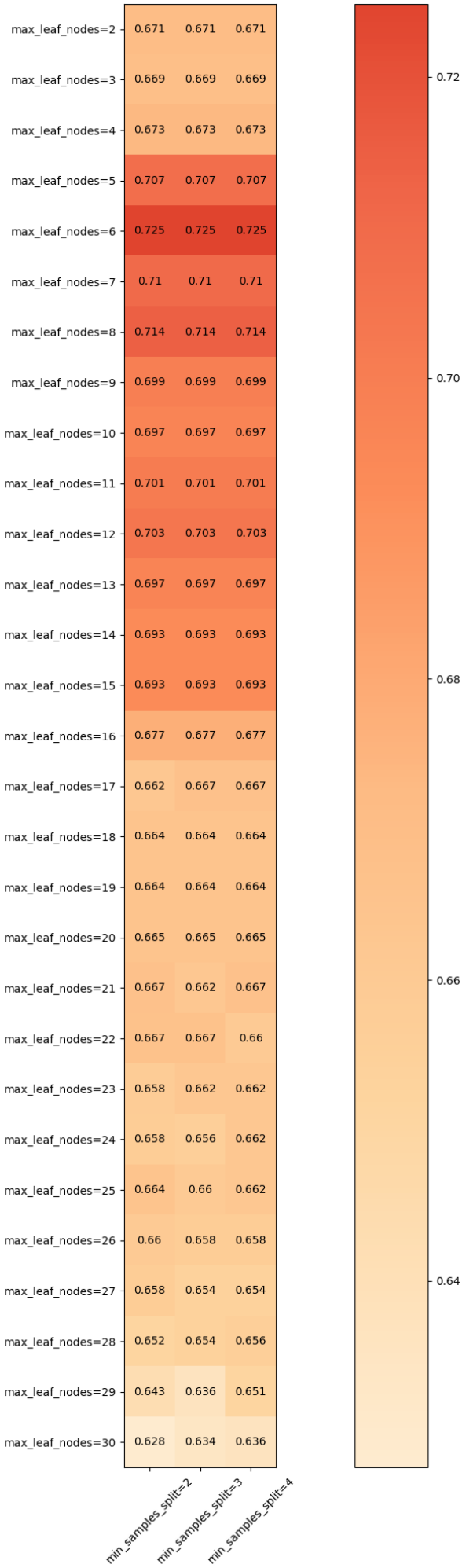


Figure 14: Test accuracy of Dataset 2 after hyperparameter tuning

	dataset_1	dataset_2
fit_time	0.004198	0.001800
score_time	0.002797	0.002900
test_accuracy	0.929731	0.696947
train_accuracy	0.991017	0.804233
test_precision	0.929956	0.583373
train_precision	0.995230	0.760943
test_recall	0.882035	0.500000
train_recall	0.980603	0.640972
test_f1	0.903748	0.526644
train_f1	0.987853	0.693063
test_roc_auc	0.890592	0.665800
train_roc_auc	0.992958	0.839497

Figure 15: Evaluation Metrics for Dataset 1 and 2 with max_depth set to 50

Figure 15 shows that having a very deep tree may give better accuracy for train data but perform worse for test data as the model overfits the training data.

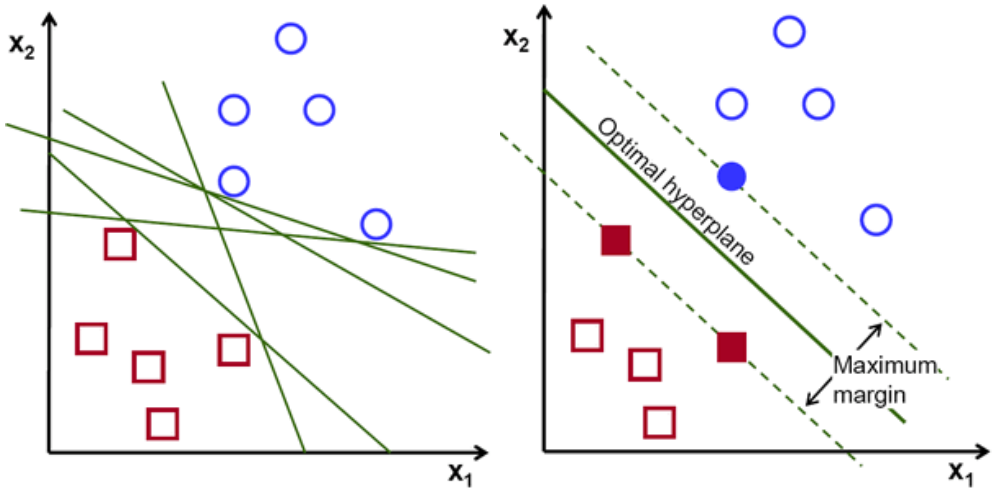
Figure 13 and 14 shows that after hyperparameter tuning, for Dataset 1, having max depth of 11 nodes and min samples to split of 4 yields the best accuracy. Similarly for dataset 2, having a max depth of 6 nodes and min samples to split of 2 yields the best accuracy.

Choosing an appropriate depth for the decision tree plays a key role in the accuracy of KNN classifier.

---

4. SVM - Support Vector Machine

The objective of the support vector machine algorithm is to find a decision boundary or a hyperplane that distinctly classifies the data points.



Finding a decision boundary or a hyperplane | Source: Support Vector Machine — Introduction to Machine Learning Algorithms by Rohith Gandhi

The objective is to find the hyperplane with the maximum margin between between two different classes

Results and Evaluation

Similar to previous algorithms, we preprocess the data before feeding it to the classification model.

Initially we use the default hyperparameters defined in the library to train our classifier with 10 fold cross validation. Figure 3 shows the evaluation metrics for both datasets. We were able to achieve an **Accuracy** of **0.95** for dataset 1 and **0.72** for dataset 2.

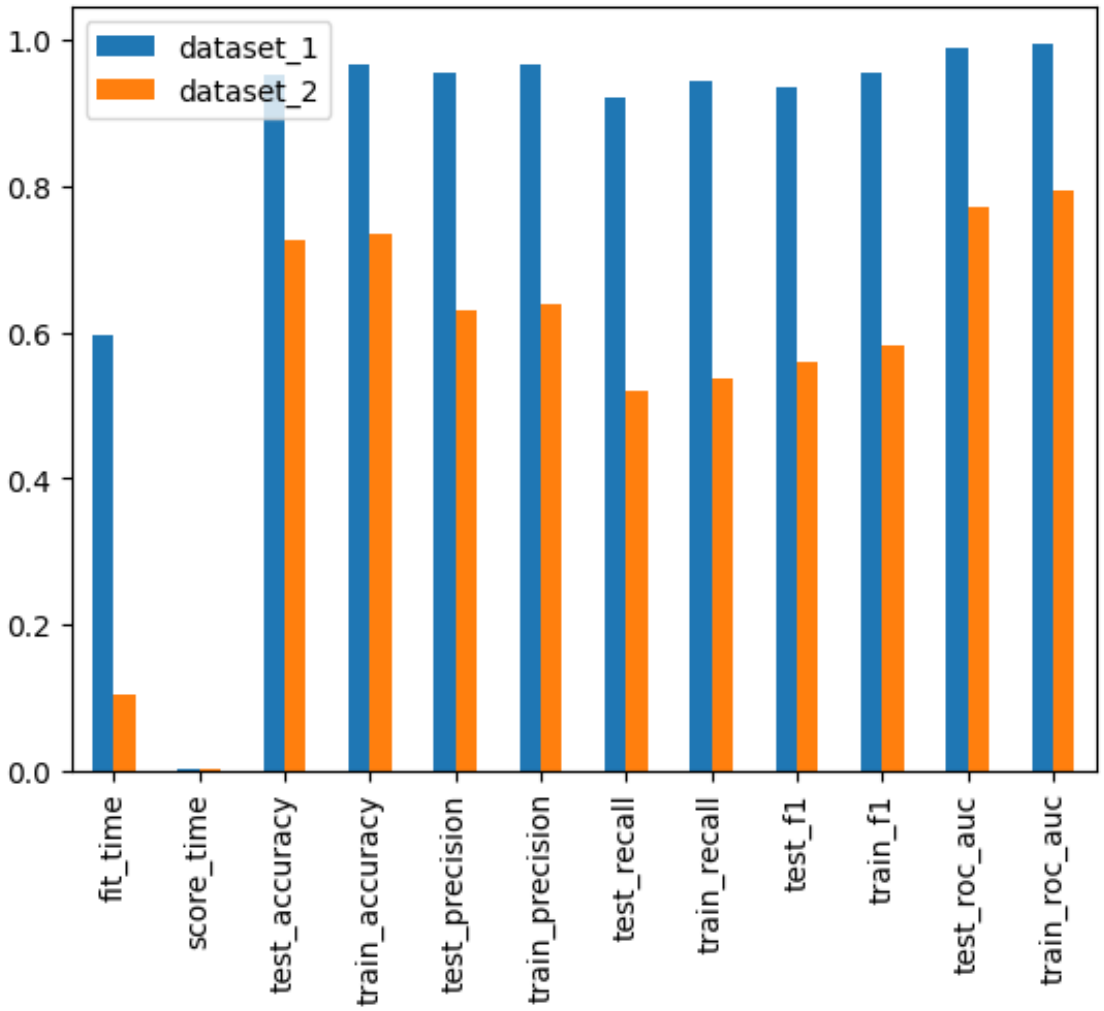


Figure 16: Results for SVM for Dataset 1 and 2 with default hyperparameters

Next, we use hyperparameter tuning to tune our SVM model to better fit the dataset. Our target is to improve accuracy and we use **GridsearchCV** function from Sklearn which performs an exhaustive search over the specified hyperparameters.

Here we tune 2 parameters. **C** which controls the regularization. Similar to Logistic Regression, **C** is inversely proportional to the regularization strength. We also try different kernels to see if we can achieve better performance.

We use 10-fold cross validation to optimize the hyper-parameter selection which can be defined in the **GridsearchCV** function itself.

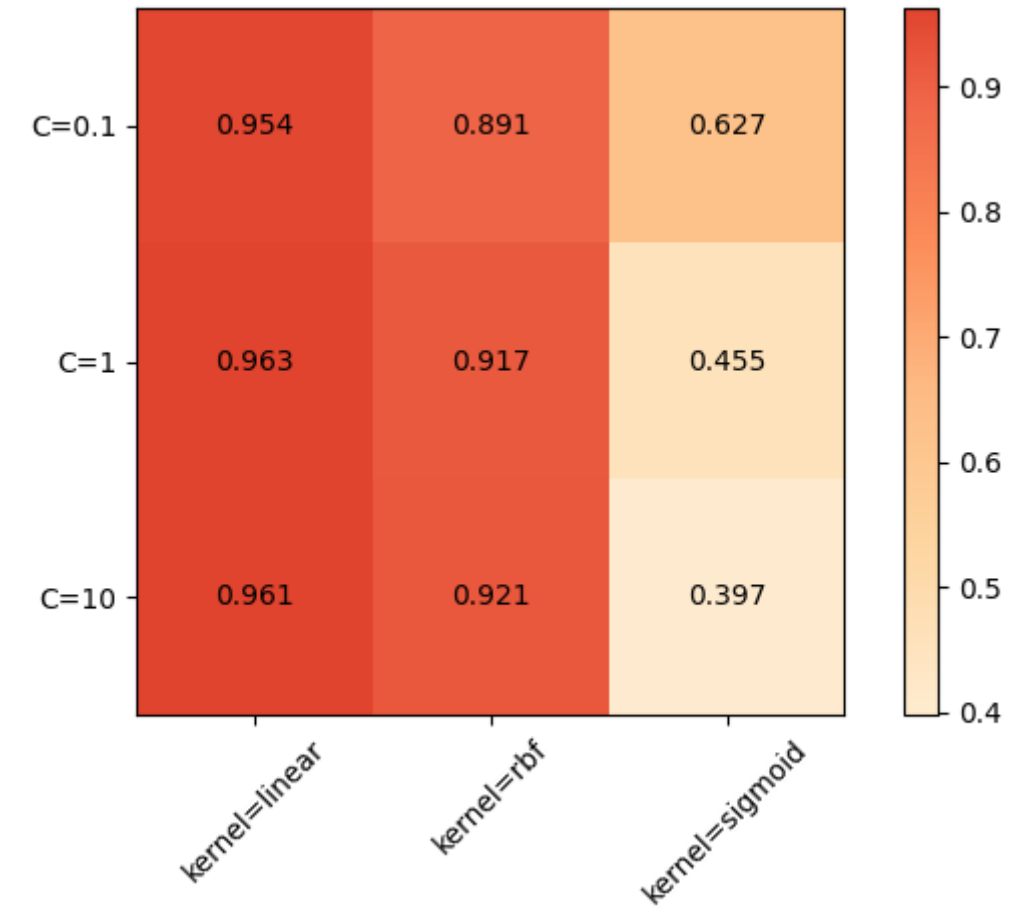


Figure 17: Test accuracy of Dataset 1 after hyperparameter tuning

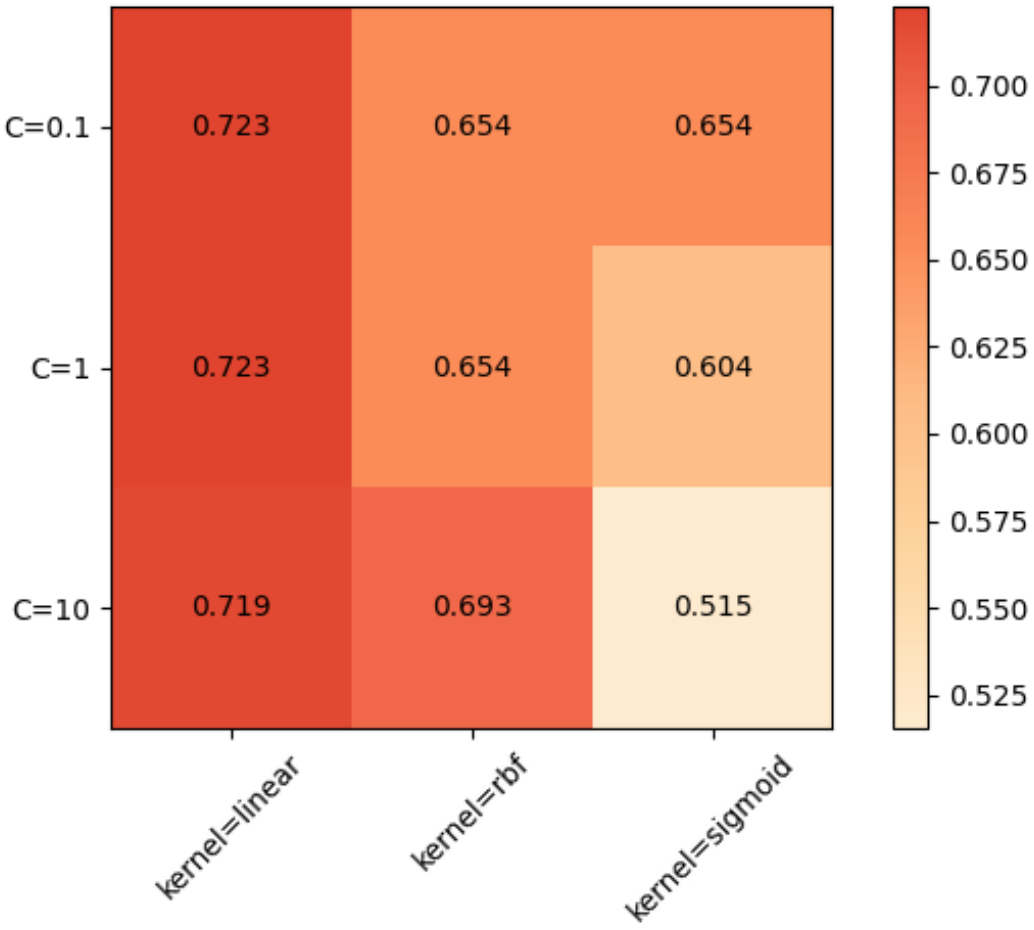


Figure 18: Test accuracy of Dataset 2 after hyperparameter tuning

Figure 17 shows that after hyperparameter tuning, **C** having a value of 1 and a **linear** kernel gave us slightly better results for Dataset 1. However, for Dataset 2, we could not see a significant improvement in accuracy despite the hyperparameter tuning.

5. Random Forest

Random forest is a supervised machine learning algorithm. Random forest uses multiple decision trees and makes predictions accordingly. It trains multiple decision trees on data subsets and averages the different multiple decision trees to create the final model.
The figure below gives the architecture of a random forest using three independent decision trees trained on three subsets. Each independent tree makes a prediction and then the predictions are averaged to produce the output.

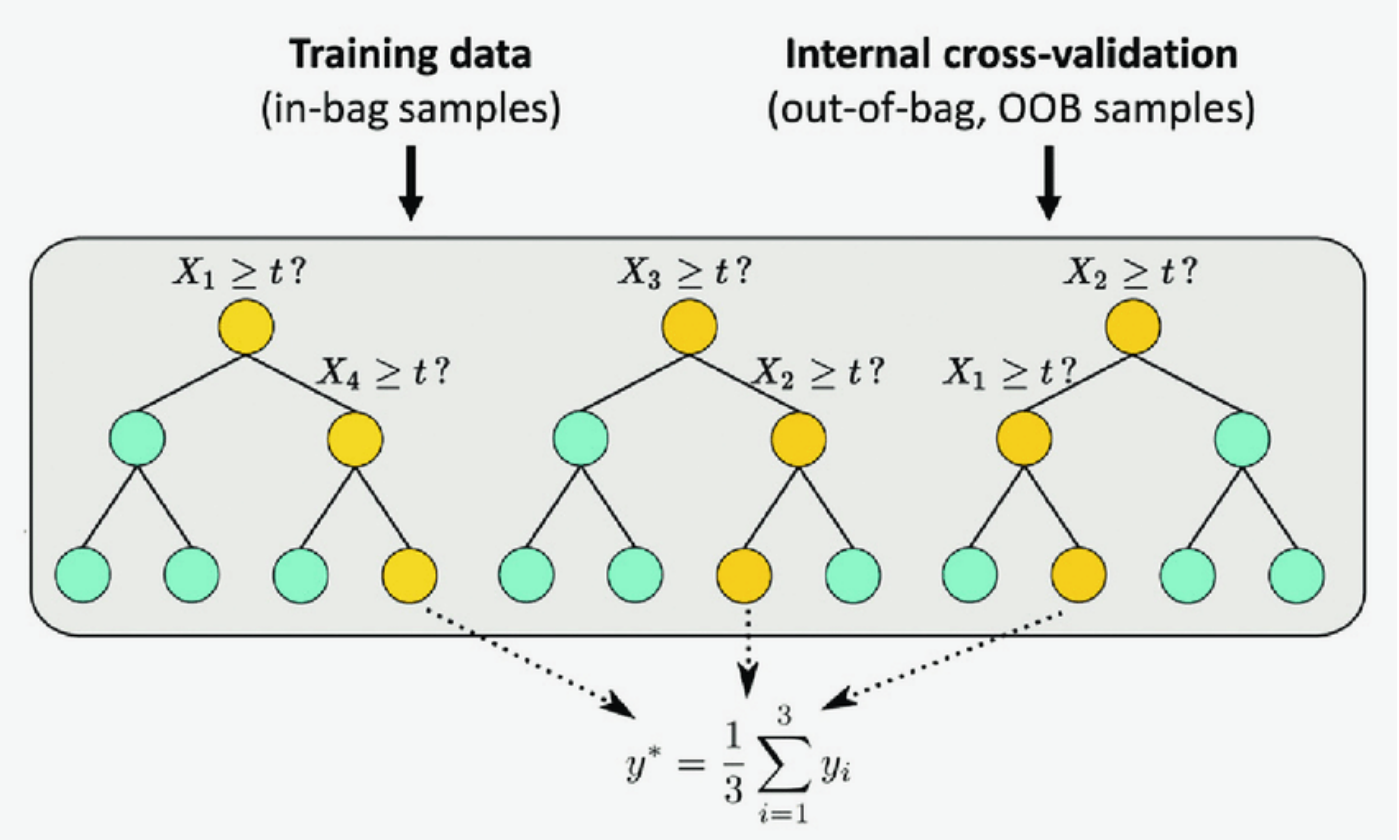


Figure 19. Random Forest Architecture

The algorithm for a random forest is as follows:

1. Bag (split) the dataset into multiple datasets, called subsets
2. Each bag is trained independently to create separate decision trees
3. Each decision tree provides a separate prediction
4. Average the predictions from each decision tree to make a final prediction.

Random forests are popularly used to handle large datasets, unbalanced and high-dimensional data or data with outliers or missing values. Random forests are popular for both regression and classification tasks.

Results and Evaluation

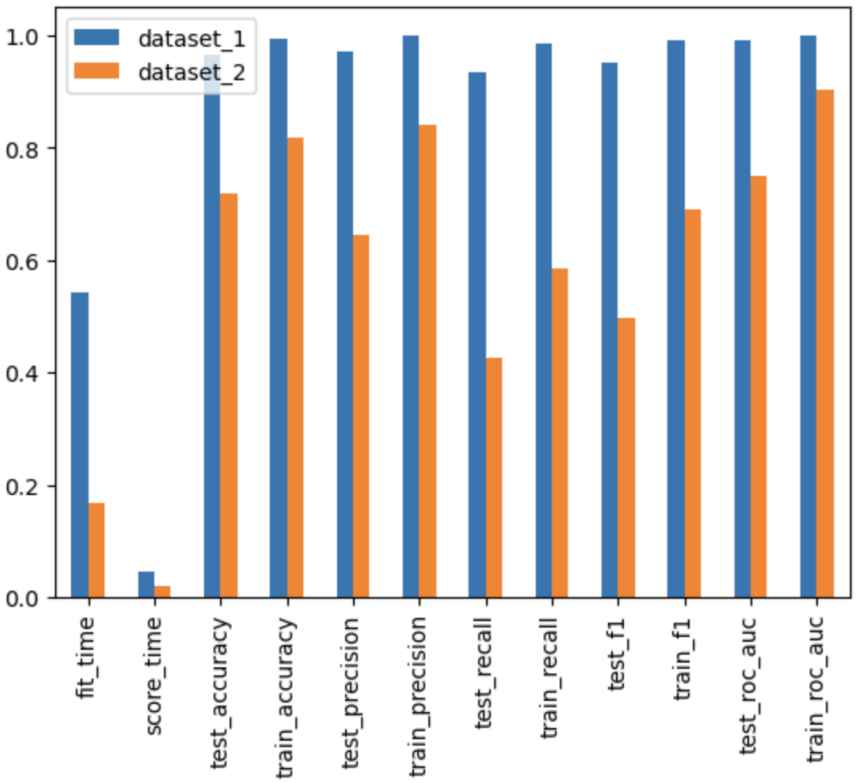


Figure 20. Evaluation scores of tuned models

Similar to previous algorithms, data was preprocessed and fed into the model. Then, we used the GridSearchCV library with 10-fold cross validation to tune the following hyperparameters on each dataset -

1. criterion: gini, entropy
2. max_depth: 4, 5, 6, 7, 8
3. max_features: auto, sqrt, log2
4. n_estimators: 200, 500

The best parameters obtained from hyperparameter tuning were noted and two models were then trained using the best parameters. The accuracy metrics for the two optimal models are shown above.

For the first dataset, the optimal model has the following accuracy metrics-

Metric Name	Training Result	Test Result
Precision	1	0.95
Accuracy	0.99	0.96
Recall	0.984	0.954
F1	0.99	0.954
ROC	0.99	0.99

And for the second dataset, the optimal model has the following accuracy metrics-

Metric Name	Training Result	Test Result
Precision	0.83	0.65
Accuracy	0.82	0.74

Recall	0.584	0.425
F1	0.69	0.5
ROC	0.90	0.75

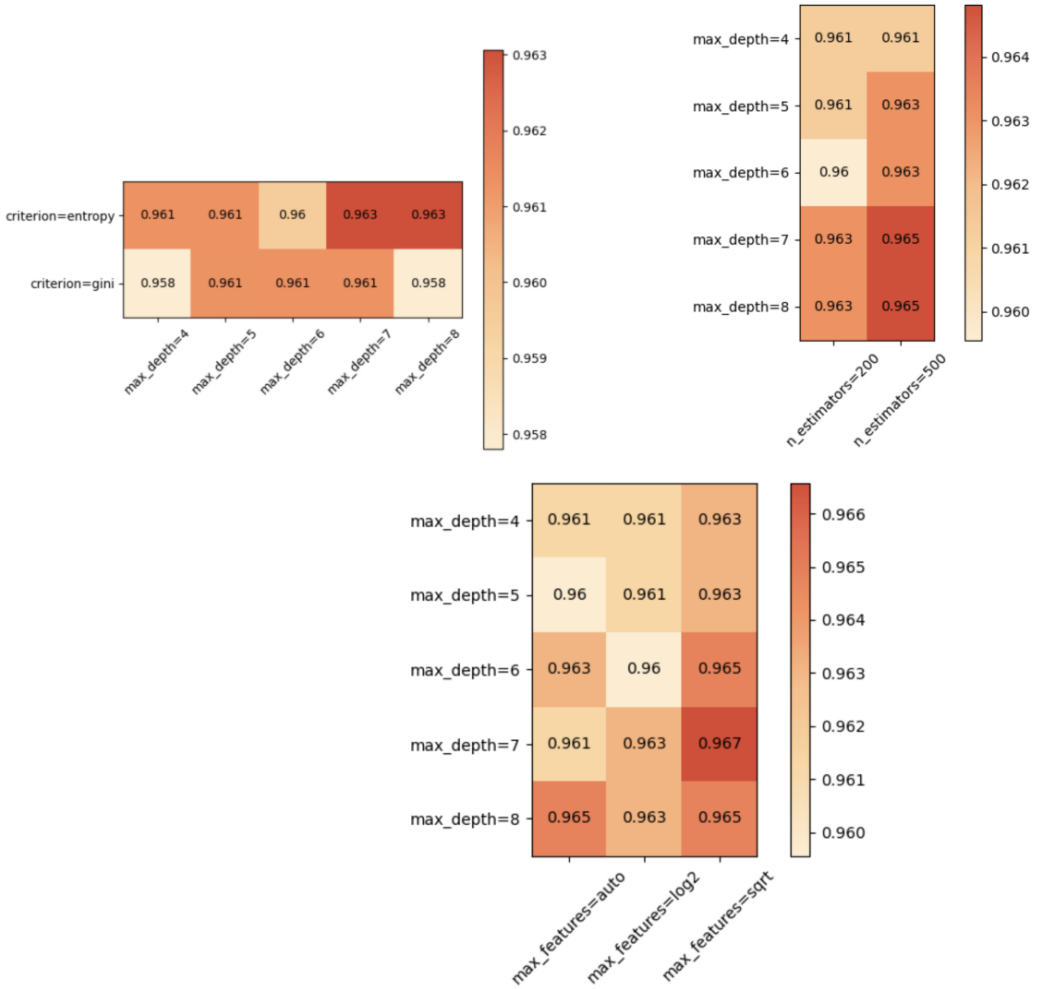


Figure 21. Dataset 1 Hyperparameter Tuning Results

As seen in the figure above, in the case of dataset 1, the highest accuracy of 0.965 was obtained for the following parameters-

1. criterion: entropy
2. max_depth: 6
3. max_features: sqrt
4. n_estimators: 500

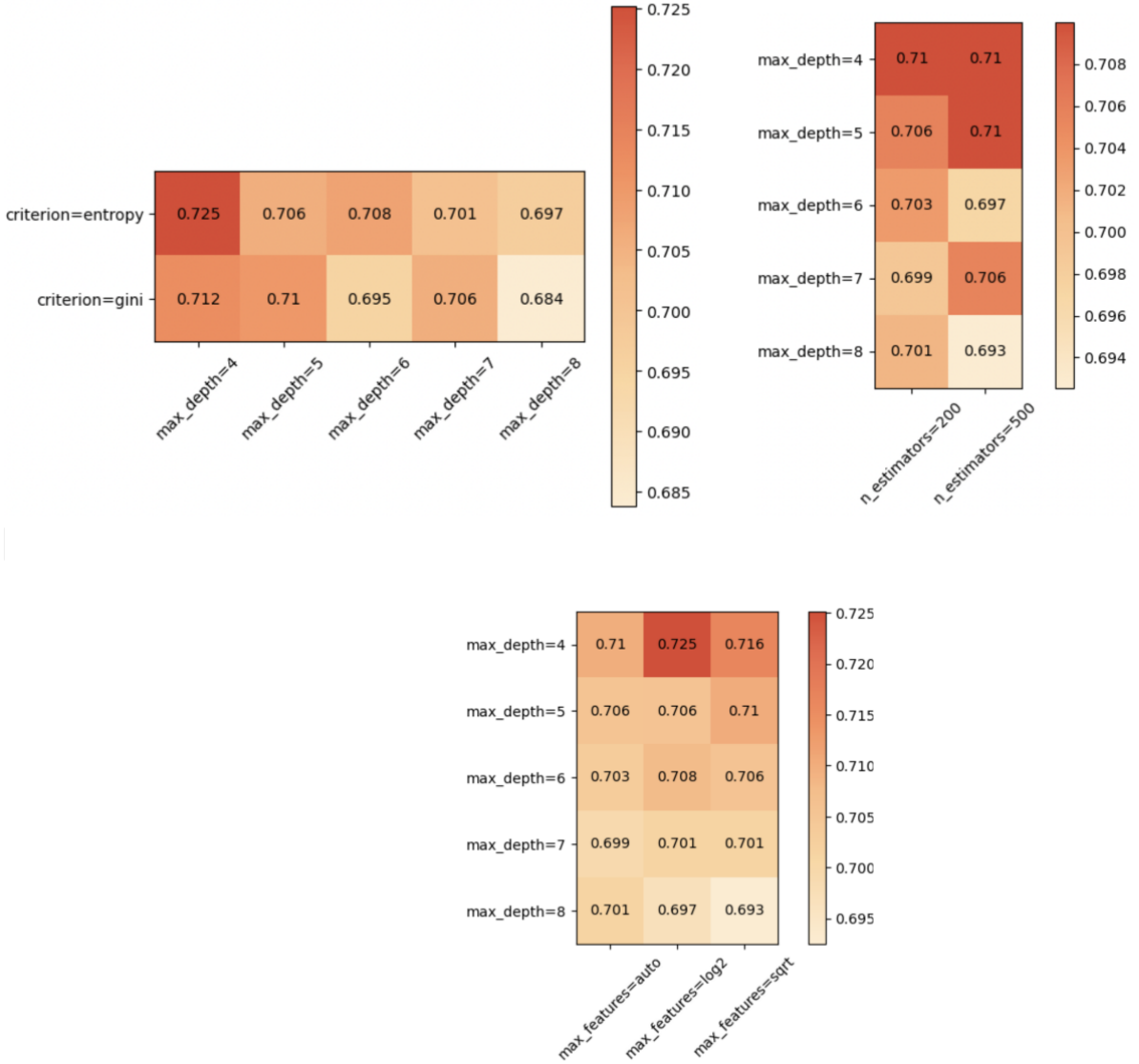


Figure 22. Dataset 2 Hyperparameter Tuning Results

And for dataset 2, the highest accuracy of 0.725 was obtained for the following parameters-

1. criterion: entropy
2. max_depth: 4
3. max_features: log2
4. n_estimators: 200

6. AdaBoost

AdaBoost is one of the most common boosting algorithms. This machine learning algorithm focuses on improving model performance. Boosting algorithms achieve high performance by training multiple weak models and then combine predictions to predict the output.

AdaBoost algorithm is explained in the figure below. The figure is depicted with three weak learners trained only training parts of the problem. The final prediction aggregates the three weak learner to create the combined model.

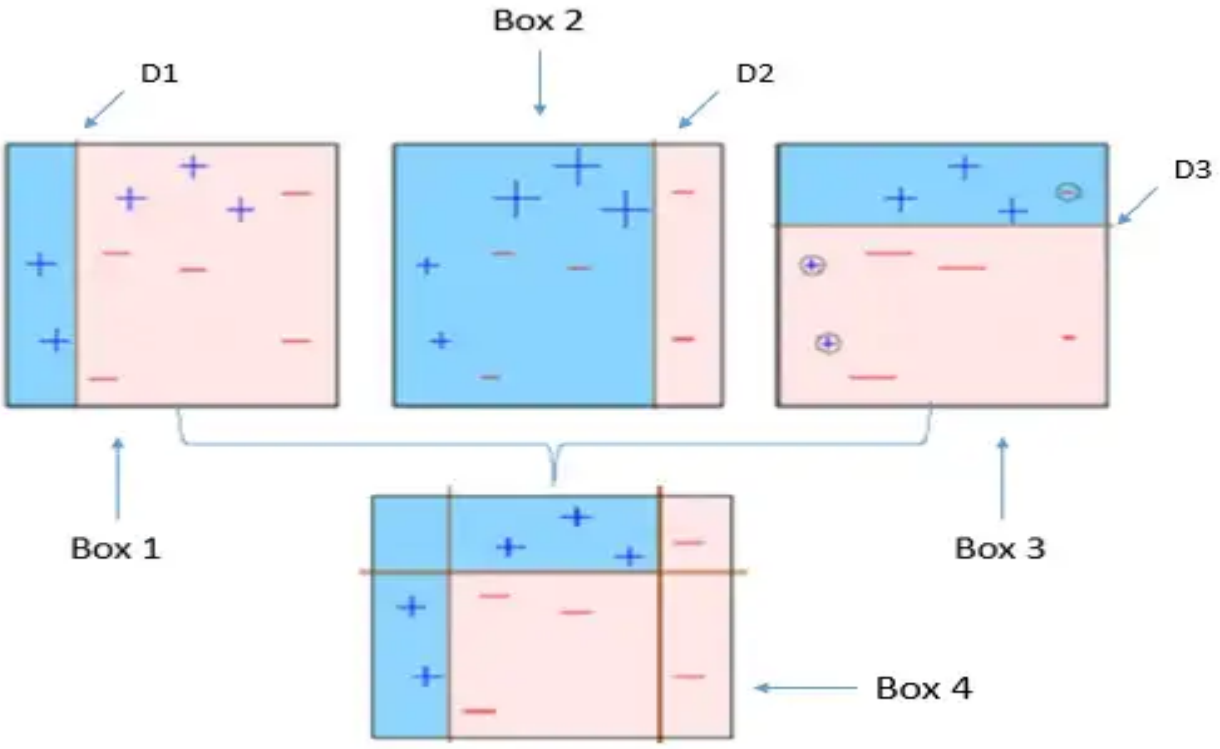


Figure 23. AdaBoost Architecture

Below is the AdaBoost algorithm:

1. Set up weights of each data point in the dataset to a constant value.
2. Multiple weak learners are trained, and independent weights are trained.
3. Error and accuracy scores are calculated for each weak learner
4. Update the weights of each data point based on the error rate, with data points that were misclassified given higher weights.
5. Repeat the steps 2-4 until convergence is reached.
6. Weak learners are combined to produce the final model.

AdaBoost can significantly reduce error rates and successfully process complex data. It is optimal for both classification and regression tasks. AdaBoost has its own challenges. It can be sensitive to outliers that can reduce the accuracy of the model. It can also consume compute resources, especially when dealing with large datasets.

Results and Evaluation

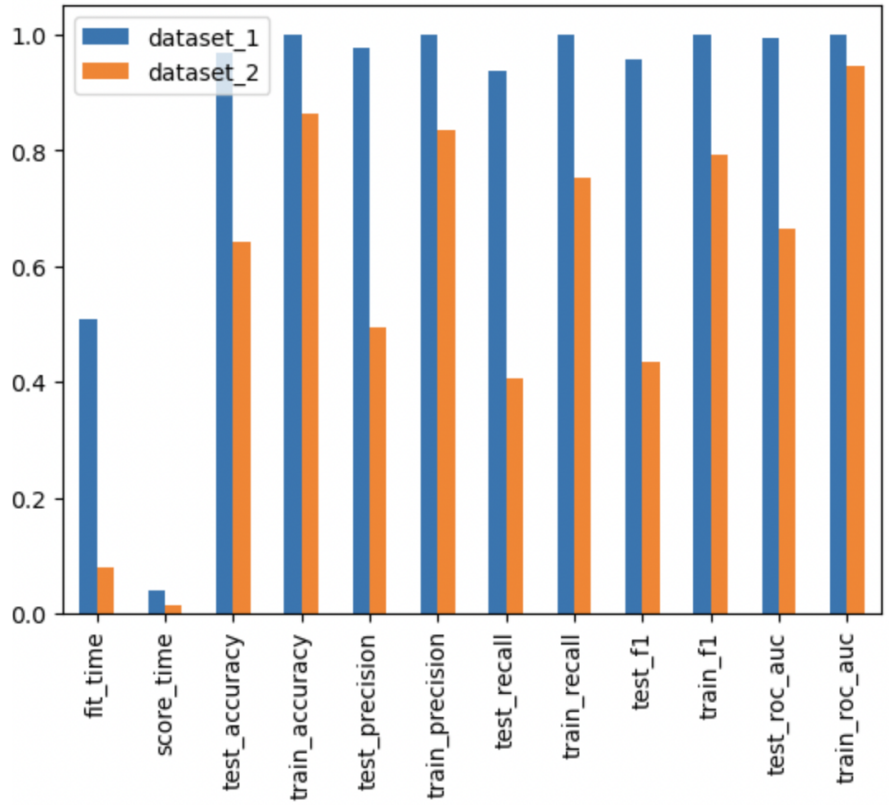


Figure 24. Evaluation Scores of Tuned Models

The Adaboost model was created similar to the other models. The preprocessed data was fed into a 10-fold GridSearchCV model to tune hyperparameters on both datasets. Three values of n_estimators (100, 200, 300) were used for the hyperparameter tuning.

The best parameters obtained from hyperparameter tuning were noted and two models were then trained using the best parameters. The accuracy metrics for the two optimal models are shown above.

For the first dataset, the optimal model has the following accuracy metrics-

Metric Name	Training Result	Test Result
Precision	1	0.97
Accuracy	1	0.96
Recall	1	0.938
F1	1	0.956
ROC	1	0.99

And for the second dataset, the optimal model has the following accuracy metrics-

Metric Name	Training Result	Test Result
Precision	0.83	0.492
Accuracy	0.862	0.643
Recall	0.752	0.406

F1	0.791	0.435
ROC	0.945	0.663

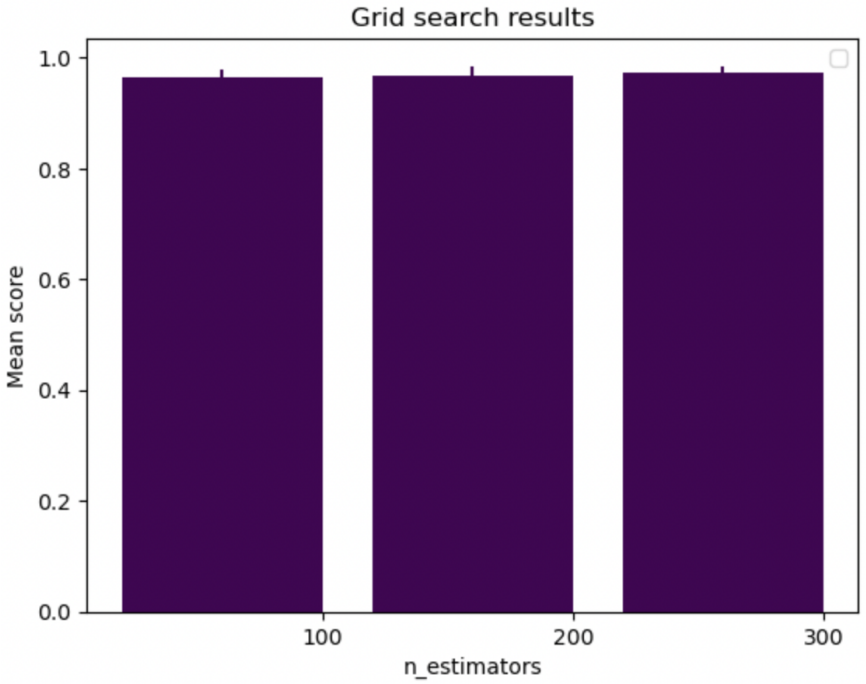


Figure 25. Dataset 1 Hyperparameter Tuning Results

In the first dataset, the highest accuracy scores are achieved for n_estimators is 300

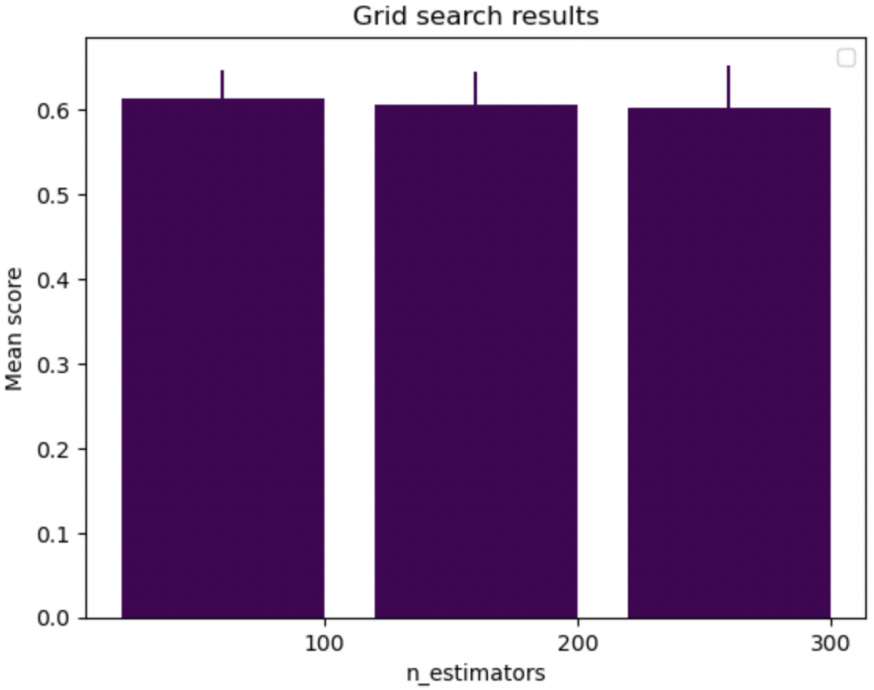


Figure 26. Dataset 2 Hyperparameter Tuning Results

And for the second dataset, the highest accuracy scores are achieved for n_estimators is 100

7. Neural Network on the MNIST dataset

Neural network is statistical learning algorithms mimicking neural networks in biology (particularly the brain in the central nervous system of animals) in machine learning and cognitive science. An neural network refers to all models that have problem-solving abilities by changing synaptic coupling strength through learning by artificial neurons (nodes) that form a network through synaptic coupling. In a narrow sense, it may refer to multilayer perceptrons using error backpropagation.

Neural networks are used when guessing and approximating a function that is generally veiled and depends on many inputs. It is usually represented as an interconnection of neuron systems that computes values from inputs and is adaptable, allowing machine learning such as pattern recognition to be performed.

For example, a neural network for handwritten digits recognition is defined as a set of input neurons, which are activated by pixels of an input image. After function transformations and weights are applied, the activation of that neuron is passed on to other neurons. This process is repeated until the last output neuron is activated, depending on which digit was read.

Like other machine learning methods, neural networks are typically used to solve a wide range of problems, such as computer vision or speech recognition, that are difficult to solve with rule-based programming.

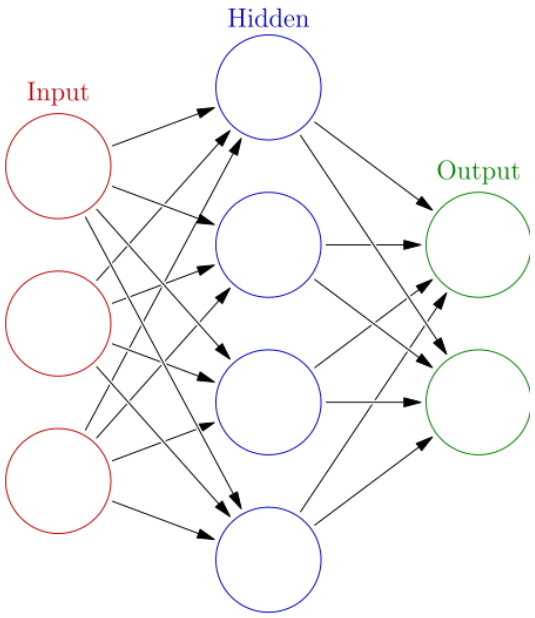


Figure 27. An example of a neural network consisting of one input, one hidden and one output layer.

A typical layer in a neural network is composed of: an input layer that is fed data, a hidden layer that receives values, applies weights, and then applies an activation function to the next layer to pass the resulting value to the next layer, and finally, an output layer that applies a function that determines the result. The depth and accuracy of the model vary according to the number of hidden layers.

The pseudo algorithm of neural networks is as below:
Forward propagation:

1. Fetch the batch size data to input layer
2. Apply weights of next hidden layer on 1
3. Apply activation function on the result of 2
4. Pass to next hidden layer until the last output layer
5. In the output layer, apply a function that can make decisions on the output value.

Backward propagation:

6. Computes the error between the answers and decisions made by the model
7. Calculate the gradient and apply on previous hidden layer to update the weights
8. Repeat 7 until reaches to input layer

→ Repeat the forward and backward propagation until the model reaches a high accuracy on decision.

In this case, the number of hidden layers, an activation function, a learning optimizer method, and a weight initialization method etc. may vary according to the type of data or classification of the model to be trained.

Implementation

In this project, to implement a neural network on MNIST dataset we used tensorflow keras module. The following section describes the code briefly.

```
from tensorflow import keras
from tensorflow.keras import layers, models, initializers
model = models.Sequential()

model.add(layers.Dense(25, activation='sigmoid'))
model.add(layers.Dense(25, activation='sigmoid'))
model.add(layers.Dense(10, activation='softmax'))
```

We use two hidden layers with sigmoid activation function and output layer with softmax function.

```
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
history = model.fit(training[0], training[1], validation_data=(test[0], test[1]),
epochs=15)
```

To train the model for fitting, we use adam optimizer and use categorical cross entropy function for calculating the loss.

Result and Evaluation

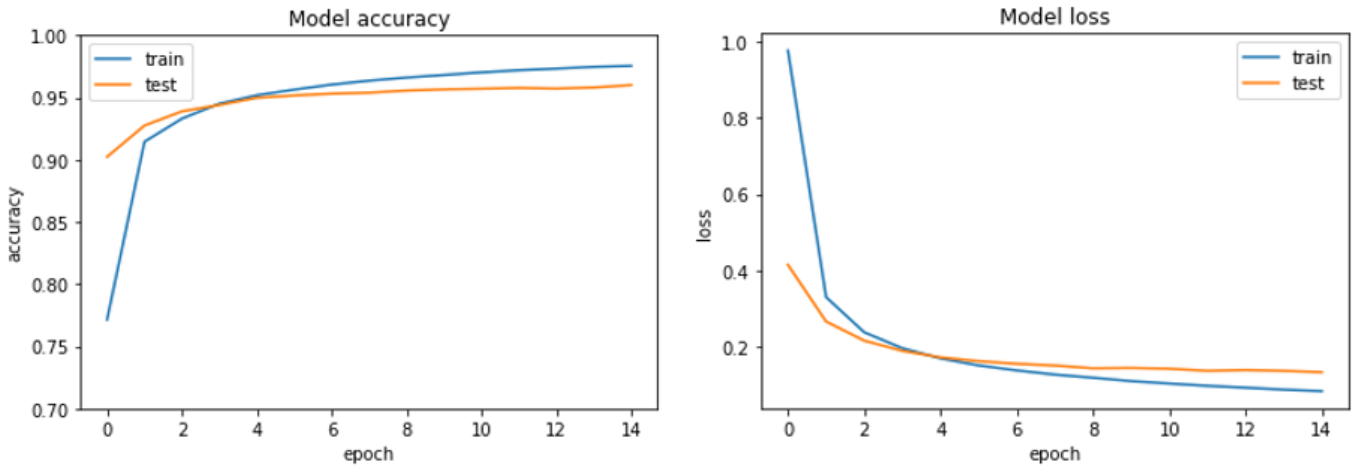


Figure 28. Model accuracy and loss on default function and parameter setting

When learning with the basic settings described above, the same result as Figure 28 was obtained. The training was completed with a final train accuracy of 97.5% and a test accuracy of 96%.

Hyper parameter

- Different number of hidden units

# of hidden layers	Hidden units	Weight initialization	Bias initialization	Learning rate	Train Accuracy	Test Accuracy
2	(25, 25)	Xavier uniform	zeros	0.001	97.5	96.0
2	(50, 50)	Xavier uniform	zeros	0.001	99.0	96.9
2	(100, 100)	Xavier uniform	zeros	0.001	99.7	97.7
2	(256, 256)	Xavier uniform	zeros	0.001	99.7	97.5
2	(512, 512)	Xavier uniform	zeros	0.001	99.7	98.1

Table 1. The train and test accuracy on MNIST dataset varying hidden units respectively

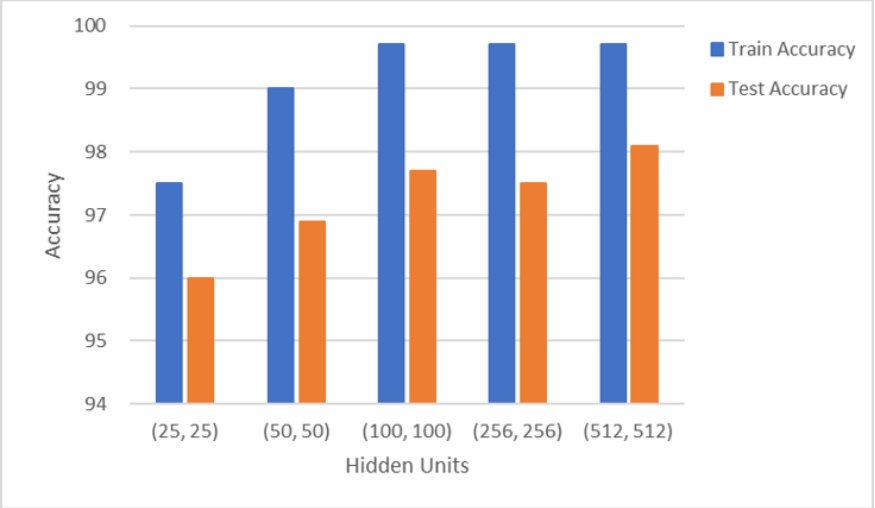


Figure 29. The train and test accuracy on MNIST dataset varying hidden units respectively

The model tended to have lower accuracy as the units in the hidden layer were smaller. Conversely, as the size of the model's hidden unit, which is the width, increases, the accuracy tends to increase both in train and test accuracy.

- Different weight initialization

# of hidden layers	Hidden units	Weight initialization	Bias initialization	Learning rate	Train Accuracy	Test Accuracy
2	(25, 25)	Xavier uniform	zeros	0.001	97.5	96.0
2	(25, 25)	Random normal	zeros	0.001	94.6	93.4
2	(25, 25)	Random uniform	zeros	0.001	55.3	55.7
2	(25, 25)	Truncated normal	zeros	0.001	97.1	95.6
2	(25, 25)	zeros	zeros	0.001	52.9	51.9
2	(25, 25)	ones	zeros	0.001	10.8	10.1
2	(25, 25)	Xavier normal	zeros	0.001	97.6	95.8
2	(25, 25)	He normal	zeros	0.001	97.6	96.0

2	(25, 25)	Identity	zeros	0.001	96.0	94.8
2	(25, 25)	Orthogonal	zeros	0.001	97.3	95.8

Table 2. The train and test accuracy on MNIST dataset varying weight initialization respectively

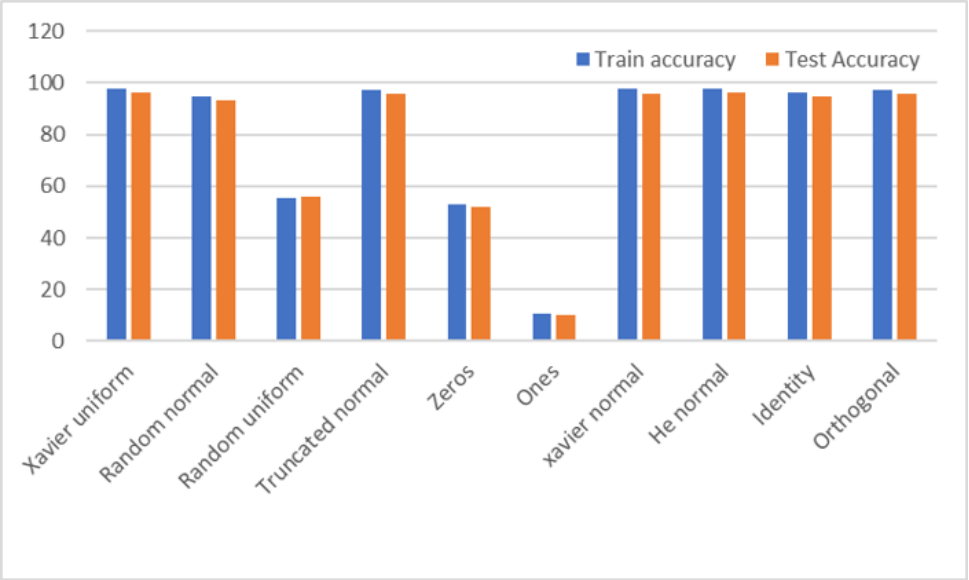


Figure 30. The train and test accuracy on MNIST dataset varying weight initialization respectively

As shown in Figure 30, the most effective weight initialization methods were Xavier uniform and He normal. Next effective methods are, Xavier normal and orthogonal appeared effective. On the other hand, the least accurate weight initialization methods were ones and zeros initialization. It is because the initial weights are extremely biased, so the weights do not properly find the direction to reduce the loss in an optimized way. We can know that the bias of the model is very far(big) from the beginning.

Therefore, we can infer from the experiment that the weight initialization method also has a great impact on learning the model.

- Different bias initialization

# of hidden layers	Hidden units	Weight initialization	Bias initialization	Learning rate	Train Accuracy	Test Accuracy
2	(25, 25)	Xavier uniform	zeros	0.001	97.5	96.0
2	(25, 25)	Xavier uniform	ones	0.001	97.5	95.9
2	(25, 25)	Xavier uniform	Random normal	0.001	97.5	95.9
2	(25, 25)	Xavier uniform	Truncated normal	0.001	97.3	95.8

2	(25, 25)	Xavier uniform	Random uniform	0.001	97.4	96.0

Table 3. The train and test accuracy on MNIST dataset varying bias initialization respectively

Figure 31. The train and test accuracy on MNIST dataset varying bias initialization respectively

In the case of the bias initialization method, it shows a stable high accuracy compared to the weight as shown in Figure 31. Even the zeros and ones method, which showed poor performance in weight initialization, shows good accuracy in bias initialization.

Therefore, we can know that the bias initialization method has little effect on learning from this experiment.

- Different learning rate

# of hidden layers	Hidden units	Weight initialization	Bias initialization	Learning rate	Train Accuracy	Test Accuracy
2	(25, 25)	Xavier uniform	zeros	0.001	97.5	96.0
2	(25, 25)	Xavier uniform	zeros	0.01	96.6	95.1
2	(25, 25)	Xavier uniform	zeros	0.1	52.9	83.0
2	(25, 25)	Xavier uniform	zeros	1	10.0	10.3
2	(25, 25)	Xavier uniform	zeros	0.0001	91.3	91.8

Table 4. The train and test accuracy on MNIST dataset varying learning rate respectively

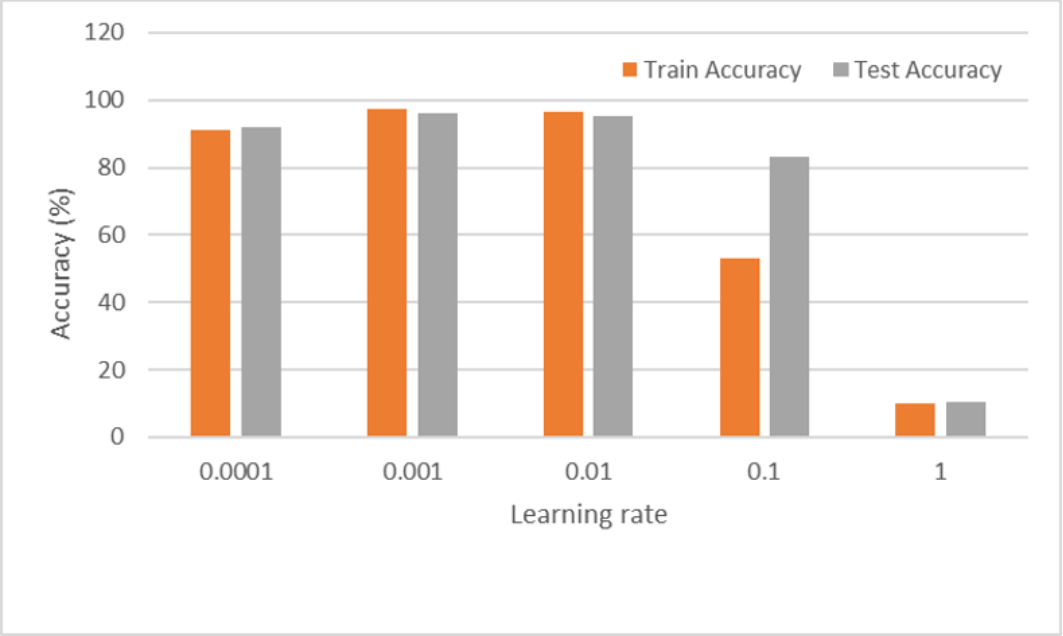
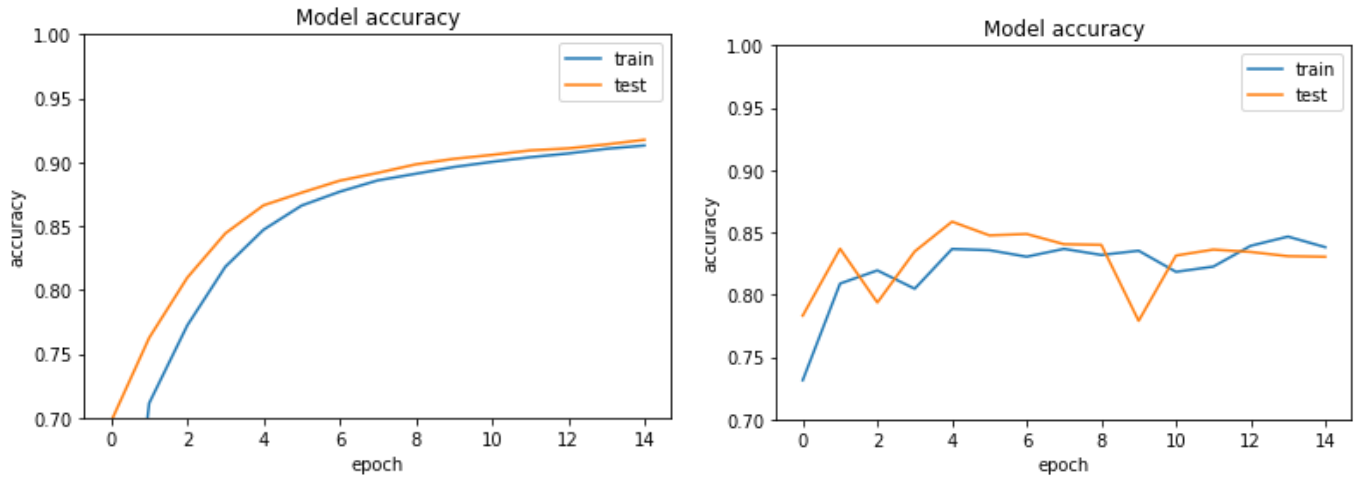


Table 32. The train and test accuracy on MNIST dataset varying learning rate respectively

In this experiment, the results of learning the model with different learning rates were observed. As a result, convergence was slow even when the learning rate was too small, so the best accuracy was not obtained. Conversely, even if the learning rate is too large, problems such as over-correction of the error during the convergence process tend to lower the accuracy. Therefore, choosing an appropriately sized learning rate helps train the model.



(a) Learning rate = 0.0001 (b) Learning rate = 0.1

Figure 33. Accuracy on different learning rate

As shown in Figure 33, the accuracy of the learning model slowly increases compared to the default learning rate in Figure 28 (a). In contrast, the model has problems with learning to converge the accuracy when the learning rate is too high.

- Best accuracy model

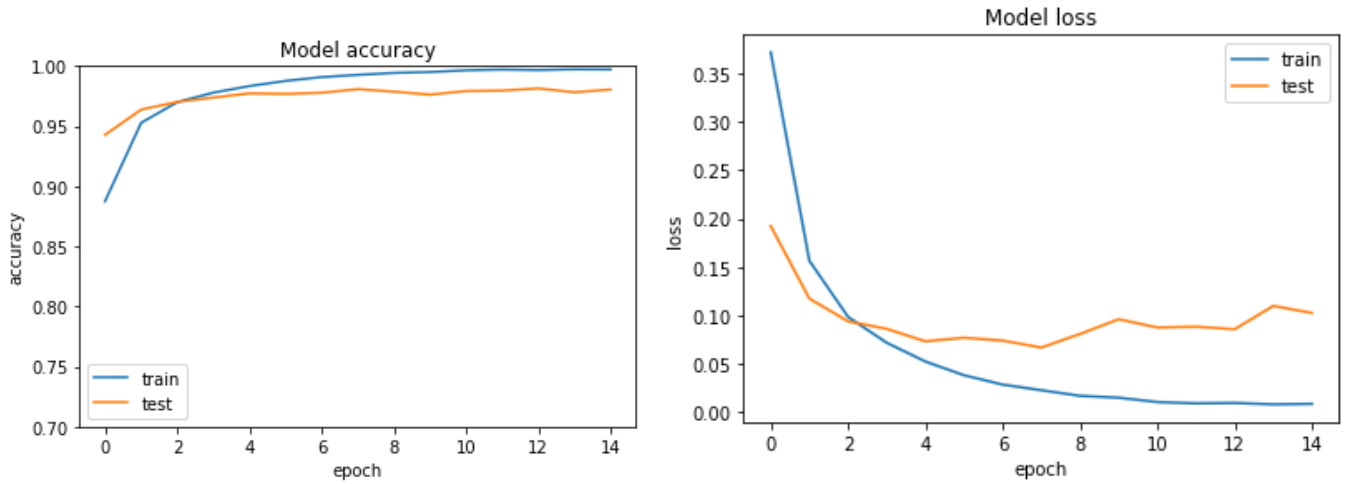


Figure 34. Effective learning model on best parameters

We finally select the final parameter as the number of hidden units is (512, 512), weight initialization is Xavier uniform, bias initialization is Zeros, and learning rate is 0.001. The reason we choose is that it shows the best accuracy. It is because the bigger hidden units, weight initialization methods and appropriate learning rate have big impacts on the learning model. So, we choose the most effective methods among each experiment. The model has the final accuracy of 99.7 on training accuracy 98.1 on test accuracy.