CS 551 Introduction to AI

Homework 03

Arash Mehrabi (S027783)

1 Pre-Processing

For pre-processing, both Standard Scaler and Min Max Scaler were used. The standard scaler standardizes the columns by removing the mean of the column and scaling it to unit variance. The standard score of a sample **x** is calculated as: $z=\frac{x-m}{s}$ where **m** is the mean of the column and **s** is its the standard deviation. It is believed that most machine learning algorithms work better if the data columns are standardized. Meaning, all have a distribution with mean 0 and standard deviation 1. The reason behind this is maybe one column usually has higher values (with mean 100 for example); so, it can affect the machine learning algorithm more compared to the column which usually gets lesser values (with mean 0 for example).

It is believed that when using the data set of RGB values where each value has a specific range (in this case, [0, 255]), the min max scaler works better than standard scaler. By using this pre-processing method, the data set will transform in the range of [0, 1]. The method works as below:

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

2 K-Nearest Neighbors Algorithm

For the first algorithm, K-Nearest Neighbors (KNN) algorithm was used. In KNN, the model saves the whole training data set. At the test time, when classifying a new data point, it searches the training data set for Ks most similar data points to the new data point which we are interested in classifying it. Then, it considers the classes of the Ks most similar points (nearest neighbors) and assigns the class of the majority of the Ks data points to the new data point.

2.1 Hyper-parameters

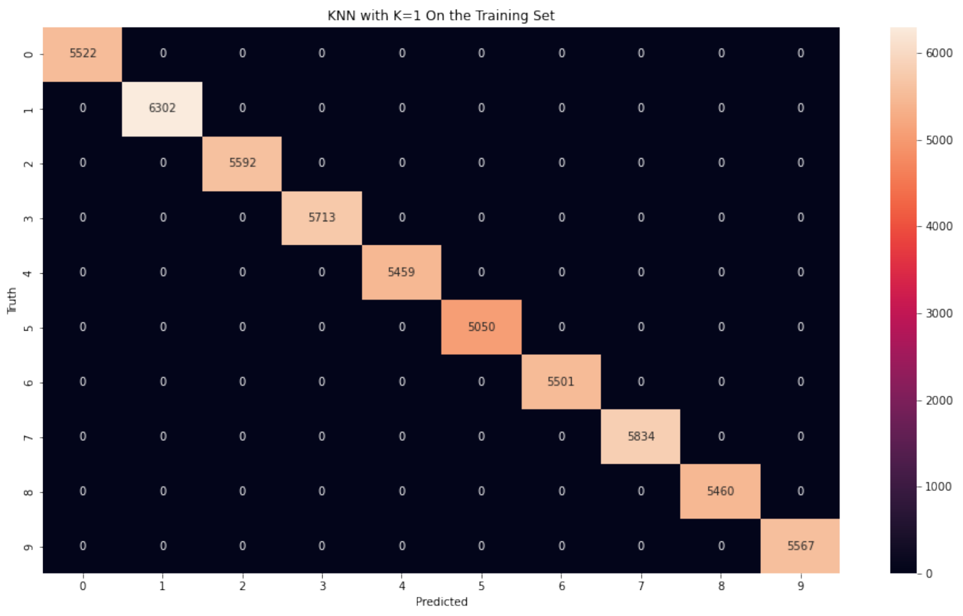
The most important hyper-parameter of this algorithm is K, meaning how many neighbors of each data point should we consider when assigning the classes to the test data set. I tried the K = [1, 3, 7] and I found that while K increases, the performance of the algorithm worsens. However, for all Ks, the algorithm performed well. The accuracy of the algorithm on the test set for each K value in the order of [1, 3, 7] is 0.972, 0.973, and 0.971. So, it can be seen that the algorithm with K=1 performs best considering that it achieved accuracy = 1 on the training set.

Another hyper-parameter is the weights. It can take two values, either ‘distance’ or ‘uniform’. I tried both values; however, I found no difference on the algorithm with k=1.

2.2 Result

This result were produced by KNN algorithm using the best hyper-parameters, meaning when K=1, weights = ‘uniform’, and the pre-processing was min max scaler.

2.2.1 On the training set



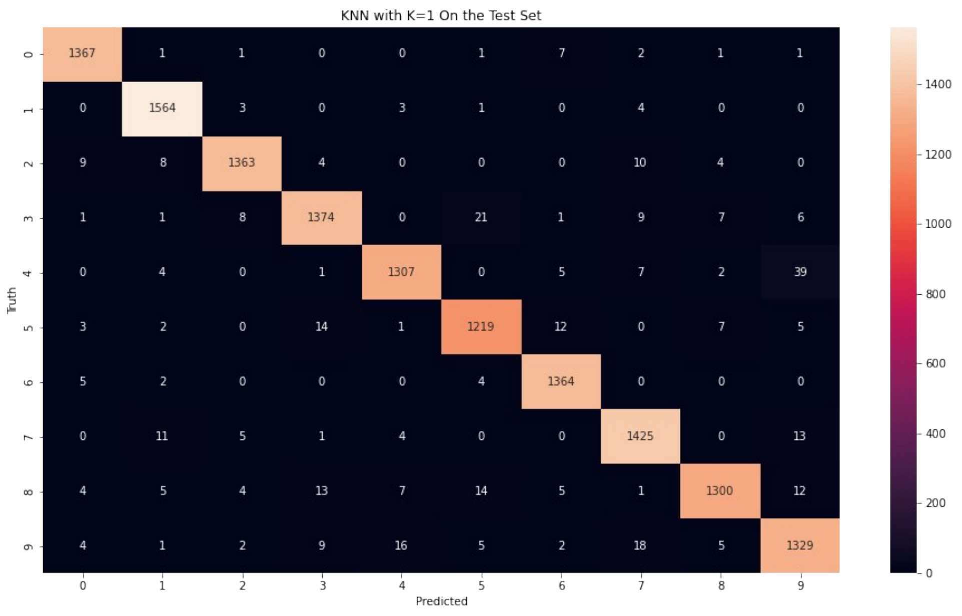
Confusion matrix of a knn algorithm with k=1 on the training set

Label	Precision	Recall	F1	Accuracy
0	1	1	1	100%
1	1	1	1	100%
2	1	1	1	100%
3	1	1	1	100%
4	1	1	1	100%
5	1	1	1	100%
6	1	1	1	100%
7	1	1	1	100%
8	1	1	1	100%
9	1	1	1	100%
Average	1	1	1	100%

Weighted Average	1	1	1	100%

Results of a knn algorithm with k=1 on the training set

2.2.2 On the test set



Confusion matrix of a knn algorithm with k=1 on the test set

Label	Precision	Recall	F1	Accuracy
0	0.98	0.99	0.99	98.98%
1	0.98	0.99	0.99	99.30%
2	0.98	0.97	0.98	97.49%
3	0.97	0.96	0.97	96.21%
4	0.98	0.96	0.97	95.75%
5	0.96	0.97	0.96	96.51%
6	0.98	0.99	0.98	99.2%
7	0.97	0.98	0.97	97.67%
8	0.98	0.95	0.97	95.23%
9	0.95	0.96	0.95	95.54%
Average	0.97	0.97	0.97	97.19%
Weighted Average	0.97	0.97	0.97	97.19%

Results of a knn algorithm with k=1 on the test set

3 Decision Tree Algorithm

Decision tree uses the features in the data set to build a model for classification. At each node of the tree, the algorithm selects the feature that results in the most information gain, meaning less impurity in child nodes.

3.1 Hyper-parameters

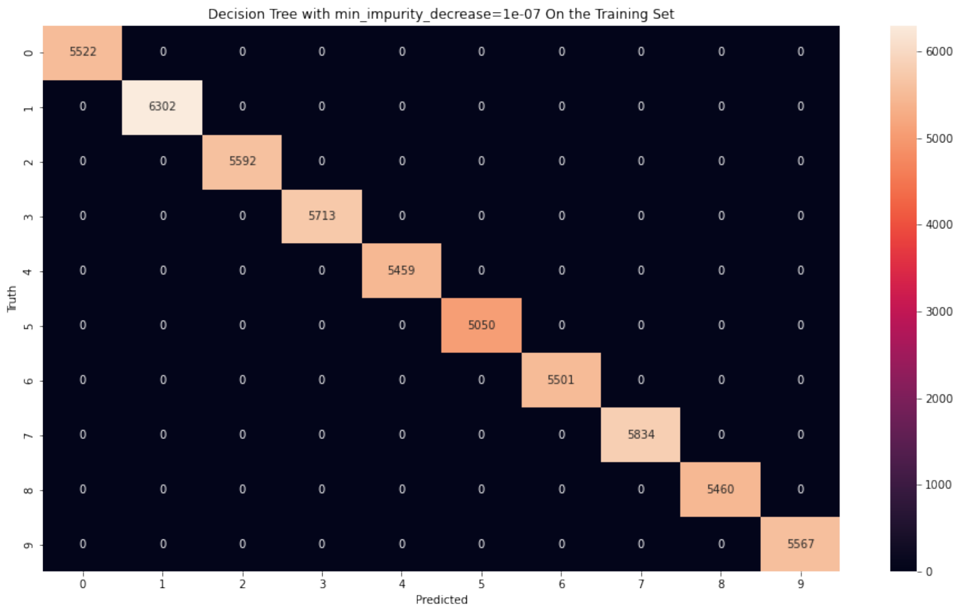
The first hyper-parameter that I tried for this algorithm was min_impurity_decrease which means a node will split if it causes the decrease in the impurity greater than the value that we set. The values [1e-7, 1e-5, 1e-3] were tried for this hyper-parameter. This hyper-parameter prevents model from seeking perfection; hence, avoiding overfitting. For example, if a node has 61 samples from a same class and only 4 from other classes, by tuning this hyper-parameter the model will not split further because it is roughly pure in this condition. For the values [1e-7, 1e-5, 1e-3] for the min_impurity_decrease hyperparameter, we got the total accuracy 0.873, 0.877, and 0.792 on the test dataset, respectively. So, it can be understood that the algorithm works best with the 1e-5 value. Also, the algorithm works better with the min max scaler as the pre-processing step, since we got 0.781, 0.791, and 0.686 respectively when we used the standard scaler.

Another hyper-parameter that tuned was max_depth. As it name suggests, the tree will not expand further if its depth reaches the max_depth value. The values [784, 78, 56] were tried for this hyper-parameter. If we prevent the tree from splitting further, at leaf nodes we may have impurity. However, we can hope that we prevent overfitting. We can see that we get 0.8737, 0.8739, and 0.8743 as accuracy number on the test set. However, they are very similar; so, we can’t deduce which number was better.

3.2 Result

The following is the result of the best hyper-parameter set found, which is min_impurity_decrease=1e-7, and max_depth = None.

3.2.1 On the training set

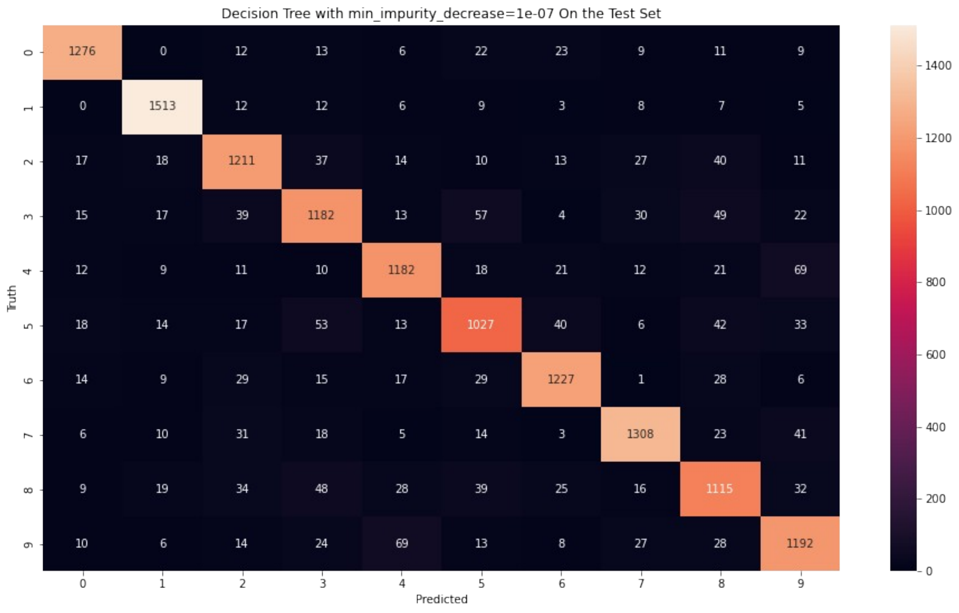


Confusion matrix of a decision tree algorithm with min_impurity_decrease=1e-7 on the training set

Label	Precision	Recall	F1	Accuracy
0	1	1	1	100%
1	1	1	1	100%
2	1	1	1	100%
3	1	1	1	100%
4	1	1	1	100%
5	1	1	1	100%
6	1	1	1	100%
7	1	1	1	100%
8	1	1	1	100%
9	1	1	1	100%
Average	1	1	1	100%
Weighted Average	1	1	1	100%

Results of a decision tree algorithm with min_impurity_decrease=1e-7 on the training set

3.2.2 On the test set



Confusion matrix of a decision tree algorithm with min_impurity_decrease=1e-7 on the test set

Label	Precision	Recall	F1	Accuracy
0	0.92	0.92	0.92	92.4%
1	0.94	0.96	0.95	97.25%
2	0.86	0.85	0.86	85.5%
3	0.84	0.82	0.83	82.5%
4	0.87	0.86	0.86	85.6%
5	0.82	0.82	0.82	82.03%
6	0.89	0.9	0.89	89.52%
7	0.9	0.9	0.9	90.13%
8	0.82	0.82	0.82	82.5%
9	0.84	0.85	0.84	84.9%
Average	0.87	0.87	0.87	87.13%
Weighted Average	0.87	0.87	0.87	87.13%

Results of a decision tree algorithm with min_impurity_decrease=1e-7 on the test set

4 Random Forest Algorithm

The random forest algorithm is made up of many decision trees. The number of decision trees is the most important hyper-parameter in this model. The algorithm makes n random decision trees and classifies the new data point based on votes between those n decision trees.

4.1 Hyper-parameters

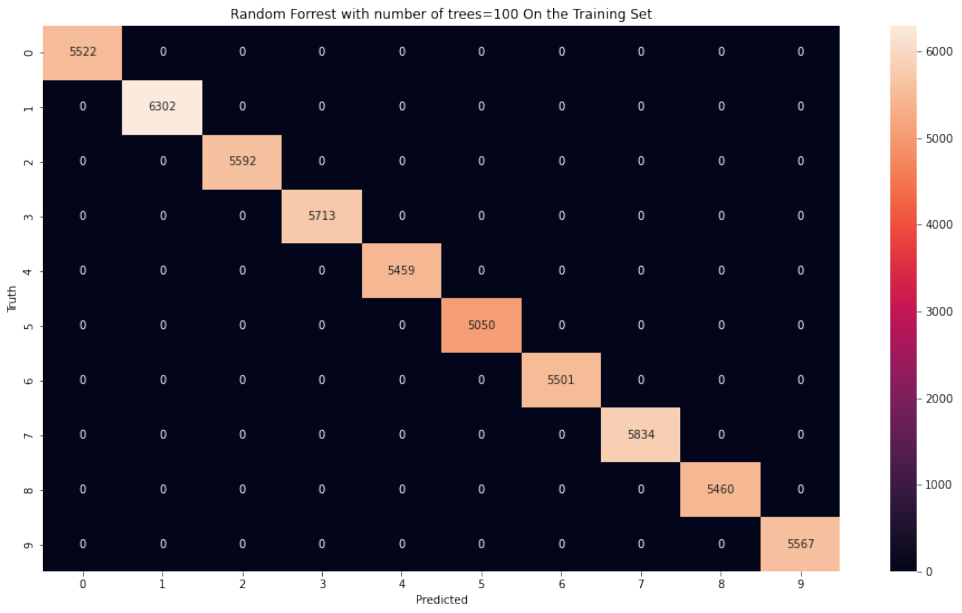
As it was mentioned, the most important hyper-parameter is the number of decision trees (n_estimators). The values [5, 10, 40, 100] were tried for this hyper-parameter. The accuracy 0.92, 0.947, 0.963, 0.967 were achieved respectively on the test set. As expected, with the increase in the number of trees, the performance of the model gets better. Since the other hyper-parameters are similar to the decision tree's, I used the best known hyper-parameters for the rest.

Although, we can see that using the min max scaler performs better in this algorithm too since we got 0.90, 0.93, 0.961, 0.966 respectively for the accuracy when the standard scaler used.

4.2 Result

The following is the result of the algorithm using n_estimator = 100. The max_depth = None which means the trees will expand without any limitation and min_impurity_decrease = 0 meaning the nodes will split until they reach complete purity.

4.2.1 On the training set

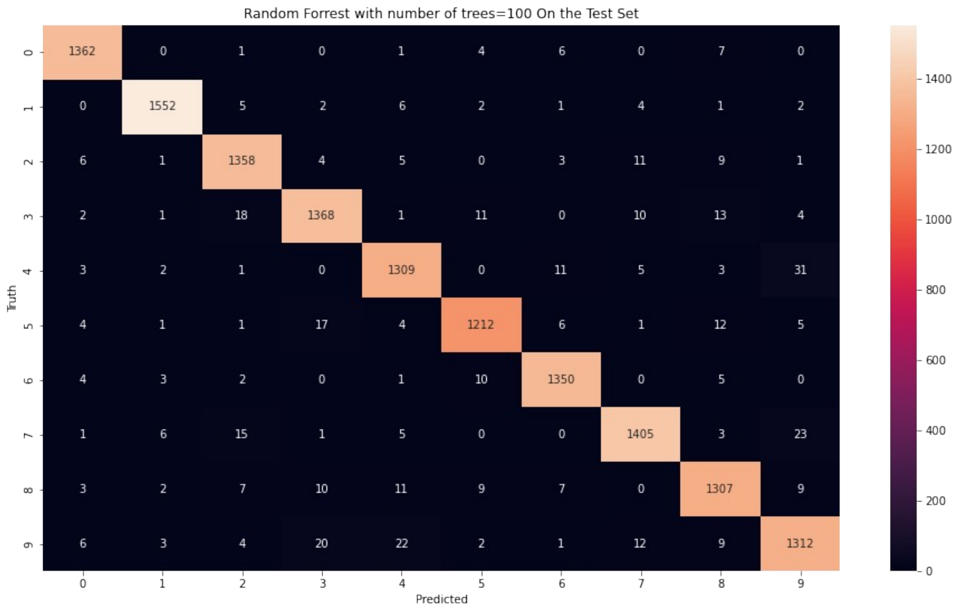


Confusion matrix of a Random Forest algorithm with n_estimators=100 on the training set

Label	Precision	Recall	F1	Accuracy
0	1	1	1	100%
1	1	1	1	100%
2	1	1	1	100%
3	1	1	1	100%
4	1	1	1	100%
5	1	1	1	100%
6	1	1	1	100%
7	1	1	1	100%
8	1	1	1	100%
9	1	1	1	100%
Average	1	1	1	100%
Weighted Average	1	1	1	100%

Results of a Random Forest algorithm with n_estimators=100 on the training set

4.2.2 On the test set



Confusion matrix of a Random Forest algorithm with n_estimators=100 on the test set

Label	Precision	Recall	F1	Accuracy
0	0.97	0.99	0.98	98.84%
1	0.99	0.99	0.99	98.54%
2	0.96	0.97	0.97	97%
3	0.96	0.95	0.96	95.45%
4	0.96	0.97	0.97	96.56%
5	0.97	0.96	0.96	95.72%
6	0.97	0.98	0.98	98.4%
7	0.97	0.96	0.97	96.09%
8	0.96	0.95	0.95	95.16%
9	0.95	0.96	0.96	95.83%
Average	0.97	0.97	0.97	96.76%
Weighted Average	0.97	0.97	0.97	96.76%

Results of a Random Forest algorithm with n_estimators=100 on the test set

5 Conclusion

As it can be seen from the results, the KNN and Random Forest algorithms were performed better comparing to the Decision Tree algorithm. KNN was the best algorithm, then with a slight difference in performance, we have the Random Forest, then with a relatively huge gap there is the decision tree algorithm.

Although, we can conclude that for this data set, the min max scaler obviously performed better comparing to the standard scaler.