Project 2 Report

Intro

In this project we are tasked with implementing a digit recognizer. The project works like a pipeline where we test the data on three different classifiers then we choose the best classifier in the end and shows the results with the best hyperparameters and accuracy score.

We used 3 classifiers:

- K nearest neighbors
- Decision tree
- Random forest

First, we choose K nearest neighbor since this is a simple classifier and one of the first classifier algorithms that we went through in this course.

Second, we choose a decision tree classifier since we have a good understanding of how this works from the first project, and found out from that project that this classifier can be effective.

Third I wanted to choose a neural network but decided against it due to both time and necessity, this problem does not necessarily need a neural network to be solved. So, I chose another classifier, random forest classifier that is known to handle large datasets efficiently.

In the project we have used the sklearn library for the classifiers. To figure out the best hyperparameters to get the optimal performance out of each of our classifiers we created multiple classifiers with different parameters and ranked them on their accuracy.

Time

It is worth noting that when running the entire jupyter notebook we run trough all the classifiers with all the different parameters, this makes it so that it takes a long time to run trough all the code, on my computer it took 1589 seconds which is about 27 minutes. A solution for this would probably be to preprocess the images in some way. Later in the report we discuss this and how the image preprocessing did not work for us. The Random Forest classifier is the one that takes the most time, given the hyperparameters.

Data and preprocessing

Our data consists of 85237 samples that are 24x24 images of handwritten digits. Each digit has a corresponding label 0-9 means digit 0-9 and 10 means there is no digit in the image. Each pixel in the image can range from 0-255 so we normalize to get the range 0-1 by dividing each pixel on 255. Our value between 0-255 is the intensity of the color where 0 is white, 255 black and numbers in between are transition coloring.

K Nearest Neighbors

K nearest neighbors shortened to KNN. Is one of the fundamental algorithms in machine learning, it is also one of the simplest forms of machine learning algorithms, although this does not mean it is not effective. Does not perform any training when we supply it with training data.

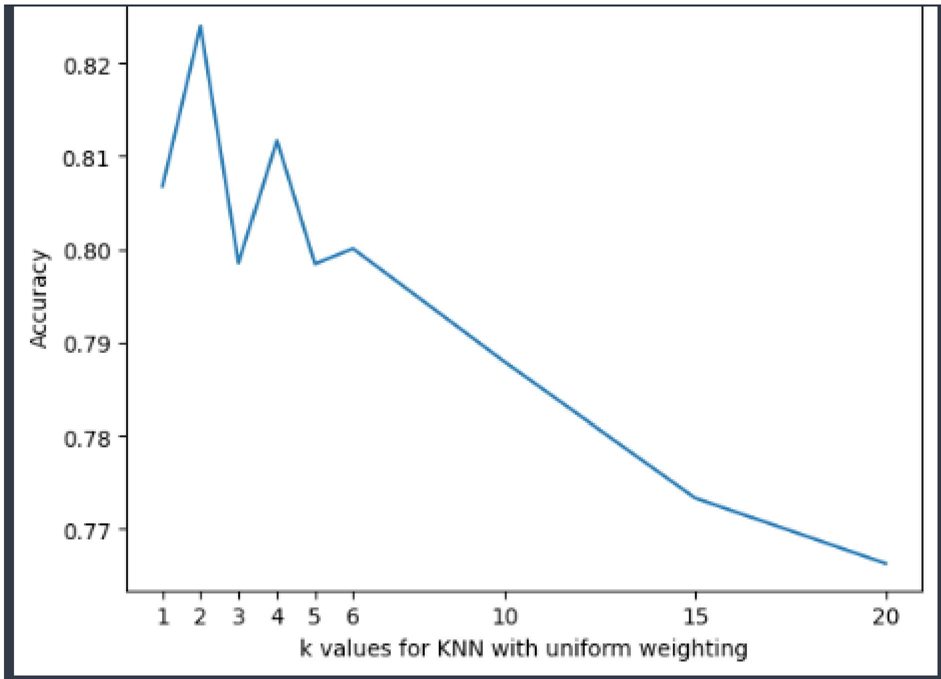
Advantages: Fast, no training period, easy to implement

Disadvantage: requires a lot of memory, sensitive to noisy data, does not work optimal with large datasets.

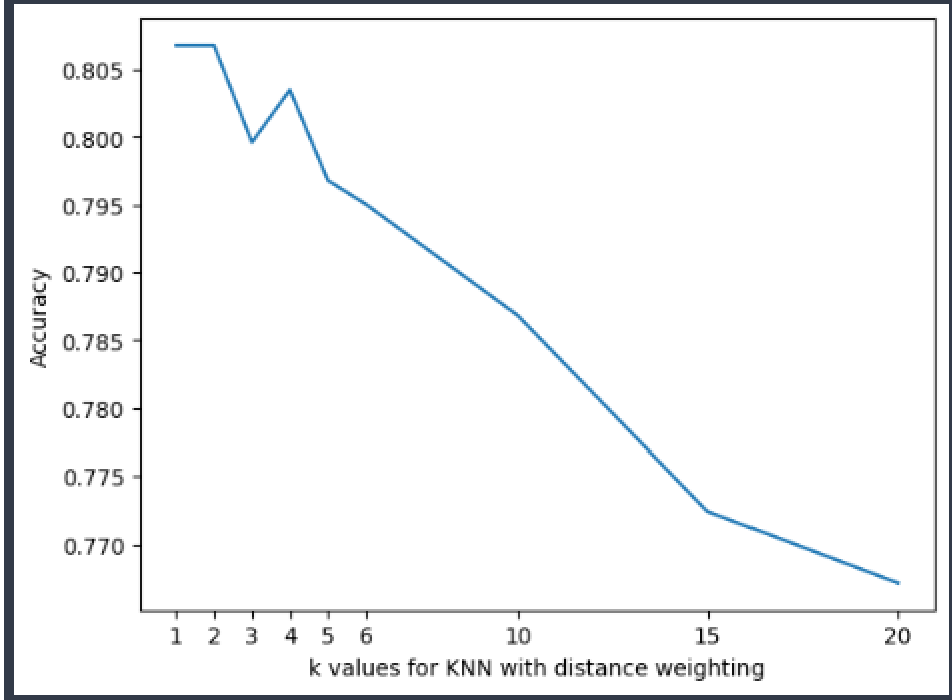
Hyperparameters:

- N_neighbors, number of neighbors to use tried 1, 2, 3, 4, 5, 6, 10, 15 and 20
- Weights wight function used in the prediction we used uniform and distance. Uniform is when all points in each neighborhood are weighted equally. Distance weigh points by the inverse of their distance.

We found that the best knn model is a model with uniform weighting and number of neighbors = 2. This gave us a accuracy of 82.4% and a running time of 0.03 seconds. Here are our results for KNN after going through our hyperparameters form the jupyter notebook:

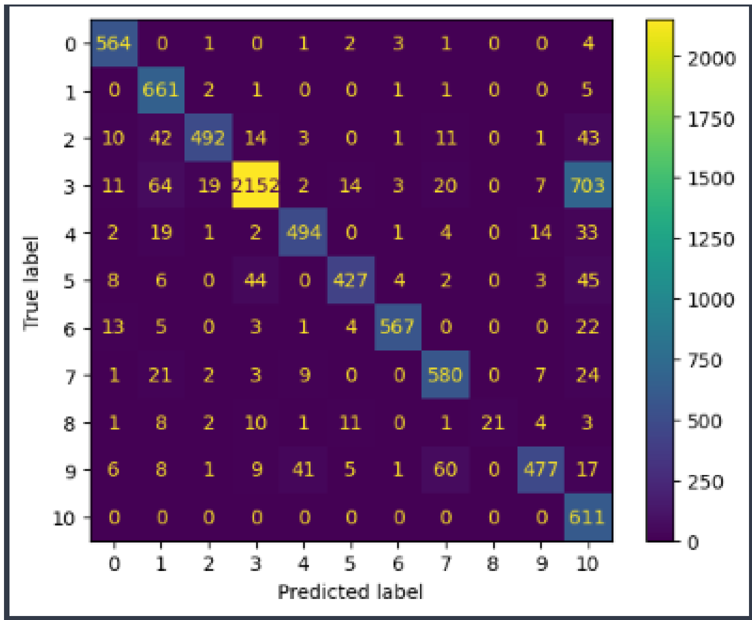


```
The best score with uniform weighting is  82.4 % with number of k= 2
```



```
The best score with distance weighting is  80.68 % with number of k = 1
The best knn model is a model with weighting:  uniform and with number of neighbors:  2
the best model give us an accuracy of  82.4
time to run the best knn classifier with best hyperparameters:  0.03125 seconds
```

<u>Confusion matrix for best knn model</u>



True label \ Predicted label	0	1	2	3	4	5	6	7	8	9	10
0	564	0	1	0	1	2	3	1	0	0	4
1	0	661	2	1	0	0	1	1	0	0	5
2	10	42	492	14	3	0	1	11	0	1	43
3	11	64	19	2152	2	14	3	20	0	7	703
4	2	19	1	2	494	0	1	4	0	14	33
5	8	6	0	44	0	427	4	2	0	3	45
6	13	5	0	3	1	4	567	0	0	0	22
7	1	21	2	3	9	0	0	580	0	7	24
8	1	8	2	10	1	11	0	1	21	4	3
9	6	8	1	9	41	5	1	60	0	477	17
10	0	0	0	0	0	0	0	0	0	0	611

We can see that we have alot of mistakes on the number 3, this is probably because the number of 3 we have in the data set is much bigger than other numbers. We have a total occurence of 30655 3s in the entire data set compared to the other digits that have around five to six thousand occurances. This means that we get a higher numner of mistakes on the digit '3' compared to other digits. We note this down and continue with the other classifiers to see if they can solve the problem. One reason for this can be that knn is a non training algorithm so it does not train itself any more on a digit that occurs often than the others

Decision Tree classifier

Is a flowchart like classifier that shows a clear pathway to a decision. We start at a single node which then branches in two or more directions. Decision trees can give good accuracy if trained correctly. Unlike KNN, a decision tree can be generated from training sets.

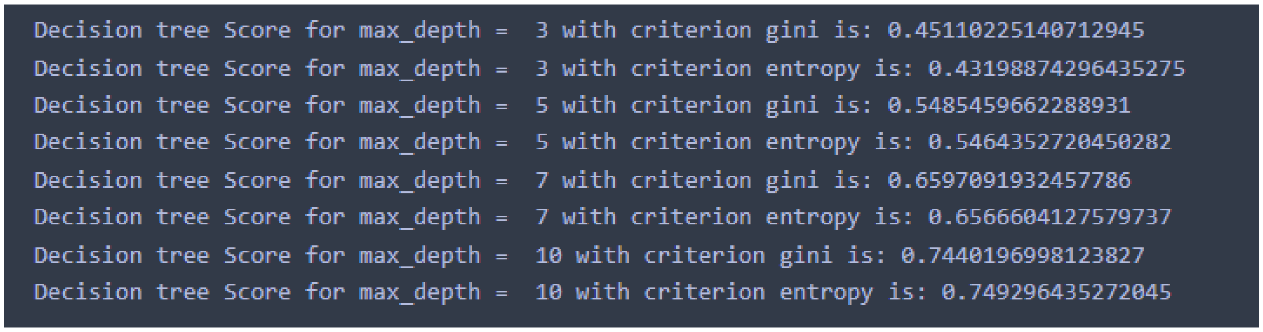
<u>Hyperparameters:</u>

- Max_depth, the maximum depth of our tree, we used the depths, [10, 13, 15, 16, 17, 18, 19, 20]
- Criterion the function to measure the quality of a split. We used entropy and gini.

Advantages: Easy to understand, handles missing values well, can handle any type of data

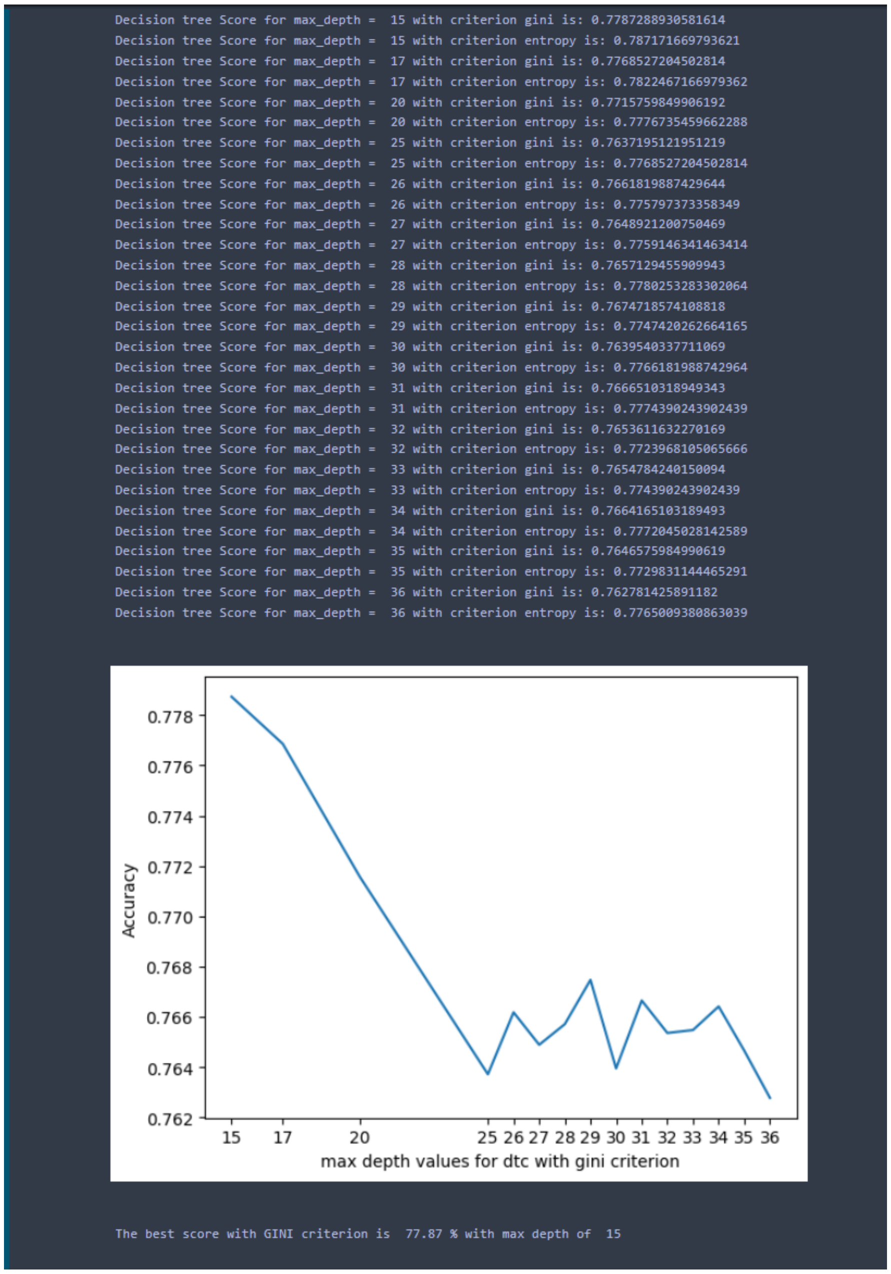
Disadvantages: High time to train, training is expensive, small change in data can cause big change in structure

We test on the different parameters of gini or entropy and max depth. I first tried depths [3,5,7,10] but got very bad accuracy on max depth bellow 10. As seen bellow

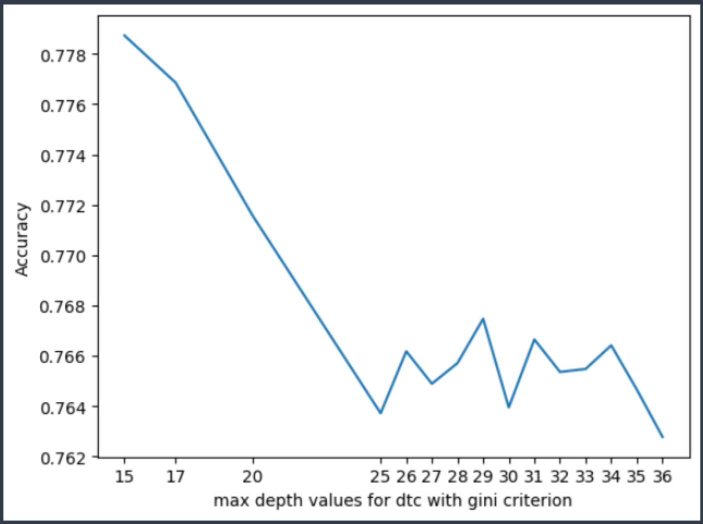


```
Decision tree Score for max_depth =  3 with criterion gini is: 0.45110225140712945
Decision tree Score for max_depth =  3 with criterion entropy is: 0.43198874296435275
Decision tree Score for max_depth =  5 with criterion gini is: 0.5485459662288931
Decision tree Score for max_depth =  5 with criterion entropy is: 0.5464352720450282
Decision tree Score for max_depth =  7 with criterion gini is: 0.6597091932457786
Decision tree Score for max_depth =  7 with criterion entropy is: 0.6566604127579737
Decision tree Score for max_depth =  10 with criterion gini is: 0.7440196998123827
Decision tree Score for max_depth =  10 with criterion entropy is: 0.749296435272045
```

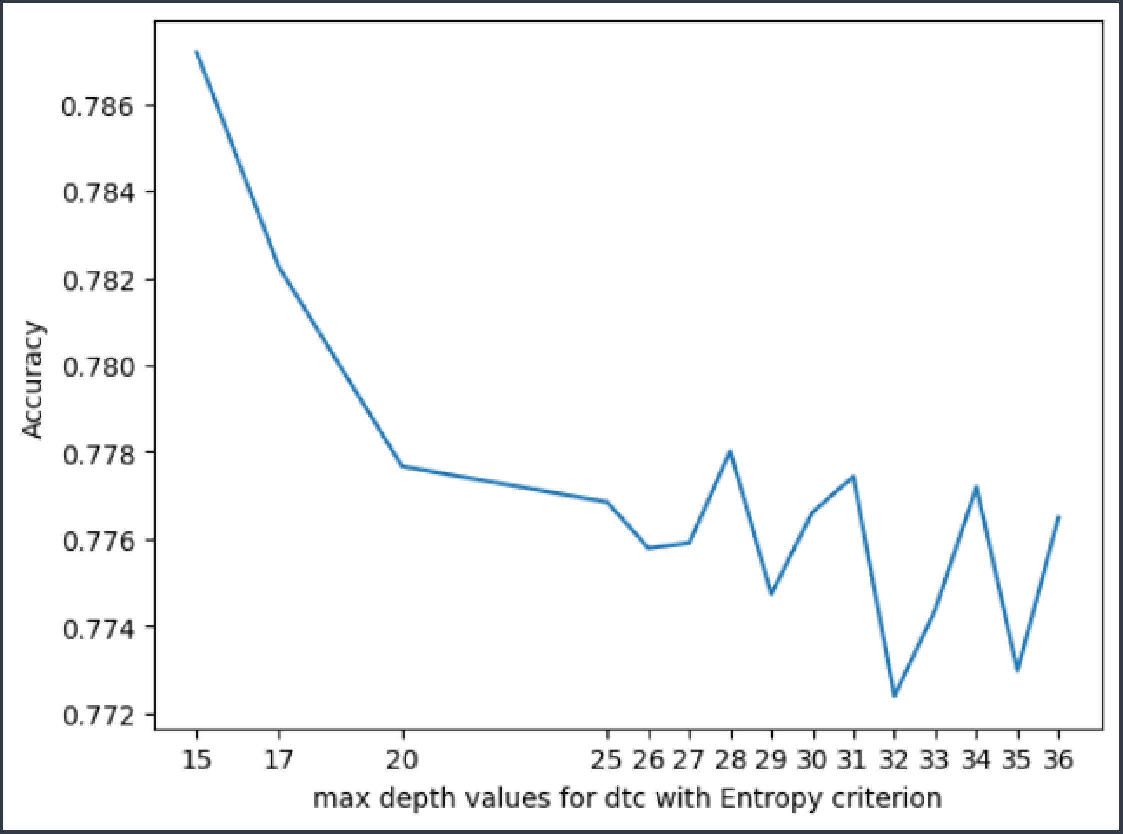
So, we changed the depths to [15, 17, 20, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36] to see how the classifier worked on higher depths after 25. This was the results.



```
Decision tree Score for max_depth =  15 with criterion gini is: 0.7787288930581614
Decision tree Score for max_depth =  15 with criterion entropy is: 0.787171669793621
Decision tree Score for max_depth =  17 with criterion gini is: 0.7768527204502814
Decision tree Score for max_depth =  17 with criterion entropy is: 0.7822467166979362
Decision tree Score for max_depth =  20 with criterion gini is: 0.7715759849906192
Decision tree Score for max_depth =  20 with criterion entropy is: 0.7776735459662288
Decision tree Score for max_depth =  25 with criterion gini is: 0.7637195121951219
Decision tree Score for max_depth =  25 with criterion entropy is: 0.7768527204502814
Decision tree Score for max_depth =  26 with criterion gini is: 0.7661819887429644
Decision tree Score for max_depth =  26 with criterion entropy is: 0.775797373358349
Decision tree Score for max_depth =  27 with criterion gini is: 0.7648921200750469
Decision tree Score for max_depth =  27 with criterion entropy is: 0.7759146341463414
Decision tree Score for max_depth =  28 with criterion gini is: 0.7657129455909943
Decision tree Score for max_depth =  28 with criterion entropy is: 0.7780253283302064
Decision tree Score for max_depth =  29 with criterion gini is: 0.7674718574108818
Decision tree Score for max_depth =  29 with criterion entropy is: 0.7747420262664165
Decision tree Score for max_depth =  30 with criterion gini is: 0.7639540337711069
Decision tree Score for max_depth =  30 with criterion entropy is: 0.7766181988742964
Decision tree Score for max_depth =  31 with criterion gini is: 0.7666510318949343
Decision tree Score for max_depth =  31 with criterion entropy is: 0.7774390243902439
Decision tree Score for max_depth =  32 with criterion gini is: 0.7653611632270169
Decision tree Score for max_depth =  32 with criterion entropy is: 0.7723968105065666
Decision tree Score for max_depth =  33 with criterion gini is: 0.7654784240150094
Decision tree Score for max_depth =  33 with criterion entropy is: 0.774390243902439
Decision tree Score for max_depth =  34 with criterion gini is: 0.7664165103189493
Decision tree Score for max_depth =  34 with criterion entropy is: 0.7772045028142589
Decision tree Score for max_depth =  35 with criterion gini is: 0.7646575984990619
Decision tree Score for max_depth =  35 with criterion entropy is: 0.7729831144465291
Decision tree Score for max_depth =  36 with criterion gini is: 0.762781425891182
Decision tree Score for max_depth =  36 with criterion entropy is: 0.7765009380863039
```

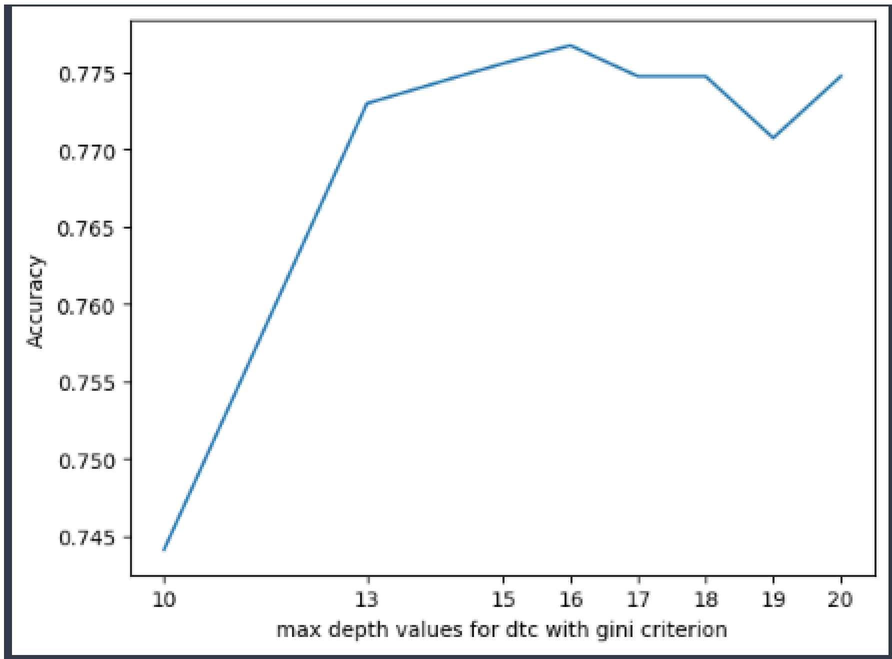


```
The best score with GINI criterion is  77.87 % with max depth of  15
```

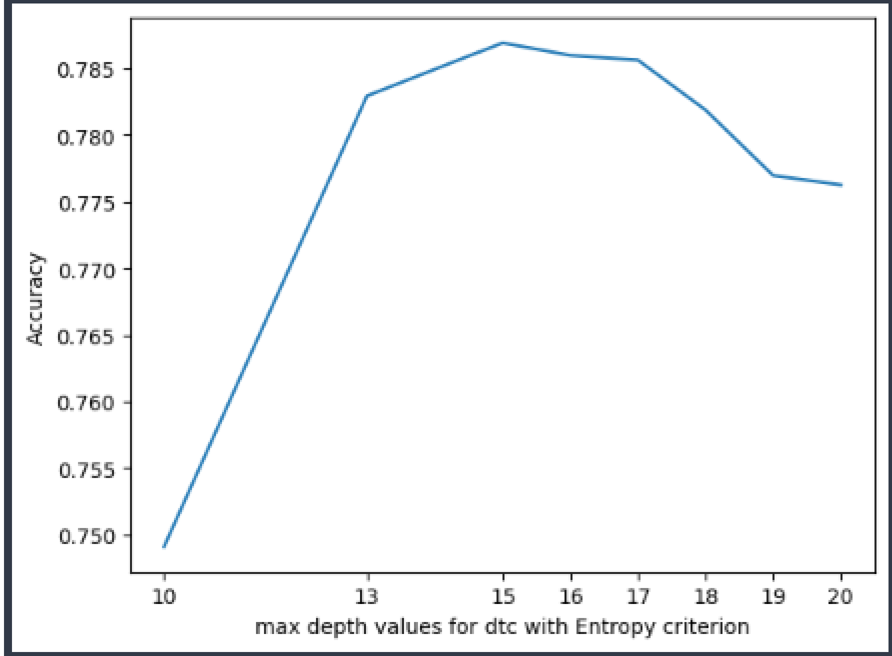


```
The best score with GINI criterion is  77.87 % with max depth of  15
The best dtc model is a model with criterion:  entropy and with max depth:  15
```

After observing this, we can see that after a max depth of 15 we have a downward trend in the accuracy, so we changed the depths to [10, 13, 15, 16, 17, 18, 19, 20] and found the best depth and criterion. The best max_depth was 17 and the best criterion was entropy with an accuracy of 78.69% and running time of 43.2 sec. This is a lot more time to run compared to knn even tough we get a worse accuracy.

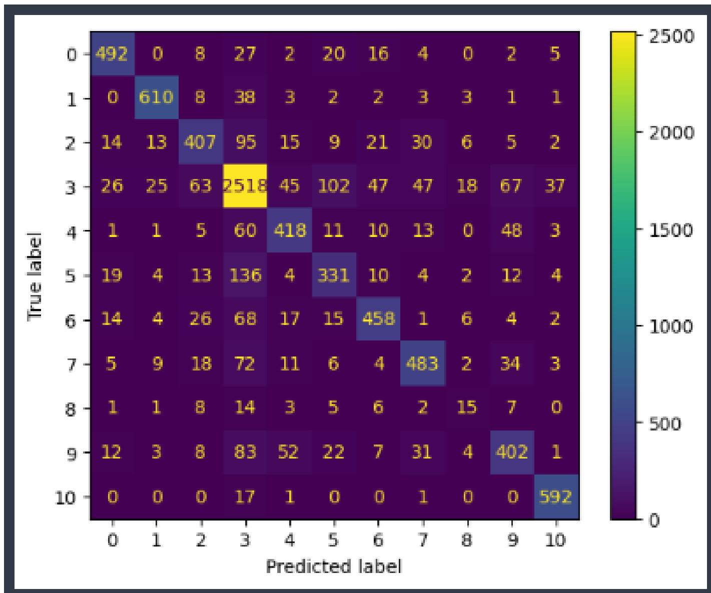


```
The best score with GINI criterion is  77.67 % with max depth of  16
```



```
The best score with Entropy criterion is  78.69 % with max depth of  15
The best dtc model is a model with criterion:  entropy and with max depth:  15
The best model give us an accuracy of  78.69
time to run the best dtc classifier with best hyperparameters:  43.234375 Seconds
```

Confusion matrix for best decision tree model



	0	1	2	3	4	5	6	7	8	9	10
0	492	0	8	27	2	20	16	4	0	2	5
1	0	610	8	38	3	2	2	3	3	1	1
2	14	13	407	95	15	9	21	30	6	5	2
3	26	25	63	2518	45	102	47	47	18	67	37
4	1	1	5	60	418	11	10	13	0	48	3
5	19	4	13	136	4	331	10	4	2	12	4
6	14	4	26	68	17	15	458	1	6	4	2
7	5	9	18	72	11	6	4	483	2	34	3
8	1	1	8	14	3	5	6	2	15	7	0
9	12	3	8	83	52	22	7	31	4	402	1
10	0	0	0	17	1	0	0	1	0	0	592

Altorugh we have a worse accurasy and longer running time than the best knn classifier we found, we did solve all the wrong predictions on the dgit 3. This can suggest that our assumption that using a non training algorithm like knn can lead to alot of wrong predictions on digits that occur alot compared to a training algorithm like decsion tree. We still have some ways to go to get a better accuracy. We try another classifier.

Random forest classifier

Random forest is a flexible and easy to use machine learning algorithm shat produces good results without needing to tune a lot of hyper parameters, this is good for us since we have been tuning of two hyper parameters in the two first classifiers without getting the best accuracy. We try using this classifier after the decision tree classifier since random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset. To put it simply, random forrest builds multiple decision trees together to get a more accurate prediction. We used the default max_subsamples = 2.

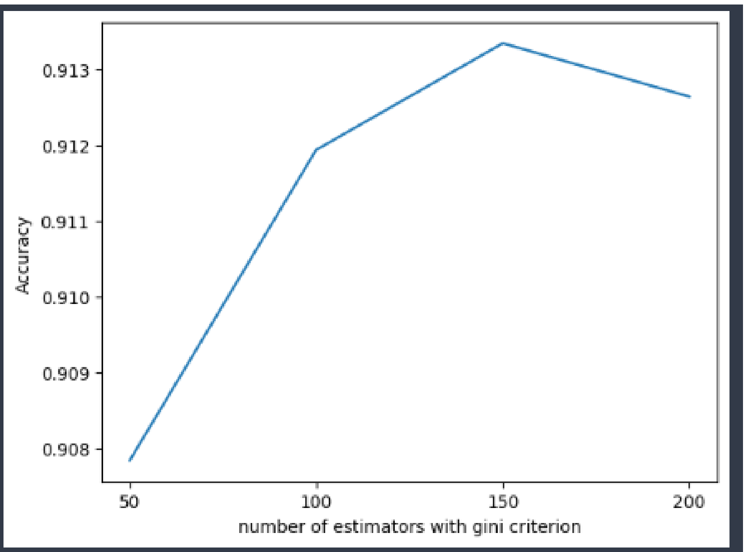
Advantages: Handles missing values well, less impacted by noise, flexible with both classification and regression problems

Disadvantages: Requires much computational power, requires much time to train since it combines several decision trees, small change in data can change the forest considerably as well

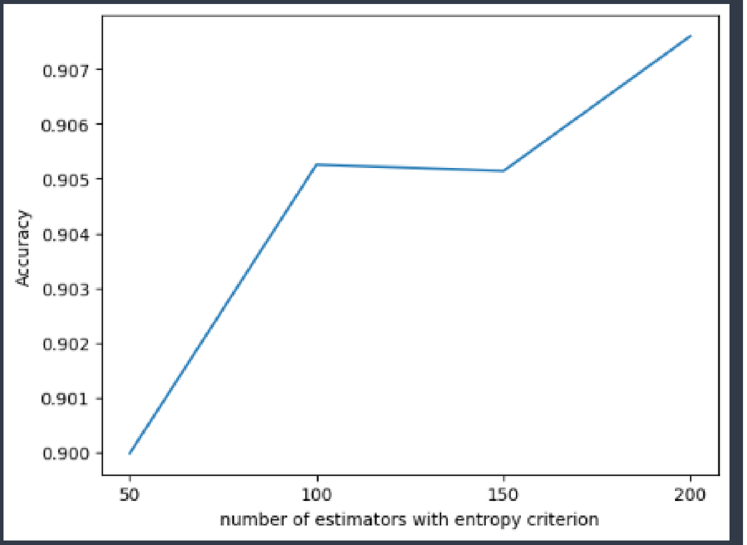
<u>Hyperparameters:</u>

- N_estimators, the number of trees in the forest we tested on 50, 100, 150, and 200. We chose these values since the default is 100, so we chose number of estimators bellow the default and number of estimators above
- Criterion the function to measure the quality of a split. We used entropy and gini. As we did in decision tree classifier.

We find that the best random forest classification is a model with criterion gini and number of estimators as 150. This gives us an accuracy of 150%

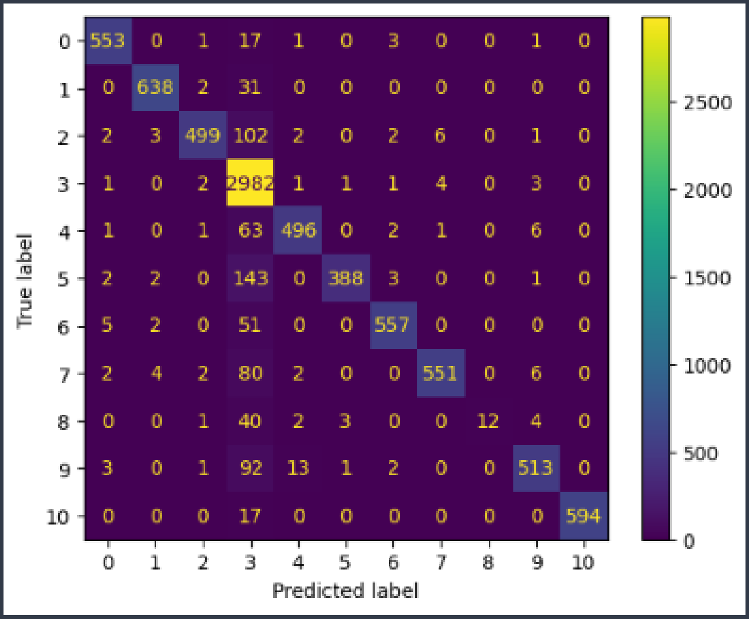


```
The best score with GINI criterion is  91.33 % with number of estimators  150
```



```
The best score with Entropy criterion is  90.76 % withumber of estimators  200
The best rfc model is a model with criterion:  gini with number of estimators: 150
The best model give us an accuracy of  91.33
time to run the best rfc classifier with best hyperparameters:  181.46875 Seconds
```

<u>Confusion matrix for best decision Random Forest classification</u>



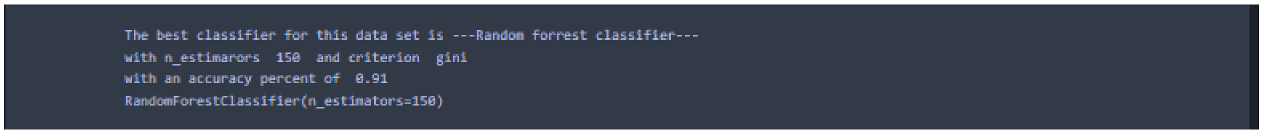
	0	1	2	3	4	5	6	7	8	9	10
0	553	0	1	17	1	0	3	0	0	1	0
1	0	638	2	31	0	0	0	0	0	0	0
2	2	3	499	102	2	0	2	6	0	1	0
3	1	0	2	2982	1	1	1	4	0	3	0
4	1	0	1	63	496	0	2	1	0	6	0
5	2	2	0	143	0	388	3	0	0	1	0
6	5	2	0	51	0	0	557	0	0	0	0
7	2	4	2	80	2	0	0	551	0	6	0
8	0	0	1	40	2	3	0	0	12	4	0
9	3	0	1	92	13	1	2	0	0	513	0
10	0	0	0	17	0	0	0	0	0	0	594

```
Entire program takes  1603.375 to run
```

We can see that in the column of the digit 3 we have a lot of values; this means that the model predicts 3 on a lot of the other digits. It could seem that if we flattened the data so that we had equal somewhat number of digits and reduced the amount of digit 3 we could improve the accuracy for the models. From this confusion matrix we see that most of the loss in accuracy is from the digit 3.

Summary and final model selection

In the end we get the model with the best accuracy out of the three. Bello is a picture of what the code returned as the best classifier based on the model with the best accuracy.



```
The best classifier for this data set is ---Random forrest classifier---
with n_estimarors  150  and criterion  gini
with an accuracy percent of  0.91
RandomForestClassifier(n_estimators=150)
```

Although the random forest gives us a good accuracy of 91.33 % it still takes a long time (181 seconds) compared to the other algorithms, this is something to think about. If we wanted the fastest algorithm that gives us ok results, we would choose KNN as it gives us a 82.4% accuracy with a running time of only 0.03 seconds since we only use 2 k values.

Challenges

- Felt a little limited by programming skills: Am relatively new to python, so a lot of time goes to reading python documentation that could be used to implement instead. Also find it difficult to program in jupyter notebook without autocomplete or syntax highlighting for variables that does not exist for example.
- Given more time I could implement methods for code that I used several times, like the plotting and testing of hyperparameters on the different models. I would also love to test out neural networks and other libraries like keras, pytorch and tenserflow if I had more time.
- Running trough the classifiers with many different hyperparameters took quite a while, so another challenge was waiting time, I tried to preprocess the images to get the running time lower, but ultimately did not get much time save from this, probably because of wrong implementation of turning the pixels to 1 or 0 based on value.