Assignment 4: Machine Learning

Problem Statement: Image orientation classification based on the values of RGB values for a given image.

Training Dataset: The train dataset consists of approximately 40,000 images. It actually consists of 10,000 images rotated 4 times to give 4 time as much data. We need to predict the orientation of each image which is multi class and has 4 unique values namely 0, 90, 180 and 270.

Test Dataset: The test dataset each image occurs only once and has 1000 images for which we need to predict the orientation using the classification algorithms.

We have implemented 3 algorithms for this task namely K Nearest Neighbours (KNN), Ada boost and Random Forest. Parameters for each model were tweaked to optimize the training time and classification accuracy on test dataset.

KNN

K Nearest Neighbours algorithm calculates the distance of each row in test dataset with each row in the training dataset. Then we identify the k rows(neighbours) from training dataset which are closest to each row in test dataset. A majority vote from the k neighbours is taken to assign a class to each row in the test dataset.

For KNN, we have implemented the program such that it can take different values of K and you can also change the distance metric to be used. There are 2 options available for the distance metric:

1. Euclidean Distance
2. Manhattan Distance

We have used pickle library of Python to store the data structures used in python in .pkl files on the disk. The model for KNN is stored in 'nearest_model.pkl'. In the training phase, the train dataset is stored in the 'nearest_model.pkl' file. This is why training time of KNN is not important for us.

During the test phase, the train dataset is loaded from the 'nearest_model.pkl' file and then classification is performed as described above. The image name and predicted labels for each image in the test dataset are stored in the file 'output.txt'.

The accuracies for different values of K and different distance metric are listed below.

K	Distance Used	Classification Accuracy	Test Time (seconds)
5	Manhattan	71.13%	356.78
10	Manhattan	71.68%	385.93
15	Manhattan	70.91%	381.88
20	Manhattan	71.02%	379..8
25	Manhattan	73.45%	361.63
30	Manhattan	71.9%	344.35
35	Manhattan	72.23%	361.1
40	Manhattan	72.01%	377.54
45	Manhattan	71.79%	362.61
50	Manhattan	71.46%	360.13

K	Distance Used	Classification Accuracy	Test Time (seconds)
5	Euclidean	70.58%	337.67
10	Euclidean	71.35%	383.36
15	Euclidean	70.8%	377.95
20	Euclidean	71.9%	344.78
25	Euclidean	71.9%	411.58
30	Euclidean	71.02%	354.8
35	Euclidean	71.68%	347.97
40	Euclidean	72.79%	336.9
45	Euclidean	72.12%	360.97
50	Euclidean	71.9%	427.99

For K=25 and when Manhattan distance is used, we get the best classification accuracy of 73.45%.

Adaboost

Adaboost algorithm uses an ensemble of weak classifiers to classify the datapoints in test dataset. We have used decision stumps as weak classifiers. Each decision stump compares 2 different features which results in a total of 192C2 decision stumps for each model. 6 different models were constructed as listed below:

1. Model to distinguish between 0 and 90 orientation
2. Model to distinguish between 0 and 180 orientation
3. Model to distinguish between 0 and 270 orientation
4. Model to distinguish between 90 and 180 orientation
5. Model to distinguish between 90 and 270 orientation
6. Model to distinguish between 180 and 270 orientation

The classification is made based on the majority votes we get from the 6 models above for each row in the test dataset. Each model can have 192C2 decision stumps(weak classifiers), where 192 is the number of features used ,and corresponding alphas which are weights of each weak classifier in each of the 6 models described above. From the 192C2 decision stumps, only the decision stumps with error less than 0.5 were considered as weak classifiers.

We have used pickle library of Python to store the data structures used in python in .pkl files on the disk. The model for adaboost is stored in ‘adaboost_model.pkl’. In the training phase, the weak classifiers for each model and their aplhas are stored in a dictionary for each of the 6 models. A list of such dictionaries is stored in the ‘adaboost_model.pkl’ file.

During the test phase, the list of models is loaded from the ‘adaboost_model.pkl’ file and then the models are used to classify each row in the test dataset. The image name and predicted labels for each image in the test dataset are stored in the file ‘output.txt’.

We performed experiments with different number of decision stumps used in each model and the results are shown below:

No. of decision stumps used (n)	Classification Accuracy	Train time (seconds)	Test time (seconds)
10	45.58%	2.98	1.49
20	51.44%	3.79	2.57
30	54.2%	4.82	3.65
40	55.86%	5.29	4.73
50	58.63%	6.43	6.41
60	58.52%	7.85	7.05
70	59.4%	8.18	15.5
80	59.73%	9.02	9.49
90	59.73%	9.85	10.94
100	59.4%	10.81	13.38
110	59.84%	11.84	12.16
120	60.4%	15.6	15.72
130	61.06%	13.9	14.98
140	59.62%	14.8	14.94
150	61.06%	14.73	18.58
160	60.62%	15.69	17.45
170	60.62%	17.2	18.27
180	59.95%	17.65	19.41
190	60.07%	18.29	22.1
200	60.07%	19.01	22.58
250	63.27%	24.6	27.76
300	65.04%	27.29	37.08
350	65.04%	31.09	44.04
400	64.38%	35.13	43.32
450	65.7%	39.89	47.7
500	65.7%	43.46	53.79
600	66.7%	51.29	64.29
700	66.9%	58.97	75.56
800	66.9%	70.46	88.23
900	66.04%	77.84	107.73
1000	65.93%	84.08	108.9
1500	68.03%	138.13	172.11
2000	68.25%	226.58	210.12
2500	68.58%	227.6	257.88
3000	69.91%	258.5	347.86
3500	70.46%	292.78	388.92
4000	69.69%	335.78	437.95
4500	71.13%	378.91	490.27
5000	70.68%	428.78	549.1
5500	72.12%	556.6	598.15
All	70.9%	1599.87	2060.66

Note: The time calculated is on burrow. The results may not be exactly reproducible for the time taken to run the code as it depends on the resource utilisation of burrow at a given time.

We achieved an accuracy of 72.12% when 4500 decision stumps were used in each of the 6 models described above. The time taken to train the model is 378.91 seconds and test time was 490.27 seconds. When all possible decision stumps (192C2 or 18,336 decision stumps) are used, we get an accuracy of 70.9% but the training time is 1600 seconds approximately and testing time is 2060 seconds which is over 30 minutes. So best adaboost model is when we use 5500 decision stumps in each model as the test time is just 598 seconds or 10 minutes approximately and we have an accuracy of 72.12%. We can also conclude that the accuracy doesn't increase much after we use more than 5500 decision stumps. Moreover the test time will cross the threshold of 10 minutes which we don't want.

Random Forest

Random Forest is a supervised Machine Learning algorithm. It uses an ensemble of Decision Trees to over come the variance observed in the results of a Decision Tree. The "forest" it builds, is an ensemble of Decision Trees, most of the time trained with the "bagging" method. The general idea of the bagging method is that a combination of learning models increases the overall result. To say it in simple words: Random forest builds multiple decision trees and merges them together to get a more accurate and stable prediction.

In this algorithm, a portion of dataset is used to generate **N** decision trees each with depth **d**. We are using 14 (square root of 192) features which are selected randomly from the available 192 features for each decision tree. We are using 15% of the dataset for our experiments. This is the training phase of the algorithm.

We have used pickle library of Python to store the data structures used in python in .pkl files on the disk. The Random Forest is stored in 'random_forest_model.pkl'.

During the test phase, model is loaded from the 'random_forest_model.pkl' file. All the Decision Trees are used to make prediction for each row in the test dataset. The prediction is made based on the maximum voting from the **N** Decision Trees generated during the train phase. The image name and predicted labels for each image in the test dataset are stored in the file 'output.txt'.

The training time for Random Forest is very high as compared to the time taken to test the algorithm on test dataset. We performed the experiments to understand the effect of **N** and **d** on the classification accuracy, training time and testing time for our implementation. The results are shown below:

Max depth (d)	Classification Accuracy	Train time (seconds)	Test time (seconds)
5	52.3%	1200	0.17
10	56.5%	1200	0.20
15	56.5%	1234	0.17
20	56.5%	1235	0.17
25	56.5%	1236	0.27
30	56.5%	1237	0.17
35	56.5%	1143	0.22
40	56.5%	1087	0.17
45	56.5%	1048	0.17
50	56.5%	1089	0.17
55	56.5%	1099	0.17
60	56.5%	2003	0.27

No. of trees (N)	Classification Accuracy	Train time (seconds)	Test time (seconds)
2	56.5%	1200	0.20
3	50.1%	1595	0.23
4	49.5%	2085	0.50

Value of Number of trees (**N**) was set to 2 when experiments to understand the effect of changing the maximum depth of trees were performed. Max depth (**d**) of 10 was used when experiments to understand the effect of changing number of trees were performed. As the accuracy of the model starts decreasing as we increase the number of trees, our best accuracy for random forest is when the max depth (**d**) is 10 and number of trees (**N**) is 2 and giving an accuracy of around 56.5%.

Best Model

Our implementation KNN algorithm performs the best for the given dataset. We achieved an accuracy of 73.45%. when Manhattan distance is used and K=25 is used. Experiments with different training dataset size were performed to understand the performance of KNN with K=25 and Manhattan distance.

Split_Dataset.py script has been used to split the dataset. It splits the training dataset by selecting the x% rows from training dataset randomly where, x is from 10 to 100. Seed has been used from the random library of Python to ensure reproducibility.

The datasets are stored with filenames 'train-data-0.1.txt', 'train-data-0.2.txt', …… 'train-data-0.9.txt'. Complete training dataset is available in 'train-data.txt'. The corresponding models are stored with filenames 'best_model_0.1.pkl', 'best_model_0.2.pkl', …… 'best_model_0.9.pkl'. Model for the complete training dataset is stored in 'best_model.pkl'.

The results are given below:

Percentage of Training Dataset	Classification Accuracy	Test Time (seconds)
10	65.6%	42.67
20	66.37%	92.43
30	67.26%	127.18
40	67.26%	157.49
50	66.59%	200.23
60	68.03%	253.77
70	68.69%	290.59
80	68.58%	318.78
90	68.91%	331.97
100	73.45%	361.63

Note: The time calculated is on burrow. The results may not be exactly reproducible for the time taken to run the code as it depends on the resource utilisation of burrow at a given time.

We analysed the labels of misclassified images and following ae the results:

Actual Label	No. of misclassified Imaged
0	63
90	53
180	62
270	62

We can see that labels 0, 180 and 270 have the maximum number of misclassified images. On deep diving into these categories, we get the following results:

Actual Label	Predicted Label	No. of images
0	90	14
0	180	31
0	270	18
180	0	32
180	90	21
180	270	9
270	0	19
270	90	28
270	180	15

We observe that most misclassified images are the ones whose actual label is 0 and predicted label is 180, whose actual label is 180 and predicted label is 0 and whose actual label is 270 but predicted label is 90.

Actual label 0 predicted as 180:

Some examples of images whose actual label is 0 and predicted label is 180 are shown below:

It is clear from the images above images which have lighter shades in the bottom of the image confuse the model and the images are misclassified as 180 instead of 0.

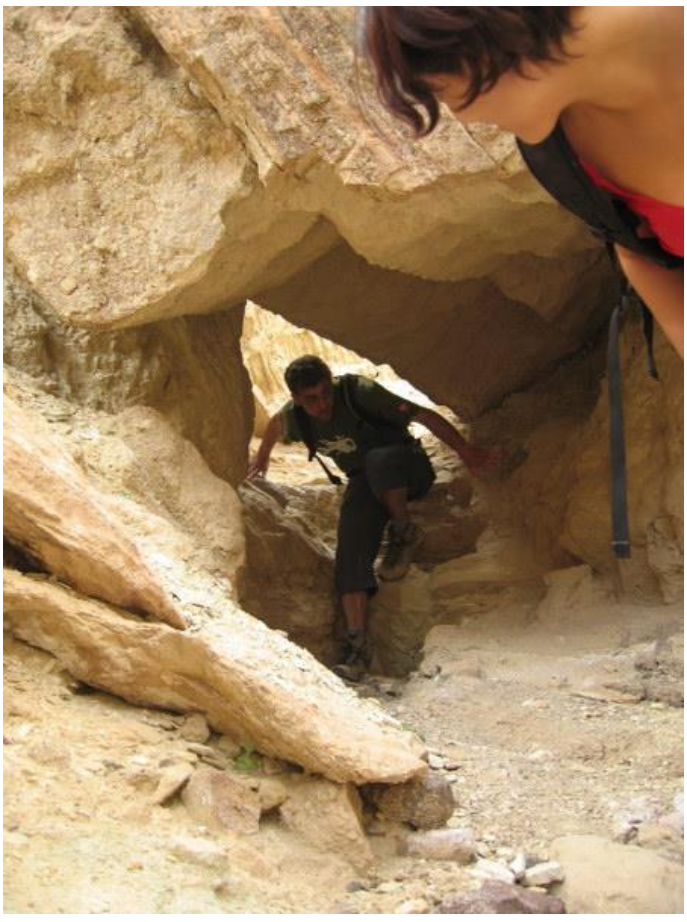
Actual label 0 predicted as 0:

Some examples of correctly classified images whose label is 0 and predicted label is also 0 are shown below:

It is evident from the images above that model is indeed looking for lighter shades in the top part of the image to predict its label as 0.

Actual label 180 predicted as 0:

Some examples of images whose actual label is180 and predicted label is 0 are shown below:



From the above images, we can see that the misclassified images have very similar colours in the top and bottom of the image which confuses the model. For the third image, even humans would classify it as 0 but it is actually an image of shadows of hikers in a water body and as the top part of the image has lighter shade, it is misclassified as 0 instead of 180.

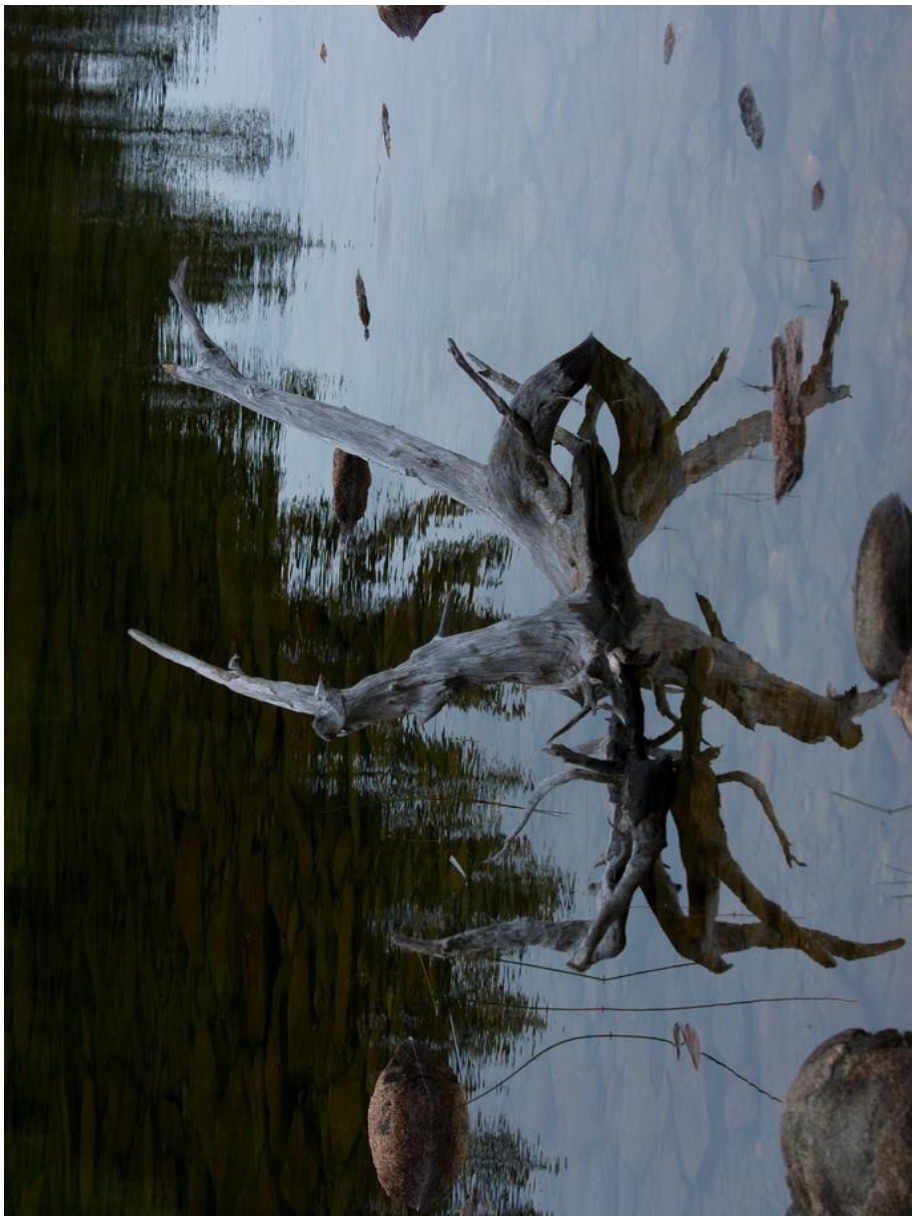
Actual label 180 predicted as 180:

Some examples of images whose actual label is 180 and predicted label is 180 are shown below:

From the above images, we can conclude that model is indeed looking for lighter shades in the bottom of the image to classify it as 180.

Actual label 270 predicted as 90:

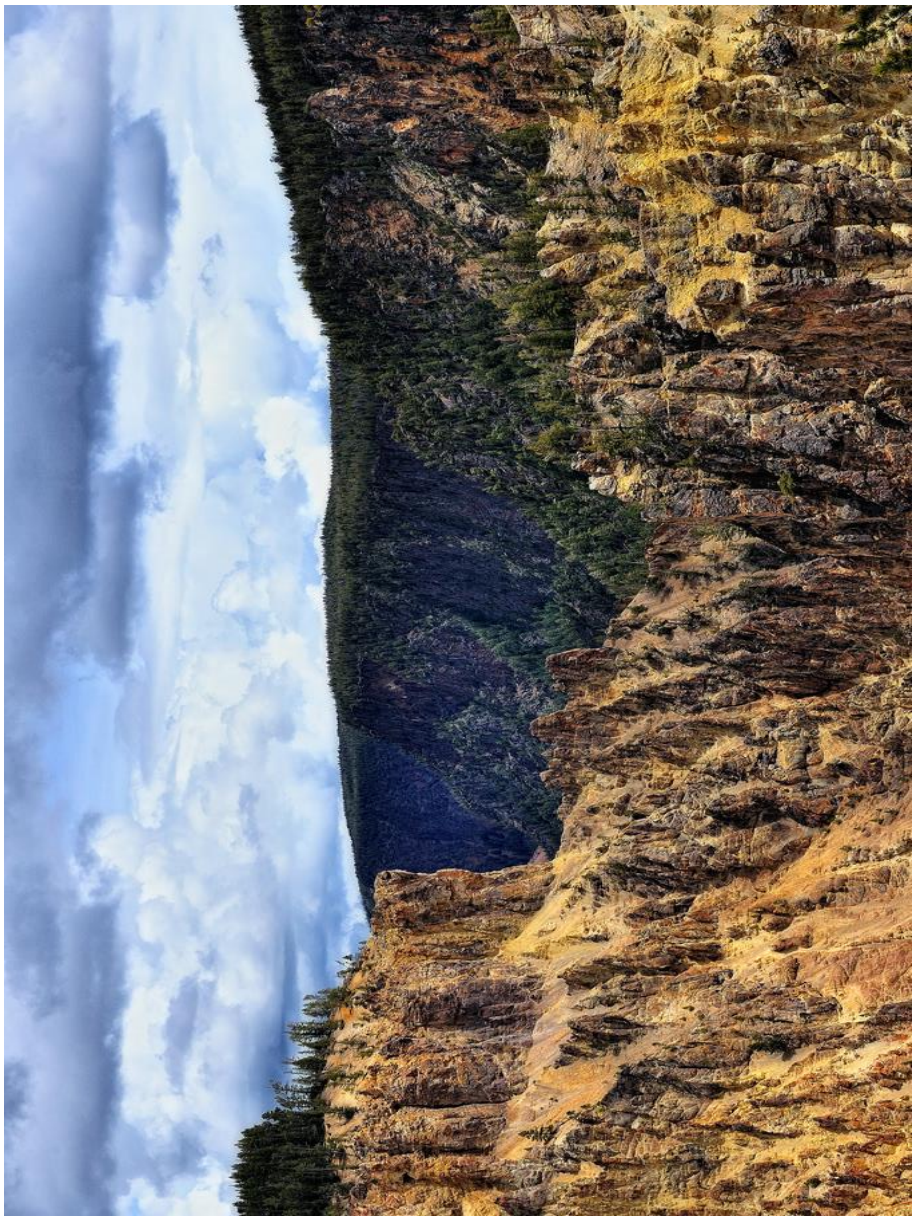
Some examples of images whose actual label is 270 and predicted label is 90 are shown below:

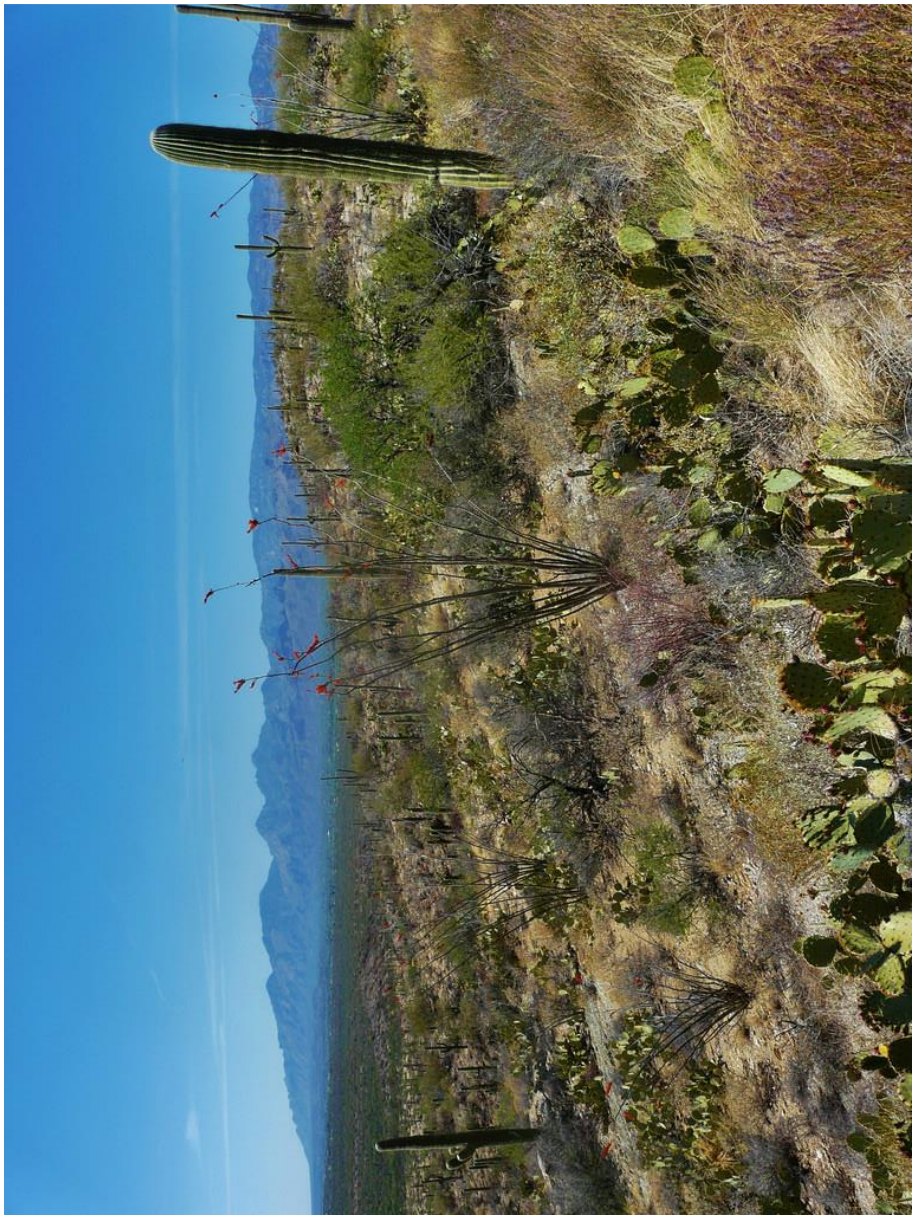


From the images above, we can see that the misclassified images have a lighter shade in the right hand side of the image. This leads to model misclassifying them as 90 even thought their correct label is 270.

Actual label 270 predicted as 270:

Some examples of images whose actual label is 270 and predicted label is 270 are shown below:





From the above images, we can conclude that the model looks for lighter shades in the left part of the image to classify it as 270.

Conclusion

From the experiments done for KNN, Adaboost and Random Forest we identified the best set of parameters that can be used for each model for the dataset given to us. We also identified KNN as the algorithm which can recommended to a potential client for the problem at hand as it takes less time to test, is easy to understand and performs surprisingly well for the given problem even though it's a very simple algorithm as compared to Ada Boost and Random Forest.

Through the analysis done, we can say that the model looks for lighter shades(colours such as blue for the sky) in the images to predict the orientation. The model is basically classifying an image with label 0 if lighter shade is found in the top of the image, 90 if lighter shade is found in the right hand side of the image, 180 if lighter shade is found in the bottom of the image and 270 if lighter shade is found in the left hand side of the image.

Future Work

The accuracy can be further increased by eliminating features and using only the features which are important and will help the most in predicting the orientation of a given image. There are different strategies that can be employed:

1. Removing highly correlated features
2. Variable elimination using the variable importance from Random Forest
3. Feature space reduction using algorithms such as Linear Discriminant Analysis (LDA)

References

1. KNN tutorial available on KD Nuggets
2. Random Forest tutorial available on Machine Learning Mastery
3. Random Forest tutorial by Searene available on GitHub
4. Random Forest Tutorial by Josh Gordon available on GitHub