

Assignment 4 Learning

UserIds:

shasmitt-asagar-tdeoras

ADABOOST

Understanding the Problem:

Adaptive Boosting is a classifier that uses learners to classify data points. Image Classification using the numerical feature vectors extracted from images. The total number of dimensional feature vectors are 192 for 8 x 8-pixel ratio of thumbnail.

Ref:<https://pdfs.semanticscholar.org/6f84/2d1d6f5954cf697c8c988ef18ab49bd75f39.pdf>

Description of Program Working:

For this Program we have used decision stumps based on pixel values. Upon observing the numerous examples in the image dataset. We found that the images contained a higher amount of landscapes and pictures with sky in them. To use this to our advantage we have implemented our first stump by considering the pixel values corresponding to blue color. These are checked in the top, right, bottom & left column in the image.

For Classification we check the value in each of these columns for the pixel values. If the bottom column has the highest concentration of blue pixel values, then the image is classified as rotated on its own axis to 180 deg. Similarly, the brown color valued pixels are checked to verify the presence of ground in the image.

The Adaboost Classifier trains on the entire dataset and weighs the performance of different classifiers. For each error made by the

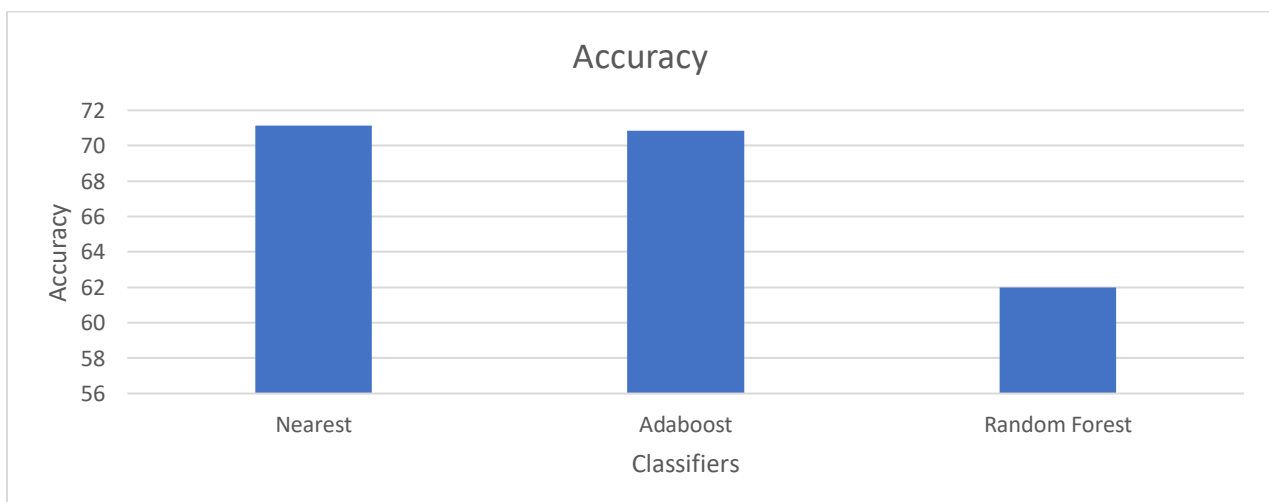
classifier we add the value of the observation's weight to the error and update it. Upon completing the program weights of all correct classifications are updated using the weight update rule. This is performed for every classifier and their corresponding weights are written in the output_file.txt. The predicted labels are also output to a file with image ids and the predicted orientation on them.

Assumptions:

Initially all the weights of the observations are assumed as $1/n$ with n being the total number of observations. Also we have used the weight observation rule of $w[i] * (\text{error} / 1 - \text{error})$ and the weights for the stumps are calculated using : $\log(1 - \text{error} / \text{error})$.

Classification Accuracies (Time unit= Min)

	Nearest	Adaboost	Random Forest
Accuracy	71.15	70.84	62
Training Time	2	5	40
Test Time	6	0.5	1





Images Correctly Classified:

1. test/6568344.jpg 90



Using a majority of pixels being blue in the sky the result is correct.

2. test/6694765677.jpg 0



Using a majority of pixels being brown on the ground this result is correct.

Images Misclassified:

1. test/6402320985.jpg 0



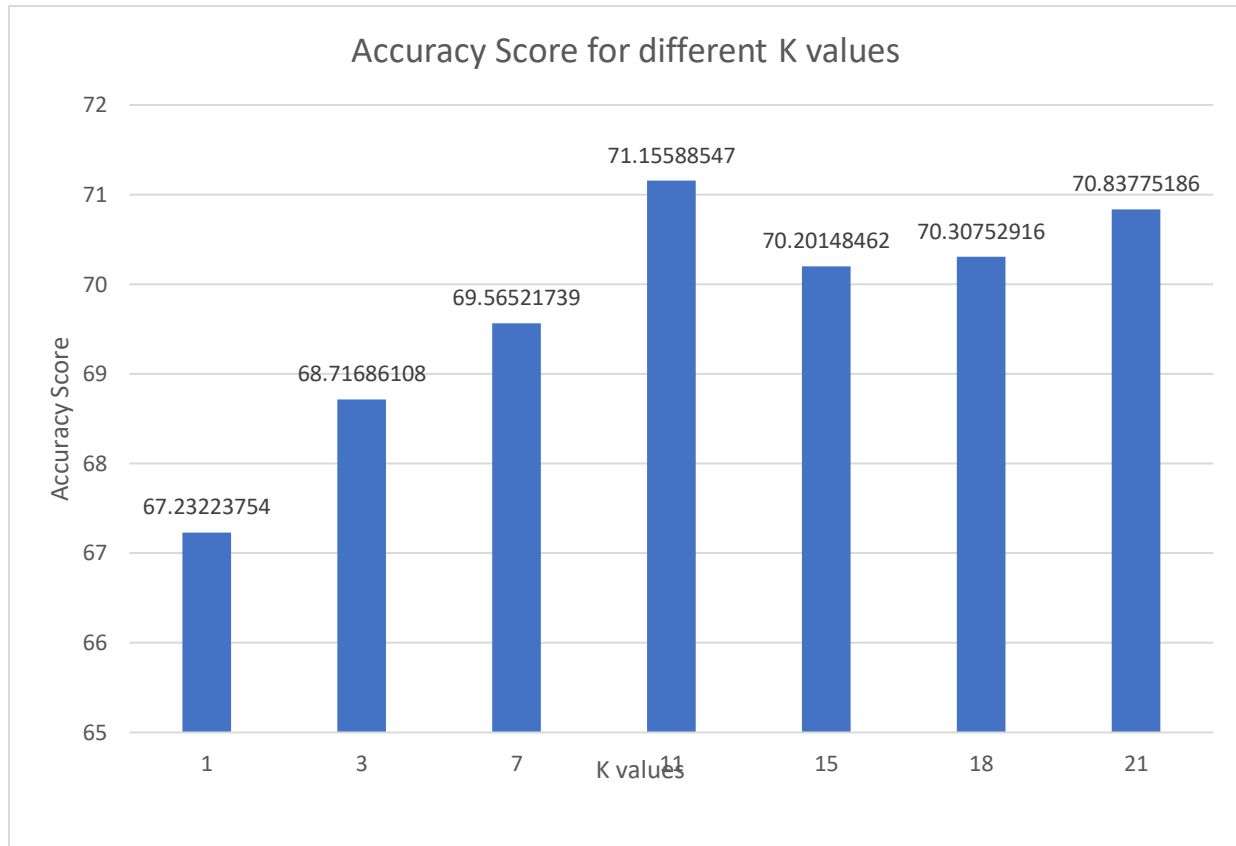
This represents the image of an bird with low number of blue or brown pixel segregation and therefore is misclassified.

2. test/68200070.jpg 180



This represents water flowing with mixed colors for pixels thereby increasing rate of error.

KNN Classifier



Accuracy for different values of K.

*Actual Accuracy will be less because we took 16000 points from training instead of 37000 to build a model file of smaller size because of the size constraint used by github.

Maximum accuracy = 71.155 , is observed for K = 11




Algorithm functioning:-

- Training Data and Testing data files are read using pandas library.
- In training phase ,the training data is converted to numpy array.
The training data, images column and respective degree column

of training files are separately stored in '*model_pram*' parameter. This data is written in *model_nearest.txt* using pickle library.

- In testing phase ,the testing data is converted to numpy array. The tesing data, images column and respective degree column of testing files are separately stored in *tst*, *tst_img*, *tst_y_lbl* parameter. The traing data is fetched using pickle library which was written in '*model_nearest.txt*' during training phase.
- *KNN(tst,trn,tr_y_lbl)* function calculates the Euclidian distance for each data point then sorts the distance in increasing order. Then based on different k values voting is done for K nearest neighbors and maximum count of degree is assigned to the image.
- *output(tr_img,final_value,tr_y_lbl)* function writes the *KNN_file.txt* that has all the testing images and predicted degrees.

Correct OUTPUT images and degree from test file:-

Image	Predicted Degree	Actual Degree
	0	0
	90	90
	90	90

Incorrect OUTPUT images and degree from test file:-

Image	Predicted Degree	Actual Degree
--------------	-------------------------	----------------------



180

90



0

270



270

90

The incorrect output is due to the maximum voting of k nearest neighbors of images in predicted degree which is different from actual degree.

Random Forest

Number of Trees	Accuracy(in %)
1	45
10	58
20	61
30	63
40	65

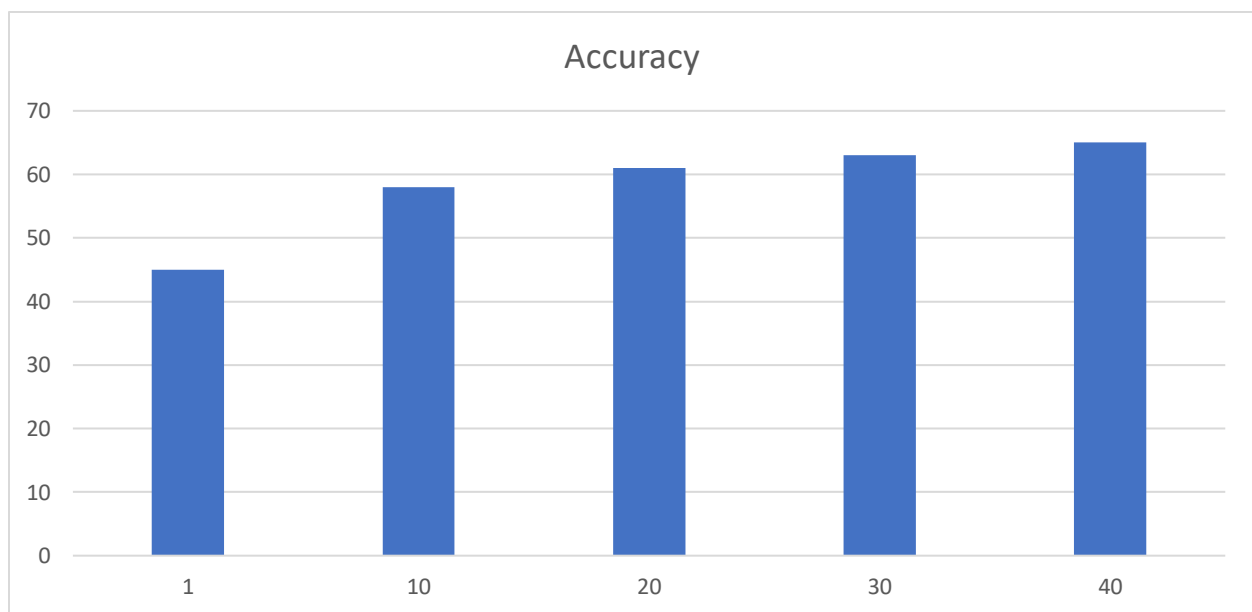


Fig 1: Accuracy for the no of trees

Number of Trees	Time Taken (in Min)
1	1
10	10
20	23
30	40
40	49

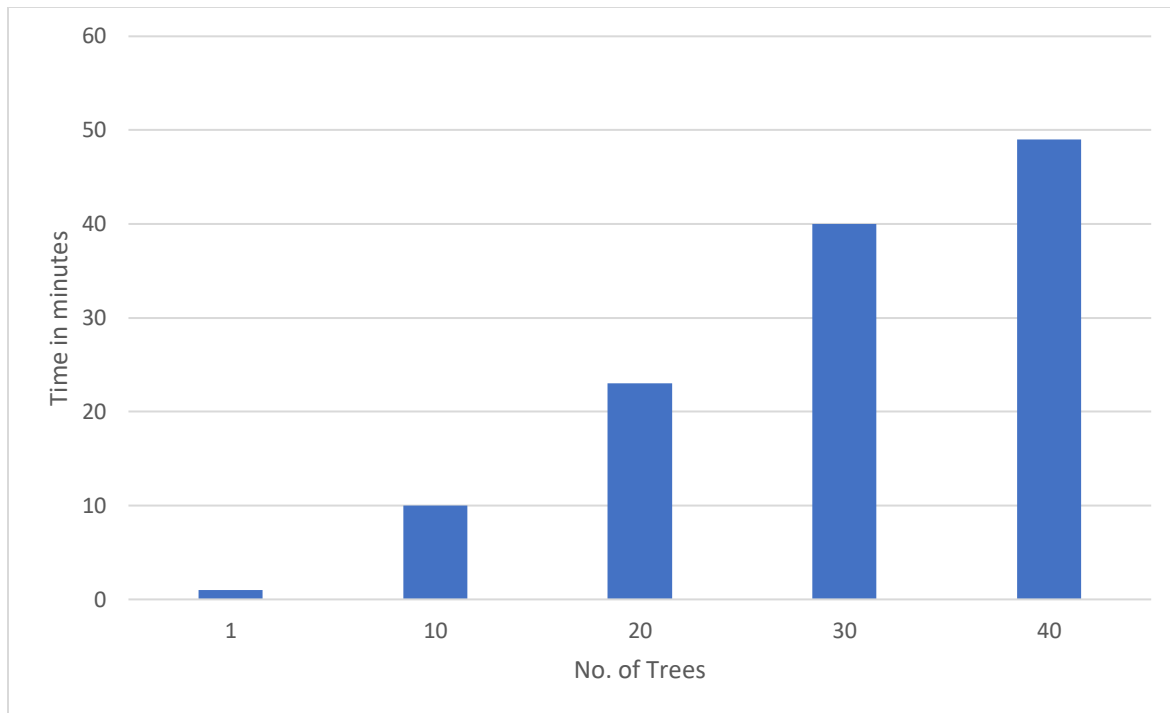


Fig 2: Time Taken For no of trees

Algorithm Functioning:

Random forest makes decision trees. For each decision tree a output is generated and there is voting. The highest occurring output is the answer.

Sampling Data:

For each tree a random sample of data is generated. Here the sample size is taken is 10000.

Sampling Attributes:

For each tree random attributes are taken. The no of attributes taken for each tree is 20.

Measure for Picking node:

Entropy was the measure used to pick the best node.(ID3)

Problem Faced:

1.Decision tree leaf : The leaf nodes didn't have a pure set. Hence the maximum occurring label was chosen.

2.Decision Tree missing branches: Some branches produced empty data. They were handled by a -1 Node.

3.Forest storage: The forest had to be stored using pickle. The root nodes were as a representation for a tree instead of all the nodes and branches.

Correct OUTPUT images and degree from test file:-

Image	Predicted Degree	Actual Degree
-------	------------------	---------------



90


90



0

0

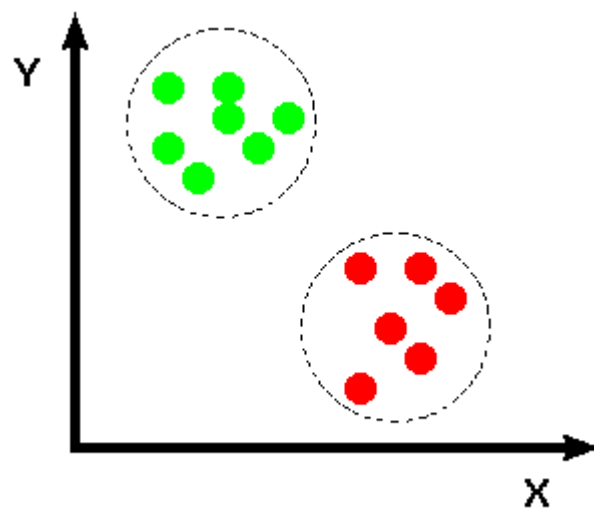
Incorrect OUTPUT images and degree from test file:-

Image	Predicted Degree	Actual Degree
	180	0

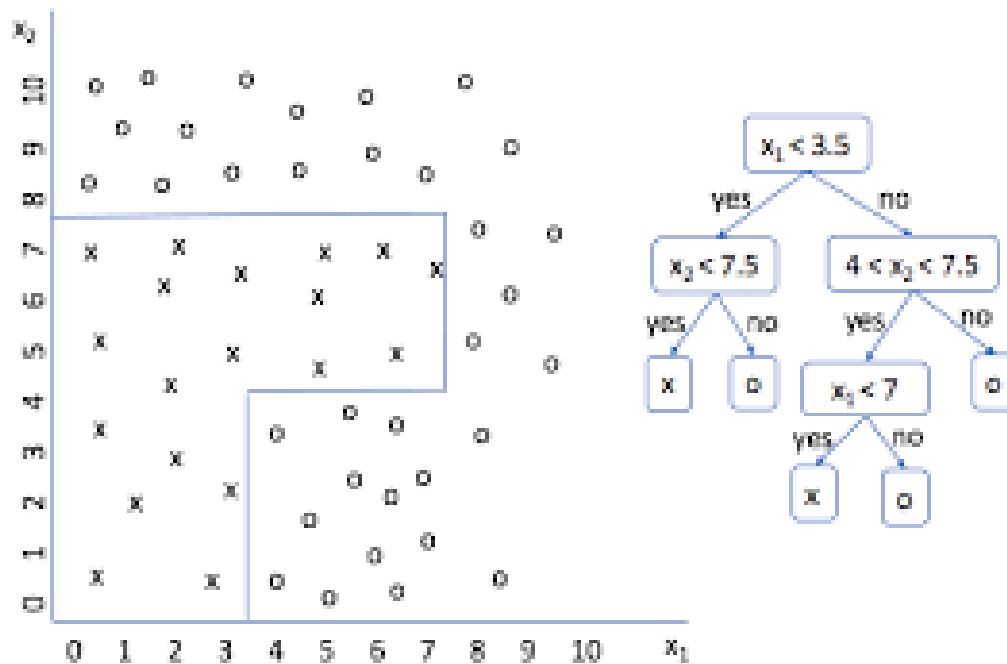
Recommendation of Classifiers to build the system:

1. Based on nature of Training Data:

KNN : If the dataset has well defined clusters such that the data points are neatly separated KNN would perform the best. The dataset should also have less number of outliers.



Decision Forest: A forest would perform the best when the datapoints can be separated by clearly defined boundaries.



*Ref Image

2. Based on Desired Consistency and Time:

If the parameters are not weighted by time constraints, then we would recommend using Random Forest as it tends to give better accuracy with good training and higher number of decision trees.