Assignment IV Machine Learning

CSCI B-551 Elements of AI

Problem Statement: Identify the orientation of the images based on the data given about the RGB pixel intensities of the image.

Result: The class variable is orientation = [0, 90, 180, 270]

Data:

- The training data consists of a total of 9244 unique images which are rotated 4 times. This gives us total of 36,976 training images.
- The testing data consists of 943 samples to predict accuracy.
- The feature space consists of a total of 192 features. Each image is represented as set of 8*8 pixels. Each pixel has 3 intensities Red, Green and Blue each ranging from 0 to 255.

Pixel Value Significance:

- Red-255, Green-255, Blue-255 denotes a white pixel.
- Red-0, Green-0, Blue-0 denotes a black pixel.
- Based on pixel color the distribution of R, G, B value changes.
- Green shades will have G value higher while yellow shades will have G and B distributions higher.

Aggregated new feature generation:

- For each image, we generate new features as follows:
    i. Take mean of red pixels in first row and subtract it by the mean of last red row. Multiply this difference by 4. Now take second and second last row and multiply by 3. Repeat for other rows.
    ii. Repeat for vertical rows or columns.
    iii. Repeat for other colors.
    iv. For the top, bottom, right and left 2 rows/columns we have generated a binary feature to check if the mean Blue and Red color is greater than a threshold.

Algorithms used: For prediction of orientation we implemented 3 algorithms namely:

- K Nearest Neighbor Algorithm
- Random Forest
- Adaboost
- [200,500,1000,1500,2000,2500,3000,3500,4000,4500]
- [0.633085896076352, 0.6670201484623541, 0.6617179215270413, 0.672322375397667, 0.6786850477200425, 0.6670201484623541, 0.6670201484623541, 0.6786850477200425, 0.6977730646871686, 0.6744432661717922]
- [31.136425971984863, 72.98162508010864, 147.01416397094727, 216.3439826965332, 286.308984041214, 361.4158561229706, 430.1212968826294, 488.02157330513, 562.5727634429932, 621.5597772598267]

- [0.23282432556152344, 0.5378847122192383, 0.9973359107971191, 1.3942232131958008, 1.8252842426300049, 2.2393109798431396, 2.669316291809082, 3.0198450088500977, 3.463733673095703, 3.6786930561065674]

1. **K Nearest Neighbor:**

   K nearest neighbor is a lazy algorithm. There is no training phase for this algorithm as it computes the output of each sample based on training data. This makes the actual KNN algorithm slow for end accuracy. The algorithm works on 3 components:

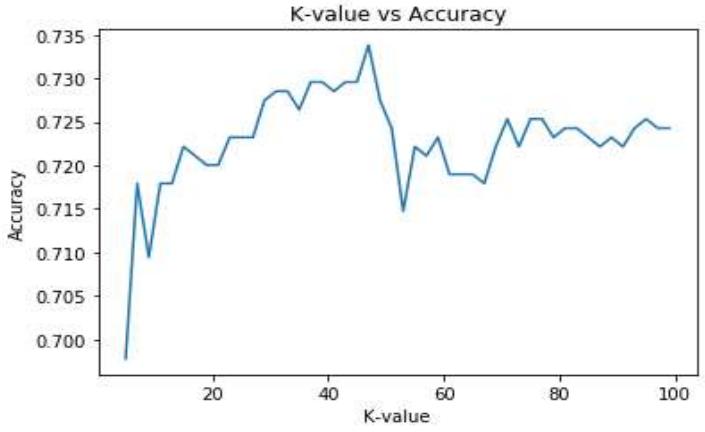
   - Train file and test file feature vector.
   - Distance calculation (Euclidean or Manhattan).
   - Max voting technique.

   After loading both training and testing files with corresponding feature vectors, we used the function of manhattan distance which is basically absolute distance between 2 points (in this case summation of all features each train sample with single test sample). We chose this distance space as it gave us the maximum accuracy for the test samples.

   K nearest neighbor working is based on calculating the proximity of single test sample within all train samples. We decide upon a value of k for the number of nearest neighbors to be found for a test sample. For this example specifically, a test sample will have a total of 36976 distances. We sort these distances and pick out top K values. Based on the class labels of each sample in K set we take the majority class label and assign is to the test sample. This process repeats for each sample in test set.

   The disadvantage of this algorithm is the time taken by it to compute class label of each test sample. We reduced this time as our number of features was reduced from 192 to 32. However, this change did not affect our accuracy. Sample results of this machine learning model are as follows:

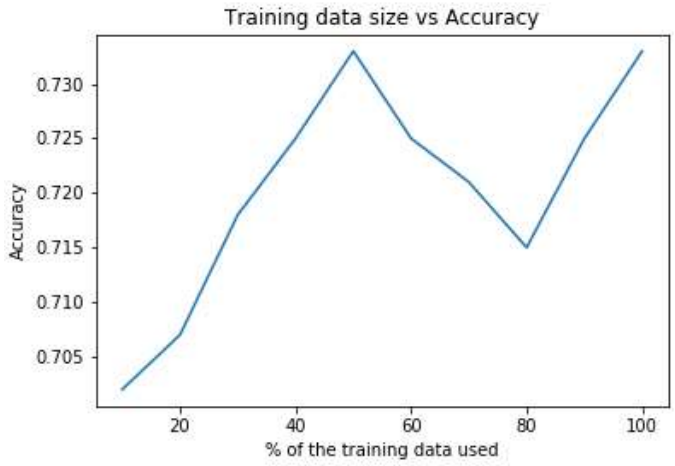
   Comparison of test accuracy based different k values:



   We tested the K value for 0 to 100. We got the best accuracy for K-value of **47**.

- **Running time :**

The approximate running time for any value between 1 to 100 is 2 to 4 minutes.
For our best value of K i.e. 47 the approximate running time on silo is:



- Correctly classified Images




- Incorrectly classified Images




If you look at the correctly classified images, you'll see a pattern. Our classifier performs well on images taken outdoor with sky and trees. We generated new features to identify on which side of the image is the intensity of Blue color the maximum. This might have helped the classifier to identify the blue color and thus the sky and usually the sky is on the upper part of image.

Our classifier performs poorly on images taken indoor, gray scale or where the sky isn't visible or obvious. This explains why the first 3 images were misclassified. If you look at the 4^{th} image (right most), the sky is present but isn't obvious. The mean intensity of Blue pixels would be much higher on the bottom part than on the left part. The actual label of this image is 270, but our classifier thought the sky was on the bottom part and predicted it 180.

2. Adaboost:

Adaboost is an algorithm which works on weak classifiers. Weak classifiers can be seen as nothing but decision trees with depth 1. For this problem we specifically take 192 features and generate combinations of 2 pixels. The algorithm can be implemented in 4 phases.

- Random features combination generation.
- Decision stump performance for weak classifier.
- Applying decision stumps to 6 models.
- Max voting technique.

We compared the intensities (0 to 255) between these 2 pixels. We can generate $^{192}C_2$ combinations. For time constraints we just select 4000 such combinations which we randomly generate with seed value. Initially, we generate weight values for each sample in train. This weight value is set as 1/(number of samples in train data). We trained 6 models based on the class labels to get max voting into picture. We implemented a one vs. one model training. The six models were:

- 0 vs 90
- 0 vs 180
- 0 vs 270
- 90 vs 180
- 90 vs 270

- 180 vs 270

For each model we first filter out the training data with the labels of the model and replace the lower value with -1 and larger value by 1. Based on our decision stump we predict the value as -1 or 1. We compare the values generated by weak classifier with actual labels.

In our case for each wrong prediction we updated the error values by the corresponding weights. Then we check if the performance of the classifier is better than a random guesswork (0.5 in this case). We also maintained a list of all correctly predicted samples. We reduced the corresponding weights of these correctly classified samples by a factor of error.

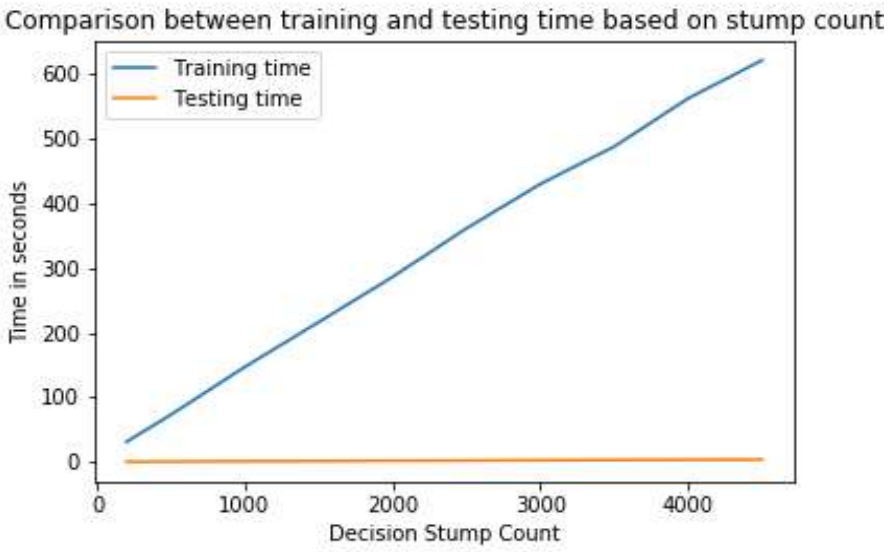
Here we use the factor as, error_value/(1-error_value)

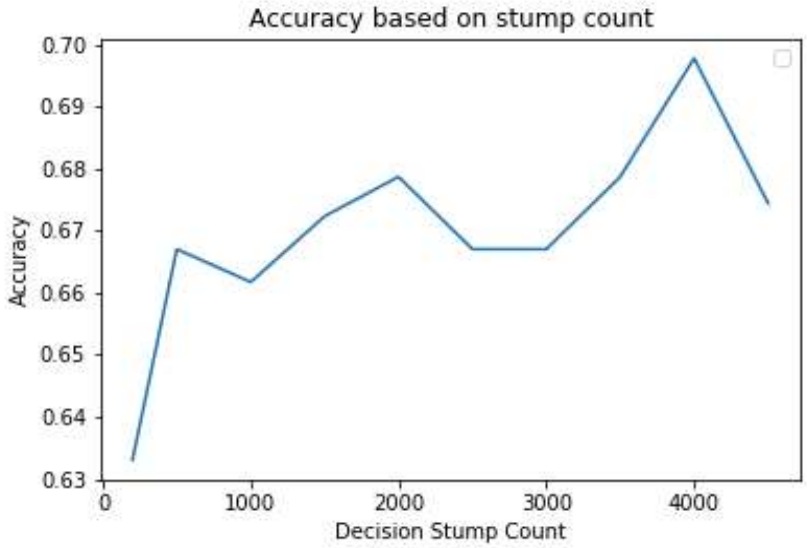
As a part of the trained model file we store the weak classifier in dictionary form with a key of stump feature combination pair. The value of this dictionary is the a_value or the power of classifier. We decide this value as log ((1-error)/error). This value decides the factor of affecting the sample test value and its label.

We form all the six models in the same way, with different feature combinations. Based on the 6 dictionaries generated before we use the same weak classifiers. We used the hypothesis generated by the weak classifier and multiply it with the classifier's a_value. We cumulatively add all these weak classifier values to get a result. We check the sign of this value to be positive or negative. We predict is as 0-90, 0-180, 0-270, 90-180, 90-270 or 180-270.

At the end of passing a test sample through all these models we take a max voting of all these models and predict the end label.

Given, below is the plot of various training and testing times for different number of decision stumps. We can see the training time increases linearly with increase in stump count. However the testing time remains constant.





We plot the accuracy for each decision stump ranging from 200 to 4500.

As a part of experiment we tried appending the existing features to the newly generated 32 features to see if it increases the accuracy or not. However this change decreased our accuracy by 4%.

- Result: For Adaboost, we used 4000 decision stumps which give us the best accuracy of **69.78%** as tested on burrow.

3. **Random Forest:**

As the name suggests, a Random forest is a collection of Decision trees. It is bagging approach where multiple Decision trees are constructed with different features and subset of training data. To better understand how the Random forest algorithm works we must first understand how a Decision tree works.

A Decision tree creates a tree like structure with Decision nodes and leaf(s). A Decision node is rule or condition for splitting the data into two buckets, one where the condition is true and other where it's not. This split is done on one value of one feature, where the Information gain is maximum (or in other words this split produces least impure buckets of data). To measure this impurity we have Gini impurity. If you are fortunate, after some splits you'll get to a point where one of your bucket has data points of only one class.

We'll call this a Leaf node. We have implemented two conditions to stop growing the tree, a Leaf node and maximum depth of tree.

In a Random forest, we build "N" such Decision trees and take their vote. So you can tune the Random forest by changing the number of trees it builds, the percent of data used to build each tree, no. of features used to split and maximum depth of tree. As you increase the number of trees your variance decreases but the training time also increases. As you increase the percent of data used to build each tree, your trees start becoming similar to each other and give a similar and probably better prediction but also increases training time. As you increase no. of features and maximum depth of tree you your predictions get better up to certain value and then decrease from there.

Initially, we considered every possible split to get the best split at each node. This means, if you have "f" features and each of those feature has 255 unique values, then you would be checking f*255 splits. To overcome this problem, we sort the unique values of a feature and split only at points that have a cluster of same class data points. This helps reduce split points and thus significantly reduces training time.

Sample Size (% of total data)	Max Depth	Accuracy
45	15	0.60
60	15	0.62
75	15	0.62
90	15	0.64
45	25	0.60
60	25	0.64
75	25	0.60
90	25	0.63
45	35	0.60
60	35	0.62
75	35	0.61
90	35	0.66
45	45	0.62
60	45	0.63
75	45	0.61
90	45	0.64
45	55	0.58
60	55	0.61
75	55	0.62
90	55	0.64

(**Note:** Because of the randomness in Random forest algorithm, these results might not be reproducible.)

The best accuracy we get for Random forest is by tuning it to the following parameters:

I. Number of trees: 70
II. Max features: “sqrt” of total features
III. Sample size of training data: 90%
IV. Max depth: 35