Assignment Report

Problem Description:
Problem: for a given image identify the orientation of the image using KNN, Adaboost, and Random Forest Models.
Specifications:
Image size: 8*8 RGB images (192 pixels)
Classified Orientations: 0 90 180 270
Training Data: 36,796 images
Testing Data: 943 images
Classifier: Adaboost:
Adaboost is an ensemble learning technique which utilizes weak classifiers and then uses voting algorithm to determine the final classification.
Abstraction of model
In training phase adaboost performs a weighted classification of all training data over each classifier. Starting from equal weights of each training data is increased if it is classified incorrectly.
Once the weights are updated they are normalized.
Depending upon the number of each training data classified wrong an error for each classifier is calculated which would eventually determine the weight of each classifier.
These weights and classifiers are used as classifier for new test data. In this example we have used small decision trees/ decision stumps as our classifiers.

How Program Works:
Training:
We first train the model using train mode of orient.py where we provide the training data, the model to train in this case adaboost and output file name.
This mode will provide a model file output which has been trained using the training data.
Command:
./orient.py train train-data.txt adaboost_model.txt adaboost

Testing:
To test the model on the test dataset we use the test mode of orient.py where we provide the test dataset, trained model and the model to be used for testing in this case adaboost.
This will generate an output.txt file which will contain predicted orientation of each test image provided in testing phase. It will also show accuracy of the model on console.
Command:
./orient.py test test-data.txt adaboost_model.txt adaboost

Algorithm Used and Key points of the code:
We have decomposed the problem into smaller binary problems where we compute all possible binary combinations of orientations (6 combinations) and then we divide our training data according to the orientation of images. Using the segmented data we compute new training dataset with only images which are included in our binary classification pair. Then we compute a classifier using adaboost

technique for each binary combination of orientation. While testing we pass each image to all 6 classifiers and predict the orientation which is predicted by most classifiers. The classifier pairs are selected in such a way that each orientation is compared to all other orientations.

In Adaboost model we select random pair of pixels (range 0-191) we generate all possible such pairs and then randomly select 500 pairs to create our sample space. For each sample on sample space, if pixel1 is greater than pixel2 than we consider that test image to be positively classified else we consider that image as negatively classified. During training phase we calculate the number of images in each orientation classified positively and number of images in each orientation classified negatively. Depending upon this count we determine which orientation is tagged as positive and negative. Once we know the positive and negative orientation we replace each positive classified image with positive orientation and negative classified image as negative orientation. Using this new tagging we calculate error. If the calculated error is greater than 0.5 we discard the sample since when we modify the weights according to errors higher than 0.5 the weights of each sample reduce over time and become zero. Using the calculated error we modify the weights of correctly classified weights and then normalize these weights. We also calculate the weight of this sample using the error. For each sample we store the positive and negative orientation tags, the pixel pair and weight of each sample.

During testing phase we pass each image through all the classifiers which determine the orientation using the same comparison on same pixel values seed in training and we get a list of orientations. The orientation with highest frequency is selected as final orientation

Using this algorithm we get an accuracy in the range of 67 - 70%

No. Of Samples (decision stumps)	Training Time(s) (36,796 images)	Testing Time(s) (943)	Accuracy (%)
200	47	18	60
300	65	23	64
400	76	29	67
500	93	35	69

Classifier: KNN:

Training
KNN doesn't perform any work during training. The trained model contains the images and orientations as they are.

Testing
Method 1: Compare test image with average image of each orientation over all images from training data

We can consider these average images as centroids representing all images of corresponding orientation. If a test image is very close to some centroid, there is high probability that the image belongs to that orientation group. Using this method we get accuracy of 67.71%

Method 2: Compare test image with randomly chosen n images of each orientation from data images and have nearest-k vote on orientation

Since comparing test-image with each training image can be time consuming, compare it with few randomly chosen images.
Results for N random images and voting by K nearest images

N = 1,	K = 1,	Accuracy = 43.05%,	Time = 1.97 sec
N = 5,	K = 1,	Accuracy = 52.39%,	Time = 2.10 sec
N = 5,	K = 5,	Accuracy = 48.99%,	Time = 2.14 sec
N = 10,	K = 1,	Accuracy = 53.34%,	Time = 2.16 sec
N = 10,	K = 5,	Accuracy = 55.99%,	Time = 2.21 sec
N = 10,	K = 10,	Accuracy = 53.13%,	Time = 2.20 sec
N = 50,	K = 1,	Accuracy = 58.64%,	Time = 2.90 sec
N = 50,	K = 5,	Accuracy = 60.87%,	Time = 2.87 sec
N = 50,	K = 10,	Accuracy = 62.57%,	Time = 2.95 sec
N = 50,	K = 50,	Accuracy = 63.63%,	Time = 3.06 sec
N = 100,	K = 1,	Accuracy = 60.55%,	Time = 4.49 sec
N = 100,	K = 5,	Accuracy = 62.25%,	Time = 4.17 sec
N = 100,	K = 10,	Accuracy = 63.94%,	Time = 4.21 sec
N = 100,	K = 50,	Accuracy = 66.28%,	Time = 6.14 sec
N = 100,	K = 100,	Accuracy = 67.34%,	Time = 4.96 sec
N = 500,	K = 1,	Accuracy = 61.93%,	Time = 21.02 sec
N = 500,	K = 5,	Accuracy = 64.48%,	Time = 20.94 sec
N = 500,	K = 10,	Accuracy = 65.11%,	Time = 20.85 sec
N = 500,	K = 50,	Accuracy = 67.97%,	Time = 21.51 sec
N = 500,	K = 100,	Accuracy = 67.87%,	Time = 21.64 sec
N = 500,	K = 200,	Accuracy = 68.93%,	Time = 23.91 sec
N = 500,	K = 500,	Accuracy = 68.08%,	Time = 22.57 sec

N = 1000,	K = 1,	Accuracy = 62.46%,	Time = 40.51 sec
N = 1000,	K = 5,	Accuracy = 65.54%,	Time = 43.18 sec
N = 1000,	K = 10,	Accuracy = 66.28%,	Time = 41.07 sec
N = 1000,	K = 50,	Accuracy = 68.29%,	Time = 40.69 sec
N = 1000,	K = 100,	Accuracy = 67.66%,	Time = 43.06 sec
N = 1000,	K = 200,	Accuracy = 69.46%,	Time = 41.50 sec
N = 1000,	K = 500,	Accuracy = 68.40%,	Time = 45.92 sec
N = 1000,	K = 1000,	Accuracy = 68.19%,	Time = 67.91 sec
N = 5000,	K = 1,	Accuracy = 65.54%,	Time = 203.27 sec
N = 5000,	K = 5,	Accuracy = 65.64%,	Time = 209.10 sec
N = 5000,	K = 10,	Accuracy = 66.49%,	Time = 232.19 sec
N = 5000,	K = 50,	Accuracy = 67.87%,	Time = 228.11 sec
N = 5000,	K = 100,	Accuracy = 69.03%,	Time = 230.97 sec
N = 5000,	K = 200,	Accuracy = 69.88%,	Time = 231.00 sec
N = 5000,	K = 500,	Accuracy = 69.35%,	Time = 235.51 sec
N = 5000,	K = 1000,	Accuracy = 69.88%,	Time = 237.06 sec
N = 5000,	K = 5000,	Accuracy = 69.14%,	Time = 253.66 sec

Method 3: Compare test image with all the images from training data

This method can return result in reasonable amount of time after applying following 2 optimizations
1. Vectorized implementation
2. Using heap to find k smallest elements

K = 1,	Accuracy = 67.23%,	Time = 132.69 sec
K = 5,	Accuracy = 68.93%,	Time = 150.87 sec
K = 10,	Accuracy = 69.78%,	Time = 163.39 sec
K = 15,	Accuracy = 70.31%,	Time = 196.87 sec
K = 20,	Accuracy = 70.63%,	Time = 143.47 sec
K = 25,	Accuracy = 70.94%,	Time = 123.02 sec
K = 30,	Accuracy = 70.31%,	Time = 123.12 sec
K = 40,	**Accuracy = 71.58%,**	**Time = 121.28 sec**
K = 50,	Accuracy = 70.73%,	Time = 121.44 sec
K = 100,	Accuracy = 70.20%,	Time = 133.62 sec
K = 200,	Accuracy = 71.26%,	Time = 149.90 sec
K = 500,	Accuracy = 71.26%,	Time = 147.49 sec
K = 1000,	Accuracy = 70.41%,	Time = 164.94 sec
K = 5000,	Accuracy = 69.57%,	Time = 138.87 sec

Random sampling in Method 2 takes more time than comparing test image with all training images in optimized Method 3. Method 3 also provides higher accuracy. Therefore this method is used in final implementation of orient.py

Classifier: Random Forest:

Random forest is an ensemble classification technique which creates multiple random decision trees based on subset of data and makes final classification based on a generalized output of each decision tree.

How Program Works:

Training:

In the training phase random forest implement multiple decision trees by picking a random predicate and recursively picking subset with least entropy. We first train the model using train mode of orient.py where we provide the training data, the model to train in this case forest and output file name.
This mode will provide a model file output which has been trained using the training data. Our model is nothing but a list of all the decision tree which we have computed based on training data.
Command:
./orient.py train train-data.txt forest_model.txt forest

Testing:

To test the model on the test dataset we use the test mode of orient.py where we provide the test dataset, trained model and the model to be used for testing in this case forest.
This will generate an output.txt file which will contain predicted orientation of each test image provided in testing phase. It will also show accuracy of the model on console.
Command:
./orient.py test test-data.txt forest_model.txt forest

Algorithm:

Training: For training, we generate multiple random decision trees from training set, we found that at an average generating 200 random trees gives decent accuracy of 70%. Following is the algorithm for training model.

1. Pick a random sample of training data to generate a decision tree.
2. Generate a set of random predicates i.e. a pixel and a threshold.
   Decision trees are based on some parameters (called predicates) which in our program is a pixel value picked up at random. This value is one of the rgb values. Once the pixel is fixed, we pick a random threshold and divide our data into two sets.
3. Divide the data into two sets based on the predicate and calculate average entropy.
4. Repeat step 2,3 with several random pixels for the given dataset and pick the one that divides the data with least entropy. This become the root of the current subtree.
5. We keep dividing until we generate a dataset with minimal entropy and assign it the predominant orientation of the group. We stop diving further.
6. Recursively repeat steps 1-5 to create multiple decision trees.
7. Store all the trees in a list which is to be used for testing.

Testing: For testing, make decision based on all the cumulative result of the subtrees generated during training phase.

1. For the current image, start with the first decision tree.

2. Retrieve the current predicate which is nothing but a pixel position and it's threshold, if the pixel at this position, in our image is smaller than threshold, we move to the left child or else to the right child.
3. If this is a leaf node, we get the orientation.
4. Perform steps 2-3 for each decision tree which keep a count of orientation given by every tree.
5. Pick the orientation with maximum votes. This is the orientation for the current image.
6. Repeat steps 1-5 for each image in our testing dataset.

Using this algorithm, we get an accuracy in the range of 69 - 72%.

Sr no	Dataset size (Total/N)	No. of Trees	Training Time(minutes) (36,796 images)	Accuracy (%)
1	1350 (Total/30)	100	5	70.20
2	1350 (Total/30)	200	10	70.33
3	1000 (Total/40)	500	17	70.63
4	4000 (Total/10)	500	104	72.11

*Every test required no more than 5 seconds.

Recommended Parameters for optimal result:

- Dataset size: 1350 (Total/30)
- No. of Trees: 200

Classifier: Best:

With our testing we found that random forest gives the best results as compared to the rest.

How Program Works:

Training:
Command:
./orient.py train train-data.txt best_model.txt best

Testing:
Command:
./orient.py test test-data.txt best_model.txt best

*For further details on working of random forest, kindly refer the Random Forest section.