K-Nearest Neighbor

KNN algorithm finds k nearest neighbors for a new test data and then determines the class label of new data based on the class label of the k nearest points.

To determine the k points we have used Euclidean distance between points

Training

KNN training takes data as it is and copies to model.txt file

Test

In test phase, we first read the model file which has all the training data.

For each test data, we have calculated the distance with each of training data.

Distance is calculated by Euclidean distance between two points which is determined as –

Distance = $sqrt(\sum (X_i - Y_i)^2)$

The algorithm then finds k training data having minimum distance with test data and determines class of test by taking majority from k training data records.

Output –

KNN accuracy - 71.26193001060446

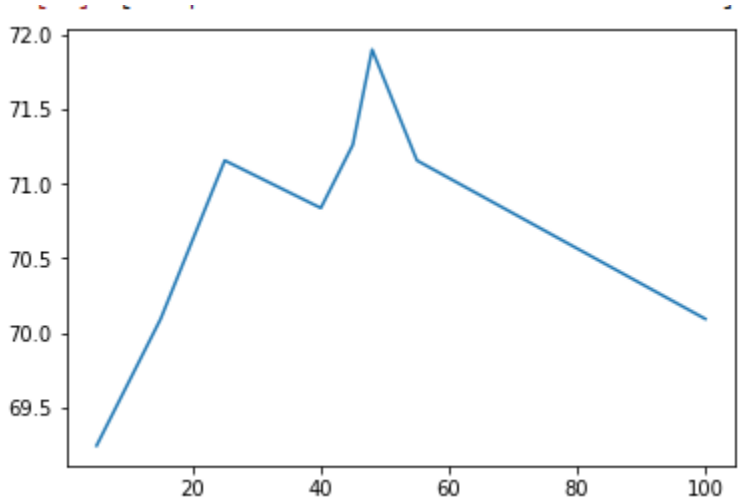
We tried to do some feature engineering. We reduced the features which are in the middle section of greed as it does not contribute much in finding the rotation.

If we consider an image as 8 * 8 matrix with each cell storing (R, G, B) pixel values, we dropped data from 4^{th} and 5^{th} row and 4^{th} and 5^{th} column.

This gave the test accuracy - 70.94379639448569 (For k = 45)

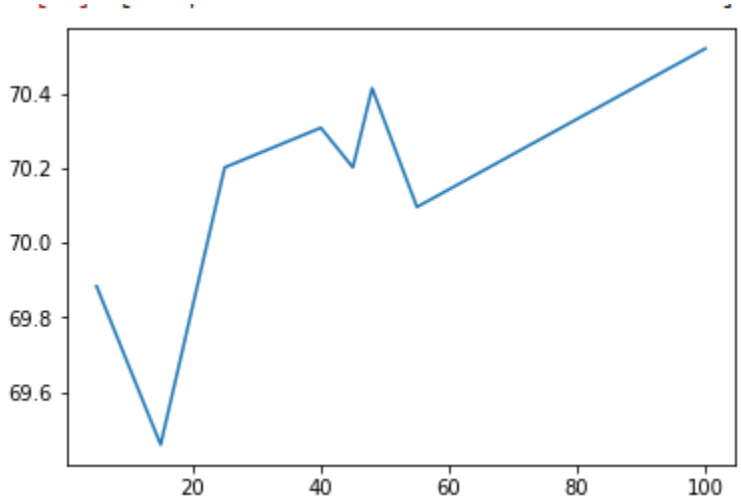
This tells that the middle pixels do not contribute much towards decision of rotation of image.

We also tuned k parameter by trying different values of k in loop and found the maximum accuracy for k value = 48 with accuracy - = 71.9



We also tried a manhattan distance for list of k values and got best accuracy of at k = 100 and got accuracy 70.25

Graph –



Performance

KNN works well with good distance function on if data do not have noise. It is simple and fast to train. But KNN can be slow and memory intensive for large data set. KNN has data overfitting as it memorizes all training data.

Running time for KNN vary depending upon number of features and training data as it calculates distance with each training set data for each test data.

Running time

Training in KNN is very simple as it is just copying all training data in model file.

Testing depends on number of features and rows in training data

Running time for k = 48 and all data set = 2.909 min

Running time for k = 5 and all data set = 2.812 min

Running time for k = 48 after feature engineering = 1.816 min

Running time for k = 5 after feature engineering = 1.759 min

Selecting parameter

To get the best parameter, we looped through some k values and found the best k value having max accuracy.

We found k = 48 and distance function Euclidean gave accuracy = 71.9

And k = 100 and distance function manhattan gave accuracy = 70.25

How does performance vary depending on the training dataset size, i.e. if you use just a fraction of the training data?

- We tried this algorithm with head(10000) and got accuracy of 70.45 for k =45
- 5000 records and found that we got 69.35 accuracy for k =45
- We also tried on top 1000 records and found that we got 67.02 accuracy with k =45
- We also tried on top 200 and got accuracy of 66.80 with k=45
- More the training data, more the time required to train.

Adabost

Working

In adaboost we create multiple weak learners and assign a weight for each learner such that we can use weighted average to determine class label.

Since we have 4 classes 0, 90, 180 and 270. We have created ${}^4C_2$ that is 6 combinations which comes as 0-90, 0-180, 0-270, 90-180, 90-270, 180-270.

We have defined weight vector which is list of n elements each with initial value 1/N where N is length of dataset under consideration.

For each pair of classes, we found 400 decision stumps that are randomly generated set of non-repeating pixels from 192 features. For each such hypothesis, we will use subset of training data with same classes as pair. Say we started with pair of 0-90 then we are considering training data with class label 0 and 90.

Based on selected hypothesis, we are finding if a class if correctly classified or not. If not correctly classified, we will find error as error = error + weight(at that index).

If the error value is more than 0.5, we discard that hypothesis as a bad decision stump.

If not bad decision stump, we have updated weight of correctly classified data as weight = weight * (error/(1-error))

In the end we have calculated alpha value for each hypothesis which is calculated as alpha = log((1-error)/error)

This is stored in dictionary which is dumped in model.txt file for further reference.

Testing

In your report, present neatly organized tables or graphs showing classification accuracies and running times as a function of the parameters, you choose.

Decision stumps	Train time	Test time	Accuracy
400	4 min 10 sec	0 min 2 sec	67.23
600	6 min 17 sec	0 min 3 sec	68.83
1000	10 min 10 sec	0 min 4 sec	68.9
5000	53 min 11 sec	0 min 12 sec	70.1

Which classifiers and which parameters would you recommend to a potential client?

- More number of decision stumps more is the accuracy till some level but it increases time required for testing.
- We are considering the 2 features with simple comparison instead we can do some feature engineering to come up with better decision stumps to increase accuracy.
- For current dataset, we would recommend to consider 600 stumps which in turn generates around 300 useful stumps for each pair of labels and gave accuracy of 68.8

How does performance vary depending on the training dataset size, i.e. if you use just a fraction of the training data?

- We tried this algorithm with head(10000) and 600 decision stumps and got accuracy of 66.808 which was not bad. 5000 records and found that we got 67.232 accuracy with 600 stumps
- We also tried on top 1000 records and found that we got 65.74 accuracy with 600 stumps
- We also tried on top 200 and got accuracy of 60.127 with 600 stumps
- More the training data, more the time required to train.

Random Forest

Working

In random forest, we first compute values for each feature for images in training dataset. After which we build multiple trees by diving the input file into k parts where k is number of trees. Each part is given as input to build a tree.

For building trees, we split input images based on threshold value decided for a threshold, such that information gain for that split is maximum. After which, for testing, we traverse through each tree in forest and get best possible answer based on maximum votes of trees.

Kindly look at inline comments for more details.

Features decided

We have created 24 features as a difference of summation of nth and (8-n)th row/col, believing that the difference of pixel values will be a lot different for the top section of image and for bottom section of image. For example, outdoor pictures usually have lighter shades at top and darker shades at bottom.

So, new features are as follows:

0: (sum of red pixel values of row 0 – sum of red pixel values of row 7)/25

1: (sum of red pixel values of col 0 - sum of red pixel values of col 7)/25

.

.

.

.

22: (sum of blue pixel values of row 3 - sum of blue pixel values of row 4)/25

23: (sum of blue pixel values of col 3 - sum of blue pixel values of col 4)/25

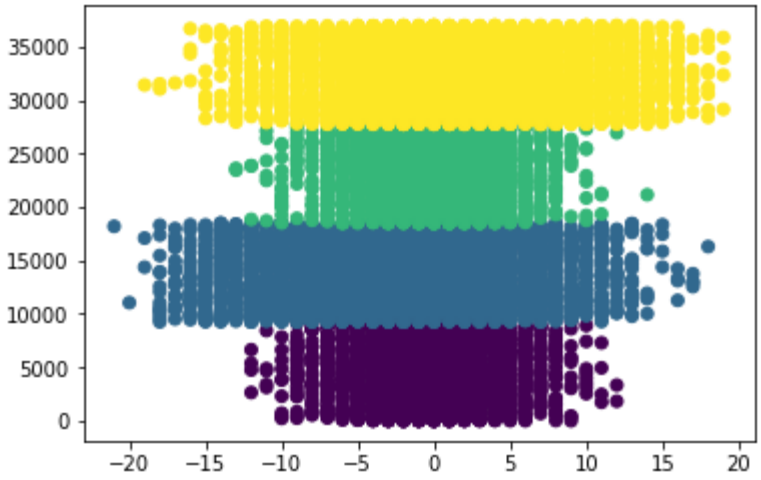
Deciding threshold value to split upon

We first thought of splitting the data as follows:

1. Compute means for each feature value of each set of images having same orientation.
2. Sort the means in ascending orders.
3. Take midpoints of two mean values as threshold

So, we expected the values of features to be distributed in non-overlapping chunks.

Unfortunately, the data was not distributed like that, even the best features had overlapping values, as shown below:



So, instead of calculating the threshold values, we decided to find the best possible threshold value for a feature by brute force. In which, checked entropy for each value of feature, and selected the combination of threshold and value for which information gain (entropy for parent – entropy for children) was maximum.

Parameters used:

1. Depth of tree: 15
2. Number of trees: 15

Performance Analysis based on parameters:

We observed that, the accuracy increased after increasing depth till 15. Also, we observed an increase in accuracy after increasing number of trees, when we fixed depth to 5. However, a combination of depth = 15 and number of trees = 15 gave better accuracy as compared to depth = 20 and number of trees = 20. All observations are noted down in table given below:

Depth	Number of trees	Accuracy	Time taken (hh:mm:ss)
5	5	64.8992576882	0:02:47.353752
5	10	65.4294803818	0:03:01.848570
5	15	65.1113467656	0:03:22.458398
5	20	65.7476139979	0:03:36.413551
5	100	67.7624602333	0:06:31.071917
10	5	69.2470837752	0:07:11.018453
10	10	68.2926829268	0:08:22.961371
10	15	69.9893955461	0:09:17.685002
10	20	69.0349946978	0:09:53.375758
15	5	68.8229056204	0:12:45.022811
15	**15**	**70.0954400848**	**0:14:36.968677**
15	20	68.6108165429	0:15:45.813437
20	20	69.4591728526	0:18:25.941448

Performance analysis based on number of images in training file:

We observed that, the accuracy decreased as we decreased the number of images used for training. All observations shown below consider number of trees 15 and depth 15.

Number of images	Accuracy	Time taken (hh:mm:ss)
4000	67.9745	35 sec
2000	67.7624	16 sec
400	65.25	7 sec
200	60.0	5 sec

Assumptions made for training file for Random forest

We have assumed that the input file has the same structure and same distribution of images i.e. each image is present 4 times with different orientations in sequence.

Image Prediction analysis

Random Forest:

Actual class	Predicted 0	Predicted 90	Predicted 180	Predicted 270
0 (total: 239)	173	17	31	18
90 (total: 224)	20	150	16	28
180 (total: 236)	38	23	156	19
270 (total: 244)	37	24	11	172

KNN

Actual class	Predicted 0	Predicted 90	Predicted 180	Predicted 270
0 (total: 239)	167	18	31	23
90 (total: 224)	21	159	14	30
180 (total: 236)	37	22	163	14
270 (total: 244)	26	33	15	170

Adaboost

Actual class	Predicted 0	Predicted 90	Predicted 180	Predicted 270
0 (total: 239)	177	14	26	22
90 (total: 224)	31	143	17	33
180 (total: 236)	35	22	158	21
270 (total: 244)	29	21	21	173

Show a few sample images that were classified correctly and incorrectly. Do you see any patterns to the errors?

When we searched for images that are classified correctly, we found that they were rich color images or having multiple objects to distinguish and predict rotation correctly.

Below images were classified by all classifiers –

When we searched for images that are misclassified, we found that they are mostly grey color images not with rich RGB and have a single object throughout and hence difficult to predict the rotation

Below images were misclassified by all classifiers –