Report

**K- Nearest Neighbors**

K -Nearest Neighbors is a discriminative classification algorithm. It is memory-based approach. The classifier immediately adapts as we collect new training data.

**Data Handling**

1. Checked for missing values in the data sets, if there are missing values, we can fill missing values by mean or median for the numeric features, and by mode for categorical values.

In the given data sets, we didn’t find any missing value.

1. Imported data files as Data Frame and converted the Data Frame to into Numpy Array for more convenience of using data and conversion in Numpy Array helped to decrease the time of execution.

**Steps for Implementation**

1. **Defined a function (def distance(x,y)) to calculate Euclidean Distance. Here, for each test data, this function calculates the distance with each training training data.**
2. **Defined a function (Function name:** **def knn(test\_data,train\_data,k)) to find the nearest neighbors by finding their index. The function takes train data , test data and k value as input and returns index of the k nearest neighbors as output.**
3. **Defined a function (Function name: def find\_max\_mode(list\_data)) to find mode of the labels of nearest neighbors (voting on correct orientation). This function takes labels of k nearest neighbors as input and produces output of the label having maximum frequency.**
4. **Defined a prediction function (Function name: def prediction (a,b,train\_label, k)) to predict the labels of the given test data. Here a – train data, b-test data, k – number of nearest neighbors).**
5. **Defined a function (Function name: def accuracy(predicted,actual)) to calculate accuracy of the model by comparing the predicted labels and actual test labels.**
6. **Calculated by maximum accuracy by tuning the hyperparameter (k – number of nearest neighbors) and plotted the graph of K-value vs Accuracy**

**Hyperparameter tuning**:

K- number of nearest neighbors

As we can observe from the graph that we are getting maximum accuracy for k = 41,

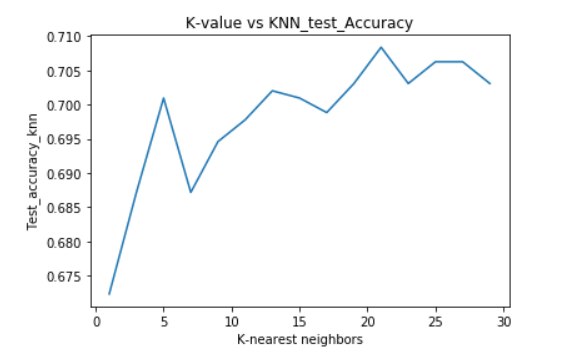
max accuracy =0.71898197242842

|  |  |
| --- | --- |
| **K value** | **Accuracy** |
| 1 | 0.672322375397667 |
| 3 | 0.6871686108165429 |
| 5 | 0.7009544008483564 |
| 7 | 0.6871686108165429 |
| 9 | 0.694591728525981 |
| 11 | 0.6977730646871686 |
| 13 | 0.7020148462354189 |
| 15 | 0.7009544008483564 |
| 17 | 0.6988335100742312 |
| 19 | 0.7030752916224814 |
| 21 | 0.7083775185577943 |
| 23 | 0.7030752916224814 |
| 25 | 0.7062566277836692 |
| 27 | 0.7062566277836692 |
| 29 | 0.7030752916224814 |
| 30 | 0.7051961823966065 |
| 40 | 0.7168610816542949 |
| 41 | 0.71898197242842 |
| 50 | 0.7051961823966065 |

Key points observed.

* Increasing the value of k after certain point would decrease the accuracy as increasing k-value leads to increase in bias.
* As the value of K increases variance decreases.
* In case of small value of k, algorithm is very sensitive to noise
* In case of large value of k, we may include points from other classes.

Below is the plot of K-value vs KNN-Test Accuracy



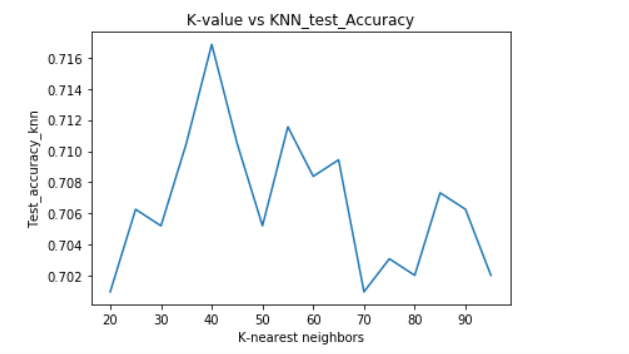
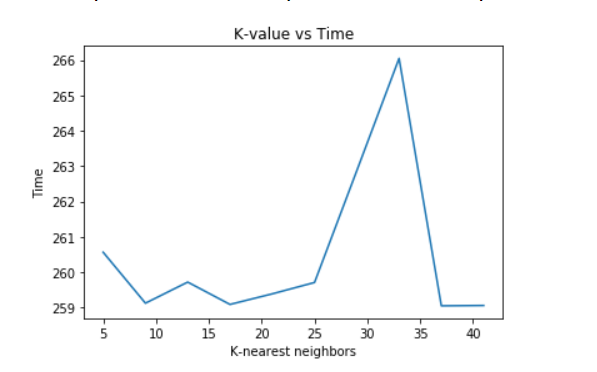


Table: Representing K-value and corresponding time elapsed.

|  |  |
| --- | --- |
| K-Value | Time Elapsed(secs) |
| 5 | 260.56389117240906 |
| 9 | 259.1237952709198 |
| 13 | 259.7204279899597 |
| 17 | 259.0888888835907 |
| 21 | 259.3880968093872 |
| 25 | 259.7094929218292 |
| 29 | 262.8705039024353 |
| 33 | 266.04530811309814 |
| 37 | 259.04996156692505 |
| 41 | 259.0589978694916 |



Observation:

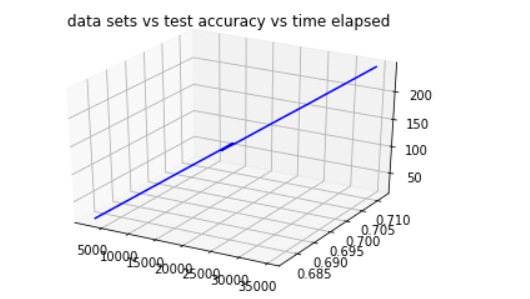
There is hardly any significant change in time duration due to change in value of K – Nearest Neighbours.

The time remains approximately same for different values of K.

Below is the table representing the Data Sets of different sizes and corresponding accuracy and time taken.

|  |  |  |
| --- | --- | --- |
| Data Sets | Accuracy | Time Elapsed(secs) |
| 2311 | 0.6818663838812301 | 15.907667875289917 |
| 6933 | 0.6998939554612937 | 48.200096130371094 |
| 11555 | 0.7073170731707317 | 81.05984258651733 |
| 16177 | 0.6967126193001061 | 114.3927230834961 |
| 20799 | 0.6935312831389183 | 147.56714153289795 |
| 25421 | 0.6988335100742312 | 180.90625643730164 |
| 30043 | 0.6988335100742312 | 214.36259841918945 |
| 34665 | 0.711558854718982 | 247.00882172584534 |
| 36976 | 0.71898 | 254.09025678332961 |

X-axis – Data sets, Y-axis-Test Accuracy, Z-axis – Time



**Observation**:

The Accuracy of the model increases as we increase the size of training data set. At the same, time we can also see that the running time increase as the size of training data set increases.

**Correctly Classified Images:**















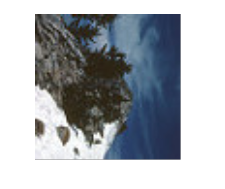




**Misclassified Images:**













**Pattern of Error**

From above samples of misclassified images, we can observe that, the model was not able to classify the images correctly which were taken in insufficient light as we can see, in above misclassified images, some images are taken in night, cloudy weather, dusk.

All the correctly classified images have been taken in sunlight or sufficient light. The images which are classified correctly are more clear.

**Random Forest**

Implementation of random forest:

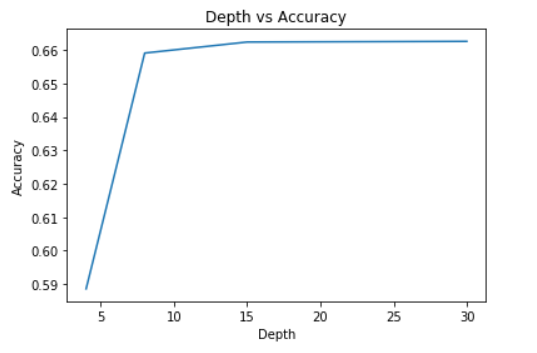
1. The main component of a random forest classifier is the decision tree.
2. In order to implement a decision we split iteratively using a greedy algorithm. At each node we find the best attribute to split on.
3. We handle the continuous splitting criteria by setting the threshold to half the value of the highest pixel. (128).
4. In order to the select the best splitting attribute we calculate the information gain of the split. Whichever split has the maximum information gain is selected.
5. Now, we create a forest of such decision trees. Each tree is trained on a subset of the training data selected randomly.
6. On testing each decision tree predicts and then the final decision is made by selecting the class which has the maximum votes.

The following hyperparameters have been tuned on silo:

**Depth** (for 50 trees):

Depth is a important tuning parameter for the random forest algorithm. As we increase the depth of the tree the decision trees of the forest because more complex and hence, the train error on them decreases.

|  |  |  |
| --- | --- | --- |
| **Depth** | **Accuracy** | **Training Time (in Mins)** |
| 4 | 0.5886 | 5 |
| 8 | 0.6591 | 14 |
| 15 | 0.6712 | 67 |
| 30 | 0.6626 | 203 |



**Trees (**for depth 15):

Trees vote and based on the votes a decision of classification is made. As, the number of trees increases the overfitting on the dataset descreases. This, is because multiple trees vote and the chances of all trees being biased is less.

|  |  |  |
| --- | --- | --- |
| **No of Trees** | **Accuracy** | **Training Time (in Mins)** |
| 1 | 0.6135 | 0.53 |
| 15 | 0.6324 | 15 |
| 30 | 0.6419 | 39 |
| 50 | 0.6712 | 67 |

**Split size** (For 50 trees and depth 15):

The trees are trained on subsets of data. Various size splits have been tried and the following results are obtained.

|  |  |  |
| --- | --- | --- |
| **Split Size** | **Accuracy** | **Training Time (in Mins)** |
| Random size | 0.6263 | 39 |
| 1/3 | 0.6129 | 10 |
| 1/2 | 0.6316 | 24 |
| 2/3 | 0.6712 | 67 |

**Entropy value for decision:**

The decision tree stops when the entropy value of a split is less than a threshold.

|  |  |
| --- | --- |
| **Value for entropy** | **Accuracy** |
| 0 | 0.6423 |
| 0.1 | 0.6470 |
| 0.2 | 0.6561 |
| 0.3 | 0.6502 |
| 0.4 | 0.6712 |
| 0.5 | 0.6497 |

The following hyper-parameters prove to be the best based on the tuning performed:

Depth = 15

Number of Trees = 50

Split size = ⅔

Entropy for decision = 0.4

**Correctly Classified Images by Random Forest**

**Incorrect Classified Images**



**Adaboost**

* To solve the multi-class problem, we converted this into a one vs all problem. That is (0 vs 90), (0 vs 180), (0 vs 270), (90 vs 180), (90 vs 270), and (180 vs 270).
* This generated 6 sets of training data.
* For each training set, we take a majority vote. If majority vote for a class then that class is assigned Positive otherwise Negative.
* For each training set, we take a majority vote. If majority vote for a class then that class is assigned Positive otherwise Negative.
* -Initially the weights are initialized as 1/N. Whenever we get the correct answer we decrease the initialized weights otherwise we increase the weights of the incorrectly classified example. Normalization is then done.
* +Initially the weights are initialized as 1/N. Whenever we get the correct answer we decrease the initialized weights otherwise we increase the weights of the incorrectly classified example. Increase is done by adding the error.
* +Normalization is then done.
* An accuracy ranging from 68-71% was achieved. Even if we increased the number of random pixel pairs compared, the accuracy achieved didn’t increase by more than 71% but at a much higher computation cost.
* An accuracy ranging from 68-71% was achieved. Even if we increased the number of random pixel pairs compared, the accuracy achieved didn’t increase by more than 71% but at a much higher computation cost.

**Hyperparameter Tuning**

**Here the Hyperparameter used is Pixel Pairs**

|  |  |
| --- | --- |
| **Pixel Pairs** | **Accuracy** |
| 10 | 47-55% |
| 100 | 56-62% |
| 500 | 64-68% |
| 1000 | 68-71% |

**Selection of Model:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Model** | **Accuracy** | **Training Time (in secs)** | **Testing Time (in secs)** |
| KNN | 71.898 | 0 | 240-381 |
| Random Forest | 67.12 | 4020 | 3-7 |
| AdaBoost | 69 | 30 | 2-6 |

**Selection of Model is the one of the important steps.**

**On the given data sets, we can observe that the performance of all three models, stated in the above table. KNN is the model with highest accuracy but it consumes more time than other models to output the results. So, if we have to recommend to the customer the model among the above models, we need to clarify upon their need, if they strictly want a model with the best accuracy, without considering time taken, then definitely KNN would be the first choice of recommendation. On the other hand, if the customer is more stringent on time duration, then AdaBoost would be the best choice.**