CSCI-B 551 Elements of Artificial Intelligence

Report for Assignment 5

Problem Statement: To create a classifier that decides the correct orientation of a given image. The orientations are 0º, 90º, 180º, and 270º.

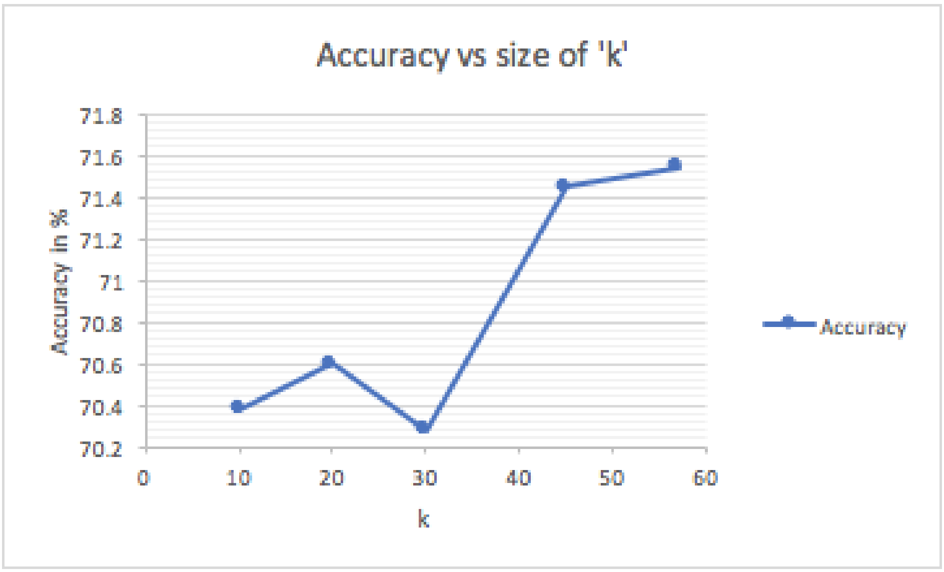
Data: 10,000 images rotated in each of the given orientation makes up 40,000 images as our training data set. The test data set includes 1000 images.:

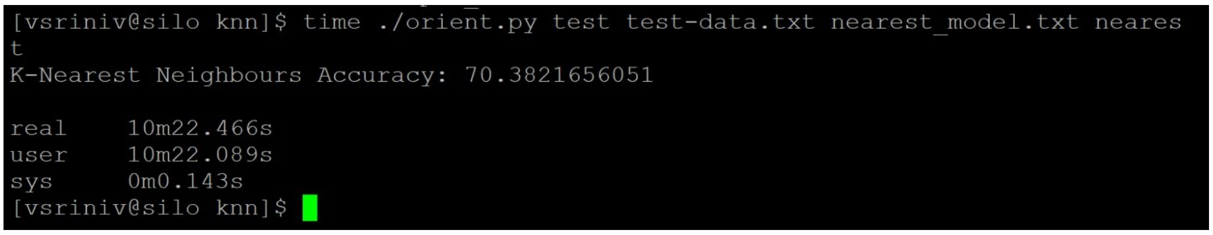
Algorithms Used: As part of the assignment we were asked to implement 3 Machine Learning Algorithms- k-nn, adaboost, and neural net

1. Nearest Neighbor:

K-nearest neighbor algorithm is a variant of nearest neighbor. We used Euclidean distance to find the k-nearest neighbors and assigned each image an orientation based on the majority number of votes. We stopped at k = 57 because we saw no improvement after that. We are reading the training file and storing the orientation along with RGB vector list to the model file [nearest_model.txt]. We are then using this model file for the Euclidean distance calculation against test data for orientation prediction.

k	**Accuracy**	**Time (min.sec)**
10	70.38	10.23
20	70.60	10.68
30	70.28	11.17
45	71.45	9.50
57	71.54	11.12

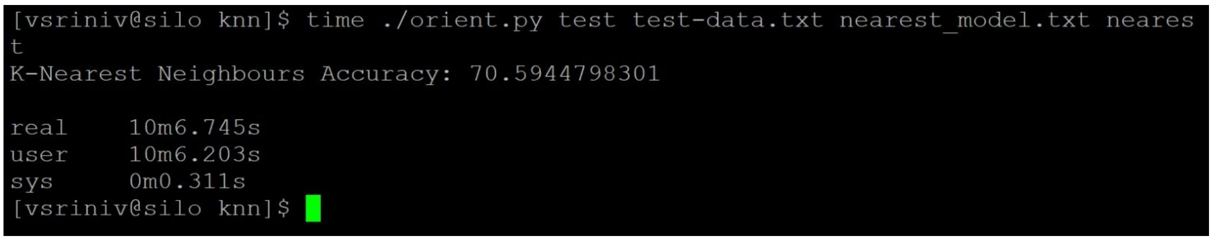




```
[vsriniv@silo knn]$ time ./orient.py test test-data.txt nearest_model.txt neares
t
K-Nearest Neighbours Accuracy: 70.3821656051

real    10m22.466s
user    10m22.089s
sys     0m0.143s
[vsriniv@silo knn]$ 
```

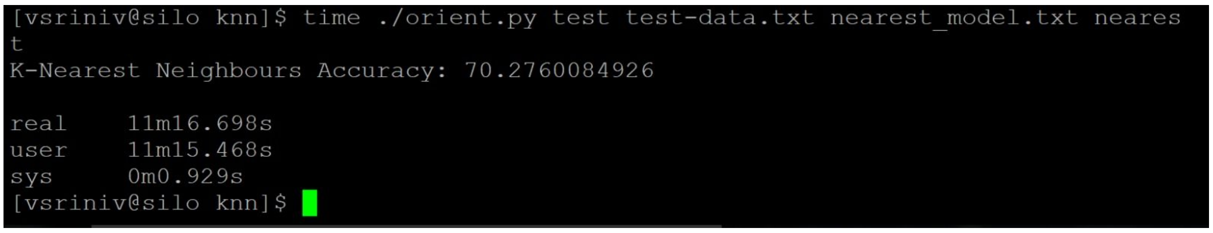
KNN [k=10]



```
[vsriniv@silo knn]$ time ./orient.py test test-data.txt nearest_model.txt neares
t
K-Nearest Neighbours Accuracy: 70.5944798301

real    10m6.745s
user    10m6.203s
sys     0m0.311s
[vsriniv@silo knn]$ 
```

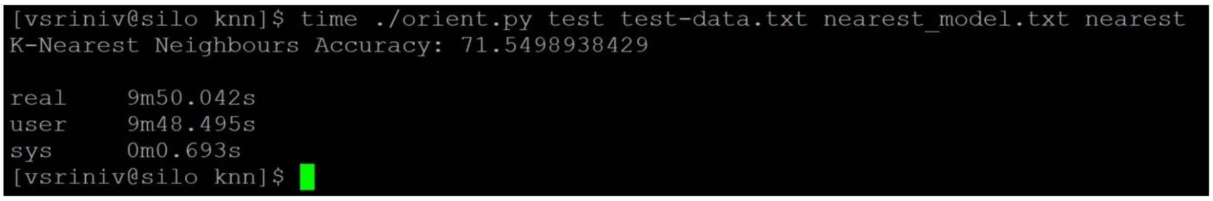
KNN [k=20]



```
[vsriniv@silo knn]$ time ./orient.py test test-data.txt nearest_model.txt neares
t
K-Nearest Neighbours Accuracy: 70.2760084926

real    11m16.698s
user    11m15.468s
sys     0m0.929s
[vsriniv@silo knn]$ 
```

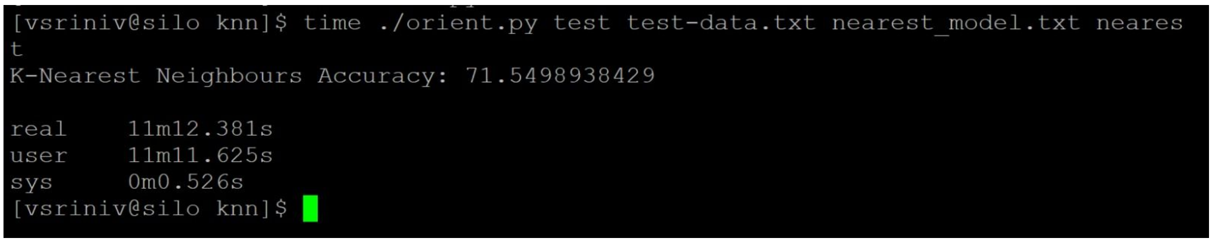
KNN [k=30]



```
[vsriniv@silo knn]$ time ./orient.py test test-data.txt nearest_model.txt nearest
K-Nearest Neighbours Accuracy: 71.5498938429

real    9m50.042s
user    9m48.495s
sys     0m0.693s
[vsriniv@silo knn]$ 
```

KNN [k=45]



```
[vsriniv@silo knn]$ time ./orient.py test test-data.txt nearest_model.txt neares
t
K-Nearest Neighbours Accuracy: 71.5498938429

real    11m12.381s
user    11m11.625s
sys     0m0.526s
[vsriniv@silo knn]$ 
```

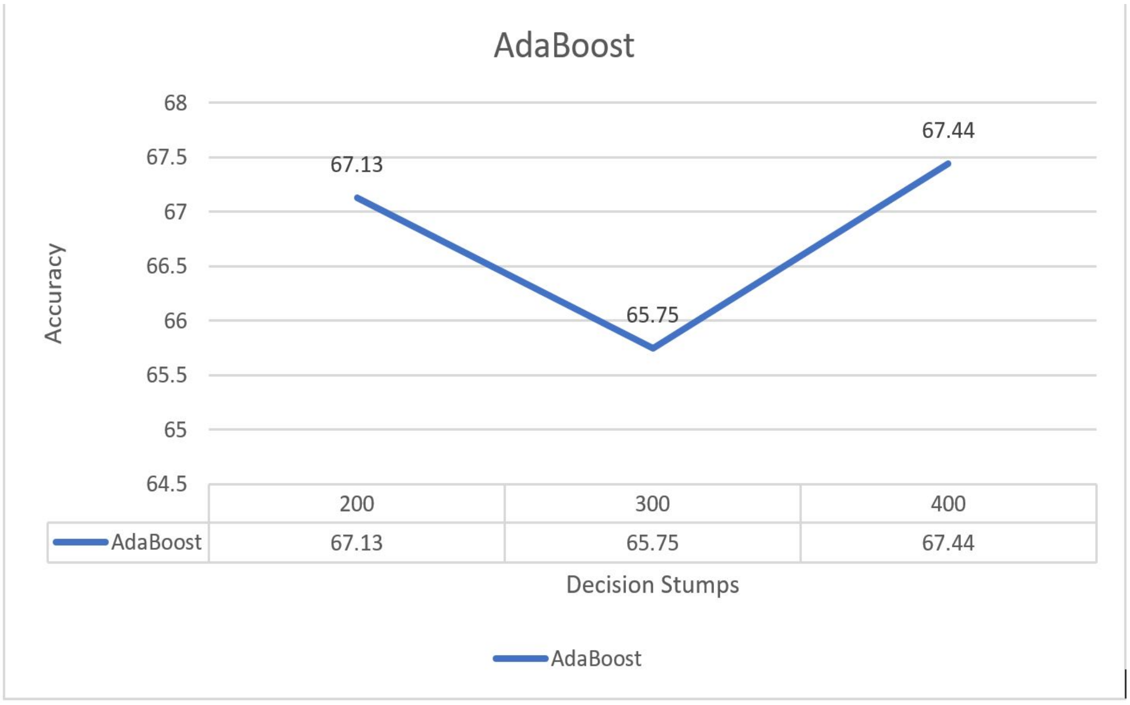
KNN [k=57]

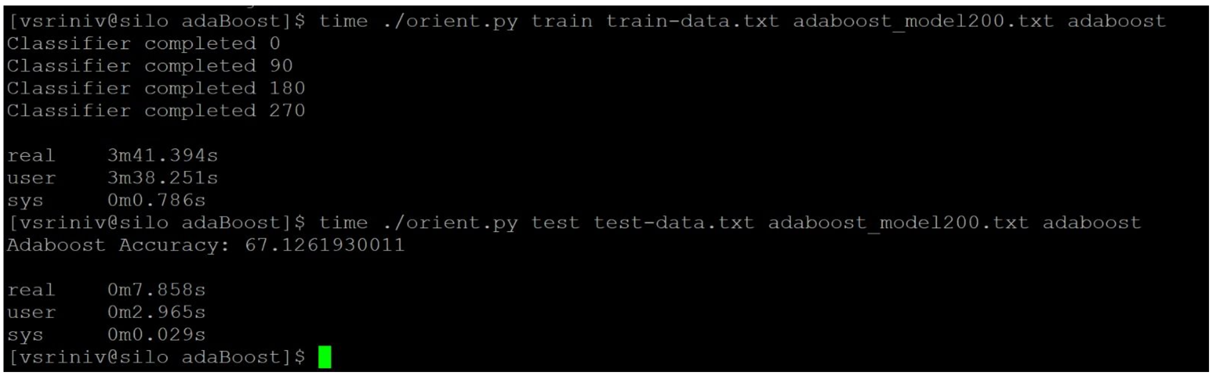
2. **Adaboost:**

Adaboost is a combination of weak classifiers to produce one strong classifier. We used the method in this tutorial - http://mccormickml.com/2013/12/13/adaboost-tutorial/ and https://stackoverflow.com/questions/12097155/weak-classifier. We have four classifiers of 1 versus All, 0º versus [90º, 180º, 270º], 90º versus [0º, 180º, 270º] 180º versus [0º, 90º, 270º] and 270º versus [0º, 90º, 180º], such that we set 1 for 0º and 0 for all other orientation. We used decision stumps as our weak classifier for training. We considered only certain features for each classifier.

We are reading the training file and storing the orientation along with classifier and alpha value in the model file [adaboost_model.txt]. We are then using this model file for the orientation prediction.

Stumps	**Accuracy**	**Run time (mins)**
200	67.13	3
300	65.75	5
400	67.44	12



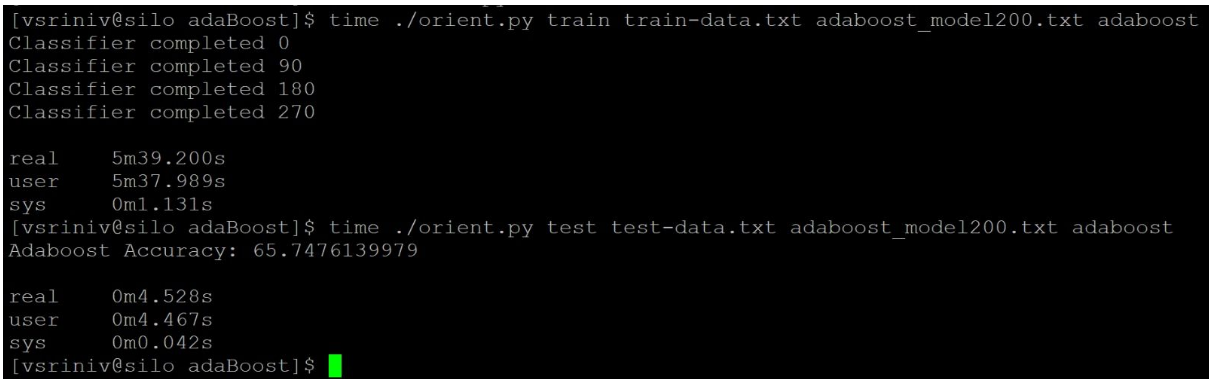


```
[vsriniv@silo adaBoost]$ time ./orient.py train train-data.txt adaboost_model200.txt adaboost
Classifier completed 0
Classifier completed 90
Classifier completed 180
Classifier completed 270

real    3m41.394s
user    3m38.251s
sys     0m0.786s
[vsriniv@silo adaBoost]$ time ./orient.py test test-data.txt adaboost_model200.txt adaboost
Adaboost Accuracy: 67.1261930011

real    0m7.858s
user    0m2.965s
sys     0m0.029s
[vsriniv@silo adaBoost]$
```

AdaBoost [Decision Stumps = 200]

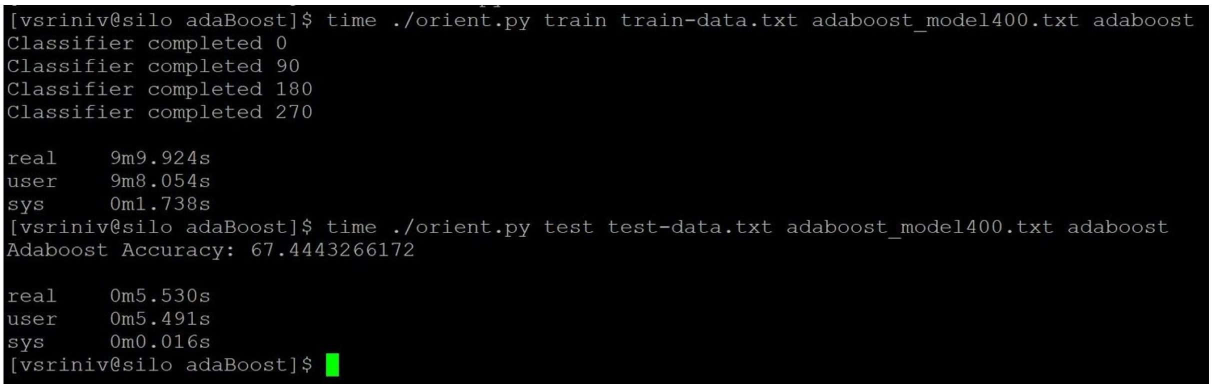


```
[vsriniv@silo adaBoost]$ time ./orient.py train train-data.txt adaboost_model200.txt adaboost
Classifier completed 0
Classifier completed 90
Classifier completed 180
Classifier completed 270

real    5m39.200s
user    5m37.989s
sys     0m1.131s
[vsriniv@silo adaBoost]$ time ./orient.py test test-data.txt adaboost_model200.txt adaboost
Adaboost Accuracy: 65.7476139979

real    0m4.528s
user    0m4.467s
sys     0m0.042s
[vsriniv@silo adaBoost]$
```

AdaBoost [Decision Stumps = 300]



```
[vsriniv@silo adaBoost]$ time ./orient.py train train-data.txt adaboost_model400.txt adaboost
Classifier completed 0
Classifier completed 90
Classifier completed 180
Classifier completed 270

real    9m9.924s
user    9m8.054s
sys     0m1.738s
[vsriniv@silo adaBoost]$ time ./orient.py test test-data.txt adaboost_model400.txt adaboost
Adaboost Accuracy: 67.4443266172

real    0m5.530s
user    0m5.491s
sys     0m0.016s
[vsriniv@silo adaBoost]$
```

AdaBoost [Decision Stumps = 400]

3. <u>Neural Nets:</u>

We modified parameters at random based on our intuitions, say bias, modifying the number of nodes in the hidden layer, etc.

Our model includes one hidden layer with 90 nodes. We achieved best performance of our code with an accuracy of 71.05% having 90 hidden nodes. The parameters and design decisions are listed below.

a. Method: Stochastic Gradient Descent
b. Activation Function: Sigmoid
c. Learning rate: 0.0001
d. Iterations: 60
e. Output Nodes: 4
f. Input Nodes: 192

We are using numpy matrices to maintain the weights, one matrix to store weight between input and hidden and one for weight between hidden and output. Then we numpy's dot function for matrix multiplication and have implemented a sigmoid function for activation with a flag to indicate the derivative for the gradient. For the stochastic method we pick a random vector set from the data set. We calculate in feed forward phase with sigmoid activation. In the back propagation phase we calculate the gradient using sigmoid function with derivative set to TRUE. We also calculate the error and the weights using the formulas discussed in class. For storing the model we use the numpy method - savez_compreressed; which saves multiple numpy matrices as a single file and hence the model file is named with a ***.npz extension**, named as ***nnet_model.npz.***

Performance as measured in my laptop.

Hidden Nodes	**Learning Rate**	**Iterations**	**Accuracy**	**Time (approx.)**
30	0.01	100	64.26	24 mins
90	0.0001	350	71.05	2 hour 30 mins **(On Tank Server)**
20	0.0001	400	69.24	1 hour
20	0.0001	600	69.35	1 hour 20 mins
20	0.0001	900	70.2	2 hour
20	0.0001	1000	71.04	2 hour 40 mins

4. <u>Best:</u>

Neural network is our best model as it has achieved the highest performance. The model file is named as ***nnet_model.npz.*** The method used for neural net implementation is discussed above.