ASSIGNMENT 2 - k Nearest Neighbors algorithm

-Srividhya Chandrasekharan (chandrasekharan.12)

INTRODUCTION:

In this assignment, kNN algorithm was implemented using assignment#1's code as starter code. Assignment#1 was about the process of finding 'k' data records that are closest or most similar to a given example. kNN algorithm takes as input - the k closest training examples in the feature space and outputs a class membership.

Training Datasets:- Iris.csv and Income.csv

Test Datasets:- Iris_Test.csv and Income_Test.csv

Output files:- output.csv and knn.csv

SECTION 1:

kNN is a lazy learning algorithm. It simply stores the training dataset after preprocessing and waits until it is given a test record to classify. Hence, this algorithm takes 0 time in training and a lot of time in predicting!

Here's how the training data is preprocessed:-

Handling missing data:

- Income dataset had 33 missing values. This was indicated by '?'.
- Iris dataset did not have any missing values.
- Instead of deleting records with incomplete data, reasonable values for attributes missing data were computed.
- When the value of a categorical attribute was missing, the '?' was replaced by the most frequently occurring value of that attribute. The mode is the value that appears most often in a set of data.
- The mean of an attribute was assigned in place of '?' for numeric attributes.

Normalization of data:

- Numeric attributes were normalized using min-max normalization so as to bring all the values to a common scale [0,1].

$$v' = \frac{v - min}{max - min}\ (max_{new} - min_{new})\ + min_{new}$$

- By normalizing attributes, we make sure that different attributes contribute equally to the distance measures.

Training data was used to compute several parameters using which the test dataset was preprocessed. This is done in order to ensure that all the data is put through the same pipeline. The mean of each numerical attribute in the training dataset were calculated and stored as an array of tuples. Similarly, the most probable category(mode) were figured out for each categorical attribute in the training dataset. Also, the maximum and minimum values of each numeric attribute in the training dataset were found and stored for future usage.

Handling missing values of test dataset:

- When the value of a categorical attribute in the test dataset was missing, the '?' was replaced by that attribute's most frequently occurring value in the training dataset.
- The mean of the training dataset numeric attributes was assigned in place of '?' for test data's missing attribute values.

Normalizing test data:

- The numeric attribute values in the test data were mapped to values in the range [0,1] using the maximum and minimum values of each attribute in the training dataset.

Note:- No transformations were applied on the training and test datasets.

TRAINING PHASE

➔ Training dataset preprocessing is done (i.e) missing data values are imputed and numeric values are normalized. Nothing more is done.

TESTING PHASE

➔ For each test data record, similarity scores(either euclidean distance or cosine similarity) are calculated between that test record and each of the training dataset examples. While calculating similarity scores, the Transaction ID and class label attributes are dropped.

➔ Based on the proximity measures calculated above, the top 'k' neighbours were chosen. (i.e) the training data records closely similar to the test record are selected.

➔ A uniform majority voting is done to figure out the most probable class among the test record's k nearest neighbors. Each of the k neighbours are given equal weight and treated uniformly. This most probable class label is given out as the class label prediction for the given test record.

➔ Posterior probability is computed:-

$$Posterior\ Probability = \frac{no.\ of\ neighbours\ having\ class\ label == predicted\ class\ label}{total\ number\ of\ neighbors}$$

Distance measures used:

- Two distance measures were used in the program - Euclidean and Cosine similarity.

Computing similarity:

- Program has a variable **k -** which is used to set the number of closest number of records to be returned for the given example.
- Equal weight is assigned to numeric and categorical attributes.
- For each data record in the test dataset, similarity measures are calculated with all the other records in the training dataset. The top **k** records are selected and written into output.csv file.
- Each data record analyzed is split into a list of numerical and categorical values and these lists are used for comparisons.
- The list of numeric attributes are used to calculate numeric score (distance).
- Each of the categorical attribute values of the 2 records in hand are individually compared value by value.
- The numerical and categorical scores are combined together based on the weights and the similarity measure is calculated.

Assumption:

We assume that the training data is extensive enough and contains several data records for all possible combinations of real world attribute values. The more the training data, the better it is!

SECTION 2 - PERFORMANCE OF CLASSIFIER

This section captures the performance of the kNN classifier on Income and Iris test datasets for different parameter values used by the algorithm.

Parameters of kNN algorithm are:

1. Number of neighbors - k
2. Proximity measure used - {euclidean distance or cosine similarity}

Section(2.A):

- Following are the confusion matrices along with the classification and error rates of **Iris** dataset using **Euclidean distance** as the similarity measure:

For k=3:-

	Iris-setosa	Iris-versicolor	Iris-virginica
Iris-setosa	1.0	0.000000	0.000000
Iris-versicolor	0.0	0.733333	0.266667
Iris-virginica	0.0	0.200000	0.800000

CLASSIFICATION RATE = 0.8285714285714286

ERROR RATE = 0.17142857142857137

For k=5:-

	Iris-setosa	Iris-versicolor	Iris-virginica
Iris-setosa	1.0	0.00	0.00
Iris-versicolor	0.0	0.70	0.30
Iris-virginica	0.0	0.15	0.85

CLASSIFICATION RATE = 0.8285714285714286
ERROR RATE = 0.17142857142857137

For k= 10:-

	Iris-setosa	Iris-versicolor	Iris-virginica
Iris-setosa	1.0	0.000000	0.000000
Iris-versicolor	0.0	0.666667	0.333333
Iris-virginica	0.0	0.150000	0.850000

CLASSIFICATION RATE = 0.8142857142857143
ERROR RATE = 0.18571428571428572

For k= 15:-

	Iris-setosa	Iris-versicolor	Iris-virginica
Iris-setosa	1.0	0.000000	0.000000
Iris-versicolor	0.0	0.633333	0.366667
Iris-virginica	0.0	0.000000	1.000000

CLASSIFICATION RATE = 0.8428571428571429
ERROR RATE = 0.15714285714285714

- Following are the confusion matrices along with the classification and error rates of **Iris** dataset using **Cosine similarity** as the similarity measure:

For k=3:-

	Iris-setosa	Iris-versicolor	Iris-virginica
Iris-setosa	1.0	0.000000	0.000000
Iris-versicolor	0.0	0.466667	0.533333
Iris-virginica	0.0	0.500000	0.500000

CLASSIFICATION RATE = 0.6285714285714286
ERROR RATE = 0.37142857142857144

For k=5:-
CLASSIFICATION RATE = 0.6
ERROR RATE = 0.4

	Iris-setosa	Iris-versicolor	Iris-virginica

Iris-setosa	1.0	0.000000	0.000000
Iris-versicolor	0.0	0.366667	0.633333
Iris-virginica	0.0	0.450000	0.550000

For k=10:-

CLASSIFICATION RATE = 0.6285714285714286
ERROR RATE = 0.37142857142857144

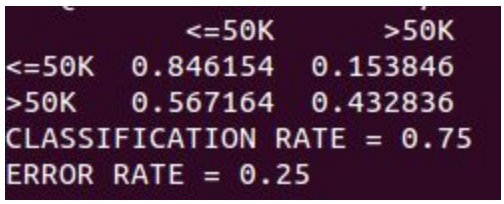
	Iris-setosa	Iris-versicolor	Iris-virginica
Iris-setosa	1.0	0.0	0.0
Iris-versicolor	0.0	0.4	0.6
Iris-virginica	0.0	0.4	0.6

For k=15:-

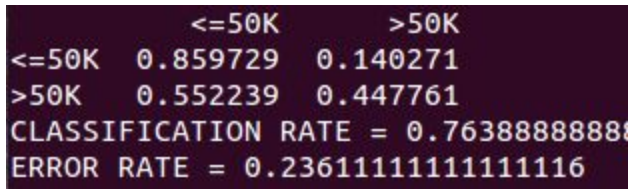
	Iris-setosa	Iris-versicolor	Iris-virginica
Iris-setosa	1.0	0.000000	0.000000
Iris-versicolor	0.0	0.433333	0.566667
Iris-virginica	0.0	0.300000	0.70000

CLASSIFICATION RATE = 0.6714285714285714
ERROR RATE = 0.3285714285714286

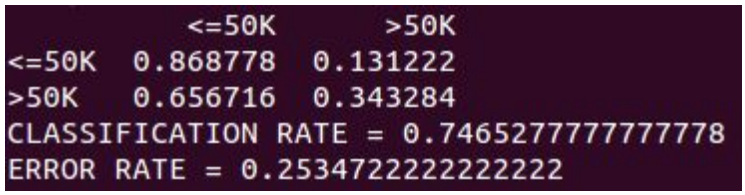
- Following are the confusion matrices along with the classification and error rates of **Income** dataset using **Euclidean distance** as the similarity measure:



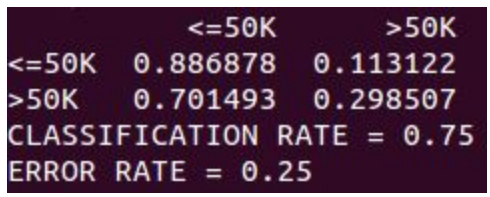
k=3



k=5



k=10



k=15

- Following are the confusion matrices along with the classification and error rates of **Income** dataset using **Cosine distance** as the similarity measure:

k=3

```
          <=50K      >50K
<=50K  0.959276  0.040724
>50K   0.820896  0.179104
CLASSIFICATION RATE = 0.7777777777777778
ERROR RATE = 0.2222222222222222
```

k=5

```
          <=50K      >50K
<=50K  0.968326  0.031674
>50K   0.880597  0.119403
CLASSIFICATION RATE = 0.7708333333333334
ERROR RATE = 0.22916666666666663
```

k=10

```
          <=50K      >50K
<=50K  0.977376  0.022624
>50K   0.895522  0.104478
CLASSIFICATION RATE = 0.7743055555555556
ERROR RATE = 0.22569444444444442
```

k=15

```
          <=50K      >50K
<=50K  0.977376  0.022624
>50K   0.910448  0.089552
CLASSIFICATION RATE = 0.7708333333333334
ERROR RATE = 0.22916666666666663
```

Section (2.B):

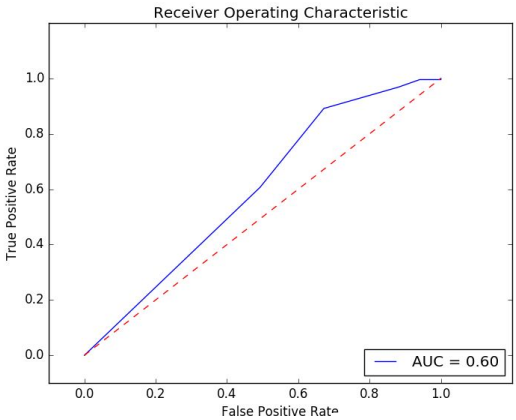
This section contains the True Positive, False Positive, True Negative and False Negative rates, Recall, Precision and F-Measure, as well as the ROC curve of Income dataset for different values of 'k'.

Using Cosine Similarity:

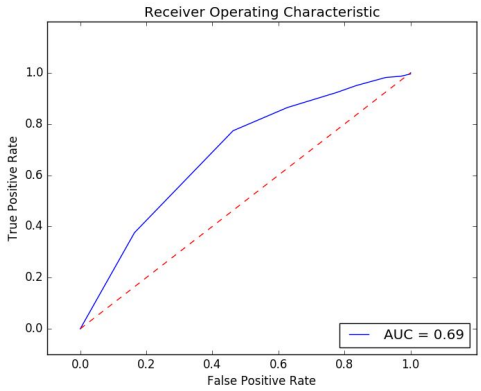
```
sri@sri-Lenovo-G50-70:~/Desktop/sem2/data mining/HW2$ python new_knn.py Income.csv 3
          <=50K      >50K
<=50K  0.959276  0.040724
>50K   0.820896  0.179104
CLASSIFICATION RATE = 0.7777777777777778
ERROR RATE = 0.2222222222222222
TRUE POSITIVE RATE = 0.9592760180995475
FALSE POSITIVE RATE = 0.8208955223880597
TRUE NEGATIVE RATE = 0.1791044776119403
FALSE NEGATIVE RATE = 0.04072398190045249
PRECISION = 0.7940074906367042
RECALL = 0.9592760180995475
F-MEASURE = 0.8688524590163935
sri@sri-Lenovo-G50-70:~/Desktop/sem2/data mining/HW2$ python new_knn.py Income.csv 5
          <=50K      >50K
<=50K  0.968326  0.031674
>50K   0.880597  0.119403
CLASSIFICATION RATE = 0.7708333333333334
ERROR RATE = 0.22916666666666663
TRUE POSITIVE RATE = 0.9683257918552036
FALSE POSITIVE RATE = 0.8805970149253731
TRUE NEGATIVE RATE = 0.11940298507462686
FALSE NEGATIVE RATE = 0.03167420814479638
PRECISION = 0.7838827838827839
RECALL = 0.9683257918552036
F-MEASURE = 0.8663967611336032
```

The above figure shows the parameter values got by running kNN algorithm on Income dataset for k=3 and k=5 using cosine similarity.

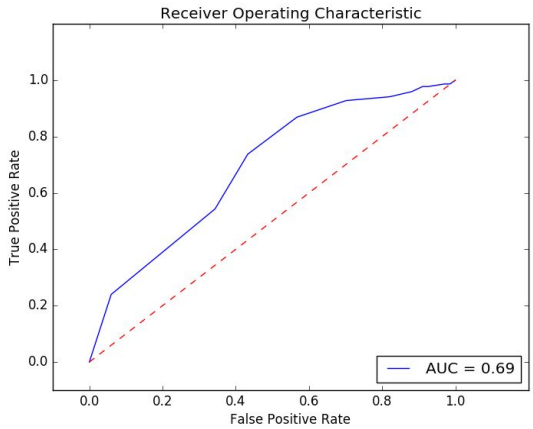
ROC curves for k=5, k=10 and k=15



k=5



k=10



k=15

```
sri@sri-Lenovo-G50-70:~/Desktop/sem2/data mining/HW2$ python new_knn.py Income.csv 10
          <=50K      >50K
<=50K  0.977376  0.022624
>50K   0.895522  0.104478
CLASSIFICATION RATE = 0.7743055555555556
ERROR RATE = 0.22569444444444442
TRUE POSITIVE RATE = 0.9773755656108597
FALSE POSITIVE RATE = 0.8955223880597015
TRUE NEGATIVE RATE = 0.1044776119402985
FALSE NEGATIVE RATE = 0.02262443438914027
PRECISION = 0.782608695652174
RECALL = 0.9773755656108597
F-MEASURE = 0.869215291750503
sri@sri-Lenovo-G50-70:~/Desktop/sem2/data mining/HW2$ python new_knn.py Income.csv 15
          <=50K      >50K
<=50K  0.977376  0.022624
>50K   0.910448  0.089552
CLASSIFICATION RATE = 0.7708333333333334
ERROR RATE = 0.22916666666666663
TRUE POSITIVE RATE = 0.9773755656108597
FALSE POSITIVE RATE = 0.9104477611940298
TRUE NEGATIVE RATE = 0.08955223880597014
FALSE NEGATIVE RATE = 0.02262443438914027
PRECISION = 0.779783393501805
RECALL = 0.9773755656108597
F-MEASURE = 0.8674698795180723
```

The above figure shows the parameter values got by running kNN algorithm on Income dataset for k=10 and k=15 using Cosine similarity.

Using Euclidean distance:-

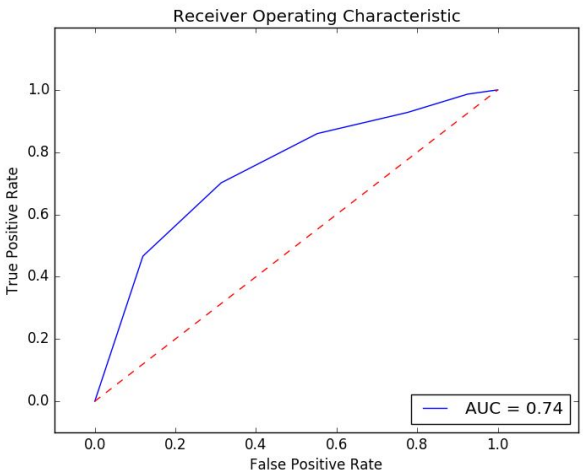
```
sri@sri-Lenovo-G50-70:~/Desktop/sem2/data mining/HW2$ python new_knn.py Income.csv 3
          <=50K      >50K
<=50K  0.846154  0.153846
>50K   0.567164  0.432836
CLASSIFICATION RATE = 0.75
ERROR RATE = 0.25
TRUE POSITIVE RATE = 0.8461538461538461
FALSE POSITIVE RATE = 0.5671641791044776
TRUE NEGATIVE RATE = 0.43283582089552236
FALSE NEGATIVE RATE = 0.15384615384615385
PRECISION = 0.8311111111111111
RECALL = 0.8461538461538461
F-MEASURE = 0.8385650224215248
sri@sri-Lenovo-G50-70:~/Desktop/sem2/data mining/HW2$ python new_knn.py Income.csv 5
          <=50K      >50K
<=50K  0.859729  0.140271
>50K   0.552239  0.447761
CLASSIFICATION RATE = 0.7638888888888888
ERROR RATE = 0.23611111111111116
TRUE POSITIVE RATE = 0.8597285067873304
FALSE POSITIVE RATE = 0.5522388059701493
TRUE NEGATIVE RATE = 0.44776119402985076
FALSE NEGATIVE RATE = 0.14027149321266968
PRECISION = 0.8370044052863436
RECALL = 0.8597285067873304
F-MEASURE = 0.8482142857142858
```

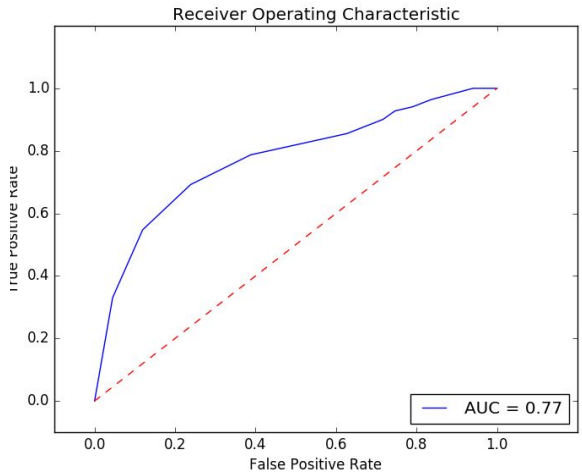
The above figure shows the parameter values got by running kNN algorithm on Income dataset for k=3 and k=5 using Euclidean distance similarity.

```
sri@sri-Lenovo-G50-70:~/Desktop/sem2/data mining/HW2$ python new_knn.py Income.csv 10
          <=50K      >50K
<=50K  0.868778  0.131222
>50K   0.656716  0.343284
CLASSIFICATION RATE = 0.7465277777777778
ERROR RATE = 0.2534722222222222
TRUE POSITIVE RATE = 0.8687782805429864
FALSE POSITIVE RATE = 0.6567164179104478
TRUE NEGATIVE RATE = 0.34328358208955223
FALSE NEGATIVE RATE = 0.13122171945701358
PRECISION = 0.8135593220338984
RECALL = 0.8687782805429864
F-MEASURE = 0.8402625820568929
sri@sri-Lenovo-G50-70:~/Desktop/sem2/data mining/HW2$ python new_knn.py Income.csv 15
          <=50K      >50K
<=50K  0.886878  0.113122
>50K   0.701493  0.298507
CLASSIFICATION RATE = 0.75
ERROR RATE = 0.25
TRUE POSITIVE RATE = 0.8868778280542986
FALSE POSITIVE RATE = 0.7014925373134329
TRUE NEGATIVE RATE = 0.29850746268656714
FALSE NEGATIVE RATE = 0.11312217194570136
PRECISION = 0.8065843621399177
RECALL = 0.8868778280542986
F-MEASURE = 0.8448275862068966
```

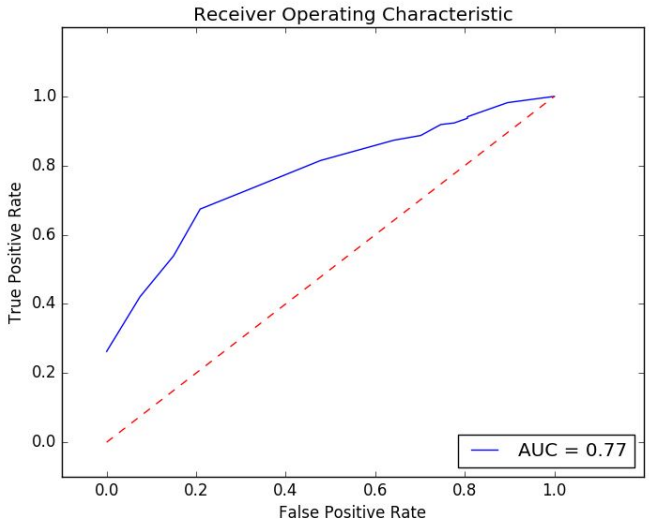
The above figure shows the parameter values got by running kNN algorithm on Income dataset for k=10 and k=15 using Euclidean distance similarity.

ROC Curves for k=5, 10 and 15 :-





k=5 **k=10**



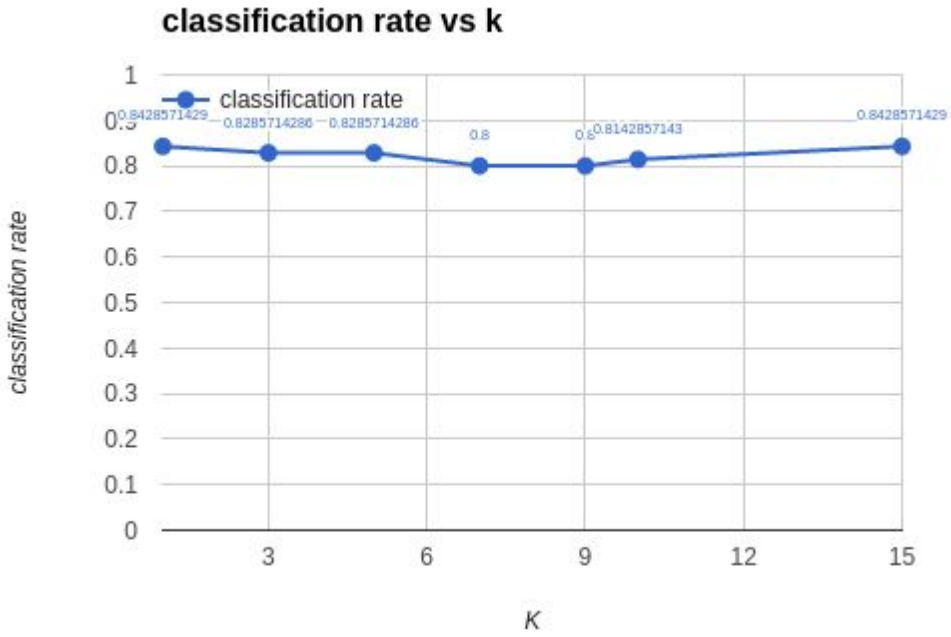
k=15

ROC Curve and implementation:

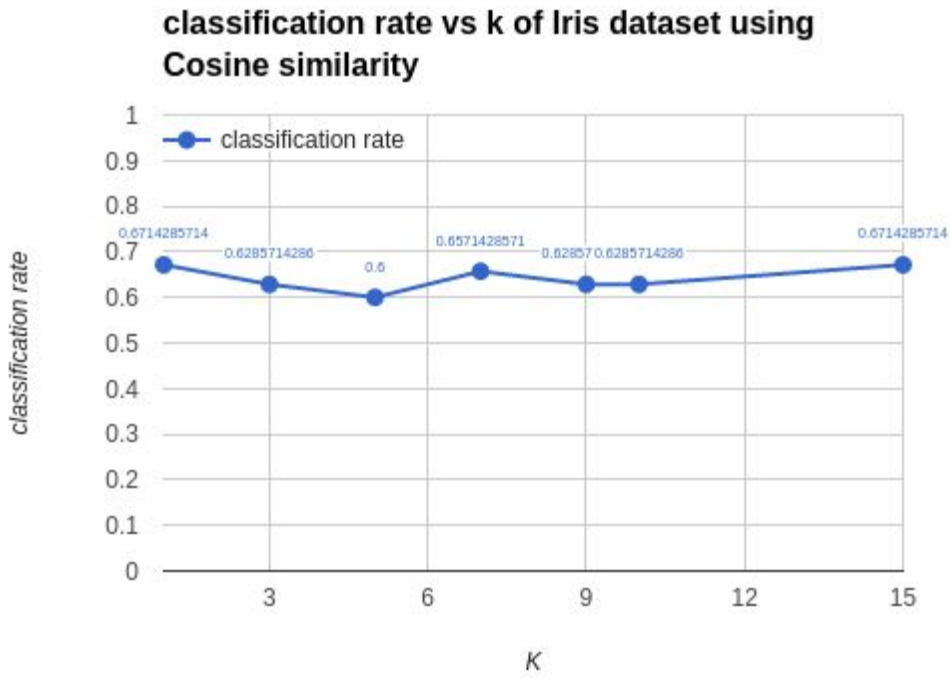
- ROC stands for Receiver Operating Characteristic and it is plotted to illustrate the performance of a binary classifier as the system's discrimination threshold is varied. The curve is plotted with True Positive Rate on Y-axis and False Positive Rate on X-axis.
- TPR measures the number of true positive examples that were correctly classified as positive examples. And, FPR measures those false examples which were incorrectly predicted to be positive.

- When Area Under Curve(AUC) of an ROC curve is less, something wrong is happening with the classifier system. Always, a higher value of AUC is an indicator that the binary classifier is doing a good job!
- ROC curves were plotted for the Income datasets for varying values of k and proximity measure. Python's s **scikit-learn** library was used to compute the parameters(TPR,FPR) for the ROC curves. Functions used are **roc_curve, auc. Matplotlib** was used to plot the ROC curve.

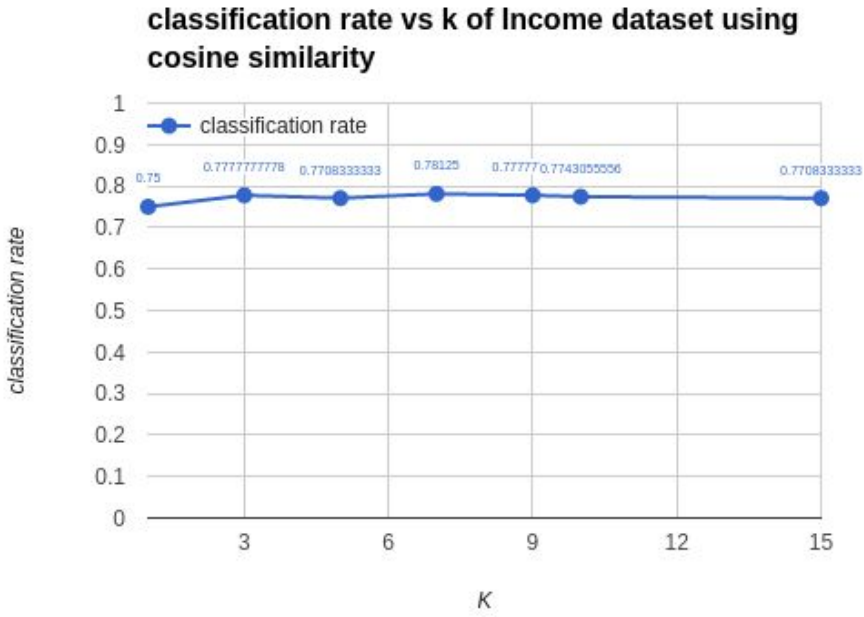
Section 2C - Analysis of results:

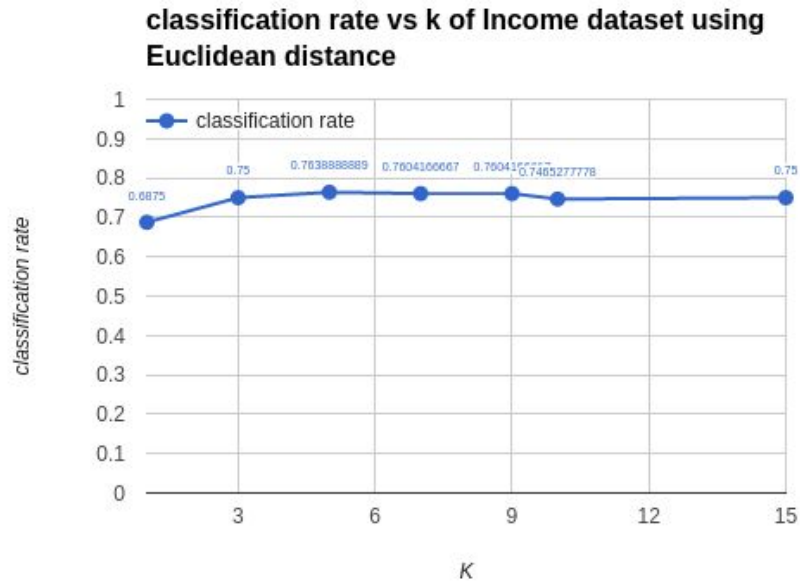


This is a line graph showing the correlation between classification rate and the value of k for the Iris dataset when Euclidean distance is used as the similarity metric.



This is a line graph showing the correlation between classification rate and the value of k for the Iris dataset when Cosine similarity is used as the similarity metric.



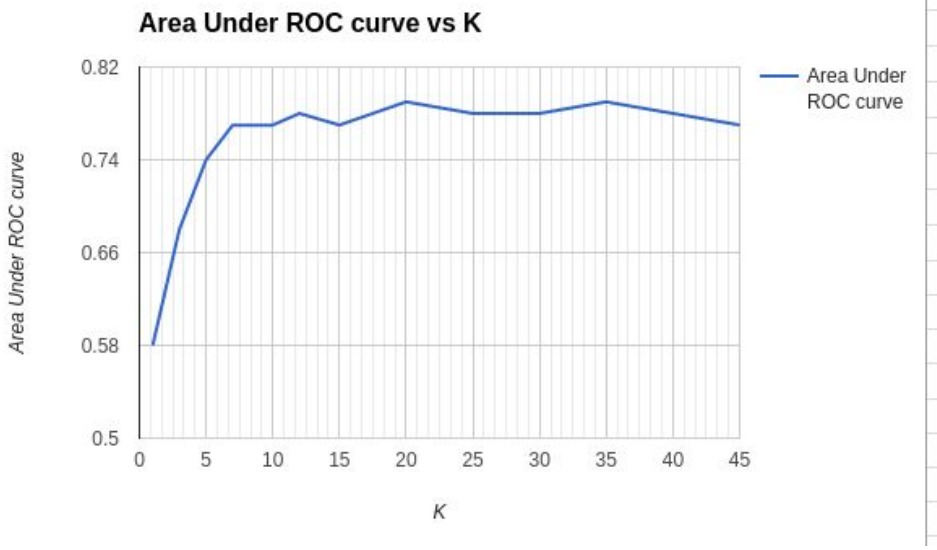


The above graphs show the correlation between classification rate and the value of 'k' for the Income dataset. Classification rate measures the number of data records that were correctly classified by the kNN classification algorithm. Classification rate is calculated from the confusion matrix produced. The diagonal elements represent the number of correctly classified records and the off diagonal records tell the mistakes done by the classifier. Ideally, we must aspire for a confusion matrix with 0 off-diagonal elements!

Error rate is calculated as (1 - classification_rate).

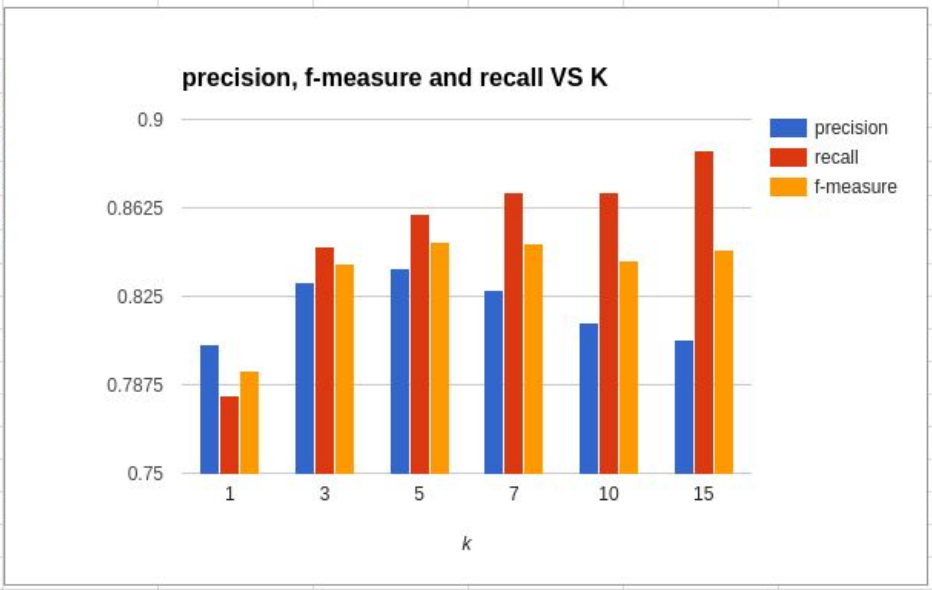
- The following visualization was generated for the **Income** dataset whose area under the ROC curve values were calculated using **Euclidean** distance as similarity measure for different values of k from 1 to 45.
- It shows that the value of area under the curve(AUC) sharply increases initially when k's value increases and at a point reaches saturation from where, the value of AUC does not increase/change much for any significant increase in the value of k.
- We can see that there's a spike in the Area under the curve value for values of k <10. Optimal AUC in this case is 0.77

K	Area Under ROC curve
1	0.58
3	0.68
5	0.74
7	0.77
9	0.77
10	0.77
12	0.78
15	0.77
20	0.79
25	0.78
30	0.78
35	0.79
45	0.77



The following figure is a combined bar graph which shows the precision, recall and F-measure values of the Income dataset for various values of k using Euclidean distance as the similarity measure.

k	precision	recall	f-measure
1	0.8046511628	0.7828054299	0.7935779817
3	0.8311111111	0.8461538462	0.8385650224
5	0.8370044053	0.8597285068	0.8482142857
7	0.8275862069	0.8687782805	0.8476821192
10	0.813559322	0.8687782805	0.8402625821
15	0.8065843621	0.8868778281	0.8448275862

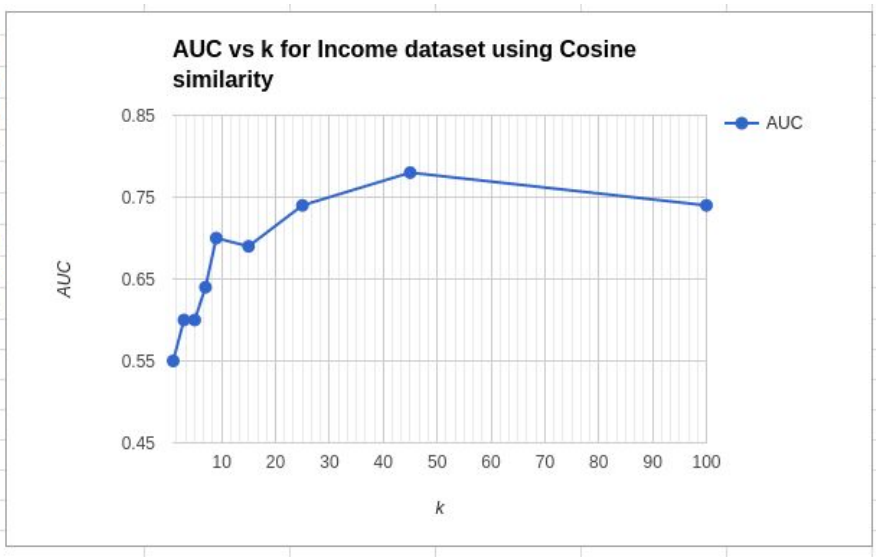


- Formula used to compute Precision :- $\frac{True\ Positive}{True\ Positive + False\ Positive}$
- Formula used to compute Recall :- $\frac{True\ Positive}{True\ Positive + False\ Negative}$
- Formula used to compute F-measure :- $\frac{2 * Precision * Recall}{Precision + Recall}$

Section 2.D:

- The 2 proximity measures used were Euclidean distance and cosine similarity.
- On comparing the classification rates got by doing kNN classification for Iris dataset using Euclidean and Cosine similarity, we can see that Euclidean version of kNN had a better accuracy than it's cosine counterpart. This can be seen in the k VS classification_rate graphs given in Page 10. Maximum classification rate got using Euclidean distance was 0.842857 while the max. got using cosine similarity was 0.6714.

k	AUC
1	0.55
3	0.6
5	0.6
7	0.64
9	0.7
15	0.69
25	0.74
45	0.78
100	0.74



- AUC values for Income dataset got by using Euclidean and Cosine similarity measures followed the same trend. The AUC values increased for increase in k and beyond a point reached saturation. The above figure is the graph plotted to represent the AUC vs K for the Income dataset using Cosine similarity.
- I observed that the usage of Euclidean distance for Iris data gave better classification rate and precision and the usage of Cosine similarity for Income dataset was better.

Section (2.E)

- I would recommend using a small odd value for k. This will make sure that the votes for positive and negative class is never equal. If k was even, in the worst case, half of the neighbours could belong to positive class and the other half could belong to the negative class. This will then put the kNN algorithm in a state that's as bad as random guessing. The probability of assigning the test record it's correct label will be 0.5 in

that case. In order to prevent this case, k must be chosen as an odd number to ensure that there's A majority voting among the neighbours.

- From the analysis done, optimal value of k can be 5 or 7. For these values, the classification rate, AUC and precision were high among the values computed for several different values of k.

Section 2.F:

The following changes were made to Assignment 1:-

- New procedure was written to handle the test data records' missing values. These values were imputed based on the training data
- New procedure was written to normalize the test data record using the minimum and maximum attribute values of the training set.
- New procedure was written to produce confusion matrix and compute classification rate, TPR, FPR, TNR, FNR, precision, recall, F-measure.

SECTION 3 - COMPARISON WITH SCIKIT-LEARN kNN CLASSIFIER

- The performance of my kNN classifier was compared with Scikit-learn's KNeighborsClassifer (click here)

- The training and testing data were given as such to scikit-learn's kNN classifier without any preprocessing(handling missing data or normalization).

- The line that generates the model is :-

  Neigh = KNeighborsClassifier(n_neighbors=k, weights='uniform', algorithm='auto', p=2, metric='minkowski')

- Parameters of the above function are :-
  - The value of k which is the number of neighbors
  - Weights : uniform weight is assigned to each neighbor
  - Algorithm ='auto' : The auto keyword will automatically fix the most appropriate algorithm to compute the nearest neighbours. Either brute force search or k-dimensional tree is chosen
  - p=2 : Power parameter for the Minkowski metric. It's set to 2 for euclidean distance.
  - metric='minkowski' : the distance metric to use for the tree. The default metric is minkowski, and with p=2 is equivalent to the standard Euclidean metric.

- The following is a table showing the accuracy scores of my kNN classifier and scikit-learn's KNeighborsClassifier for the Iris dataset for various values of 'k' using euclidean distance as the similarity metric.

- Though both my model and scikit-learn's knn model have comparable accuracies, scikit-learn's classifier performs slightly better than my model.

k-value	**My kNN accuracy**	**Scikit learn's kNN accuracy**
1	84.285%	85.71428%
3	82.8571%	82.8571%
5	82.8571%	82.8571%
7	80%	82.8571%
9	80%	82.8571%
10	81.428%	82.8571%
15	84.285%	87.1428%
25	85.714%	88.571428%
50	85.714%	87.1428%
75	87.142%	85.71428%

CONCLUSION:

Thus, k Nearest Neighbors were computed for Income and Iris datasets for different parameter values and the results were analyzed and compiled into this report.