# PROGRAMMING ASSIGNMENT 1
## CSE 6363 - MACHINE LEARNING

NAVYA R SOGI
1001753085

This assignment aims at implementing K-nearest neighbors algorithm on three different datasets from scratch, along with K-fold cross validation and comparing the accuracies of their results through Python code and Weka software predictions.

## K-Nearest Neighbors Algorithm - How it works?

The K-nearest neighbors (KNN) algorithm is a type of supervised machine learning algorithms. The intuition behind the KNN algorithm is one of the simplest of all the supervised machine learning algorithms. It simply calculates the distance of a new data point to all other training data points. The distance can be of any type e.g Euclidean or Manhattan etc. It then selects the K-nearest data points, where K can be any integer. Finally it assigns the data point to the class to which the majority of the K data points belong.
The following three datasets have been used and converted into .csv files:

- Hayes-Roth dataset (6 attribues)
- Car Evaluation dataset (7 attributes)
- Breast Cancer dataset (10 attributes)

The Python program to implement KNN from scratch is done in the following steps:

- Loading the dataset, preprocessing the dataset by converting to .csv files, removing empty records. **load_file()** function serves this purpose.
- Another important step in data preprocessing is to vectorize the dataset, I.e., convert the rows into a matrix of numerical values. Each row is treated as a matrix/vector of numbers. This is necessary because in KNN implementation, we use distances to measure similarity between two rows and also to calculate the mean and standard deviations of the distances, to know which of the new data point's neighbors are nearest to it. Hence, the distances have been implemented using vectorized computation through **str_column_to_int()** function.
- The minimum and maximum values for each column is calculated and the dataset is again normalized to scale between 0-1 as a part of further preprocessing. This is implemented through **dataset_minimax()** and **normalize_dataset()** functions.
- Distance between two vectors is calculated using Euclidean, Manhattan, Hamming distance measures. Using these, the K nearest neighbors are obtained based on the closest distance between the two vectors. If distance between two vectors is 0, then both vectors are the same. **euclidean_distance(), manhattan_distance(), hamming_distance()** functions implement these distance measures. **get_neighbors()** function locate the most similar neighbors based on the k value passed.

- The most represented class among the neighbors can be returned by performing the *max()* function on the list of output values from the neighbors. Given a list of class values observed in the neighbors, the max() function takes a set of unique class values and calls the count on the list of class values for each class value in the set in **predict_classification() and k_nearest_neighbors().**
- K-fold cross validation is used for evaluating the model performance. K-Fold CV is where a given data set is split into a ***K*** number of sections/folds where each fold is used as a testing set at some point.In the first iteration, the first fold is used to test the model and the rest are used to train the model. In the second iteration, 2nd fold is used as the testing set while the rest serve as the training set. This process is repeated until each fold of the K folds have been used as the testing set. **cross_validation()** is used for this. In this assignment, 10 fold cross validation has been used.
- The algorithm is then evaluated to find out the accuracy results using **accuracy_metric().**

## Accuracy Comparisons:

## K = 5 with 10-fold cross validation

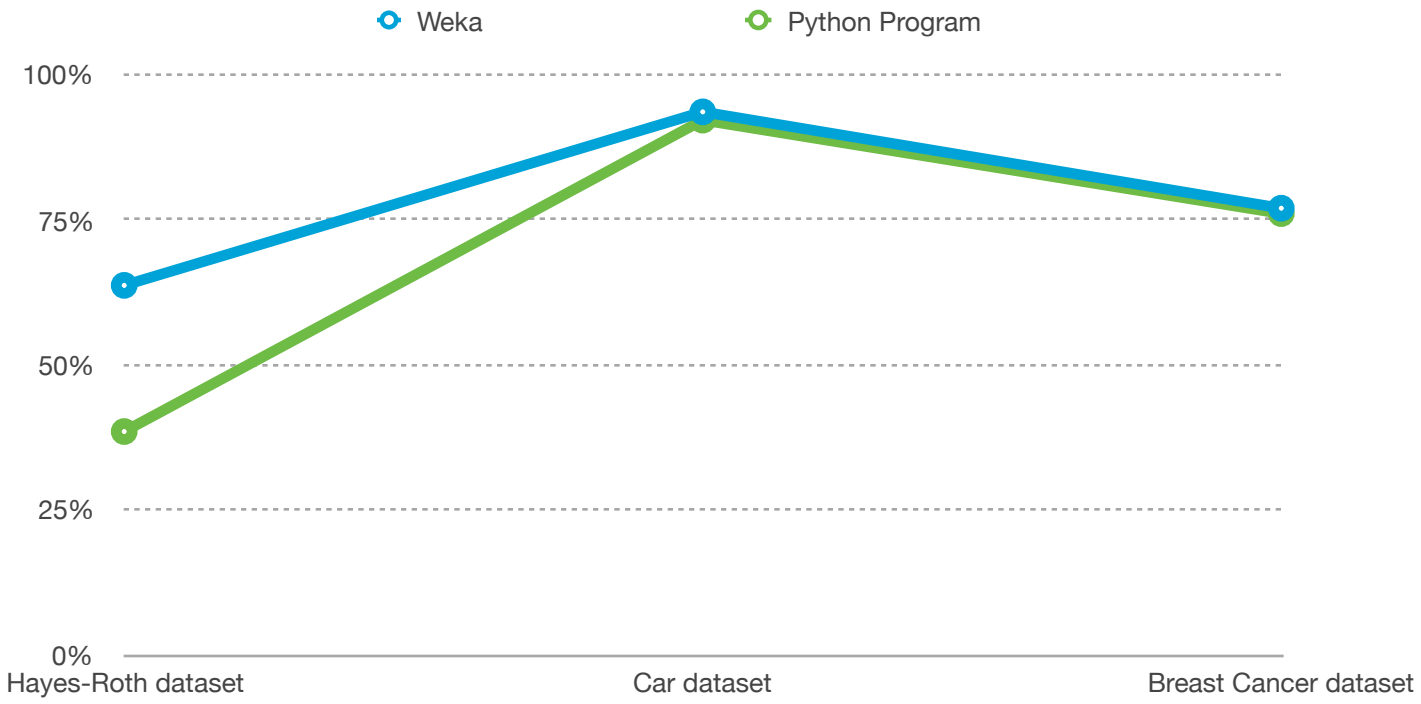
## Python program results:

| | Hayes-Roth dataset | Car dataset | Breast Cancer dataset |
|---|---|---|---|
| **Mean Accuracy** | 38.462% | 92.849% | 76.071% |

```
(base) Navyas-MacBook-Pro:Desktop navyasogi$ python3 Hayes_Roth.py
Scores: [53.84615384615385, 30.76923076923077, 53.84615384615385, 7.692307692307
6925, 61.53846153846154, 23.076923076923077, 30.76923076923077, 61.5384615384615
4, 23.076923076923077, 38.46153846153847]
Mean Accuracy: 38.462%
```

```
(base) Navyas-MacBook-Pro:Desktop navyasogi$ python3 car.py
Scores: [94.18604651162791, 90.11627906976744, 93.02325581395348, 93.60465116279
07, 91.86046511627907, 93.02325581395348, 93.02325581395348, 91.86046511627907,
94.18604651162791, 93.6046511627907]
Mean Accuracy: 92.849%
```

```
(base) Navyas-MacBook-Pro:Desktop navyasogi$ python3 breast_cancer.py
Scores: [75.0, 67.85714285714286, 75.0, 67.85714285714286, 71.42857142857143, 78
.57142857142857, 78.57142857142857, 82.14285714285714, 96.42857142857143, 67.857
14285714286]
Mean Accuracy: 76.071%
```




# Weka results:

| | Hayes-Roth dataset | Car dataset | Breast Cancer dataset |
|---|---|---|---|
| **Mean Accuracy** | 63.6364% | 93.5185% | 76.9231% |

**Classifier output**

```
Scheme:       weka.classifiers.lazy.IBk -K 5 -W 0 -A "weka.core.neighboursearch.LinearNNSearch -A \"weka.core.EuclideanDistance -R first-last\""
Relation:     hayes-roth-weka.filters.unsupervised.attribute.NumericToNominal-Rfirst-last
Instances:    132
Attributes:   6
              name
              hobby
              age
              educationalL
              maritalS
              class
Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===

IB1 instance-based classifier
using 5 nearest neighbour(s) for classification


Time taken to build model: 0 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances          84               63.6364 %
Incorrectly Classified Instances        48               36.3636 %
Kappa statistic                          0.4143
Mean absolute error                      0.3501
Root mean squared error                  0.4079
Relative absolute error                 80.7647 %
Root relative squared error             87.6388 %
Total Number of Instances              132

=== Detailed Accuracy By Class ===

               TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
               0.843    0.370    0.589      0.843    0.694      0.463    0.840     0.748     1
               0.686    0.222    0.660      0.686    0.673      0.461    0.850     0.779     2
               0.200    0.000    1.000      0.200    0.333      0.402    0.988     0.945     3
Weighted Avg.  0.636    0.229    0.710      0.636    0.604      0.448    0.877     0.805

=== Confusion Matrix ===

  a  b  c   <-- classified as
 43  8  0 |  a = 1
 16 35  0 |  b = 2
 14 10  6 |  c = 3
```

**Classifier output**

```
Attributes:   7
              buying
              maint
              doors
              persons
              lug_boot
              safety
              Class
Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===

IB1 instance-based classifier
using 5 nearest neighbour(s) for classification


Time taken to build model: 0 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances        1616               93.5185 %
Incorrectly Classified Instances       112                6.4815 %
Kappa statistic                          0.853
Mean absolute error                      0.1122
Root mean squared error                  0.1953
Relative absolute error                 48.9977 %
Root relative squared error             57.7645 %
Total Number of Instances             1728

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.998    0.066    0.973      0.998    0.985      0.949    1.000     1.000     unacc
                 0.911    0.058    0.818      0.911    0.862      0.822    0.988     0.958     acc
                 0.708    0.000    1.000      0.708    0.829      0.836    1.000     1.000     vgood
                 0.188    0.000    1.000      0.188    0.317      0.427    0.994     0.859     good
Weighted Avg.    0.935    0.059    0.940      0.935    0.925      0.896    0.997     0.985

=== Confusion Matrix ===

    a    b    c    d   <-- classified as
 1207    3    0    0 |    a = unacc
   34  350    0    0 |    b = acc
    0   19   46    0 |    c = vgood
    0   56    0   13 |    d = good
```

**Classifier output**

```
Instances:    286
Attributes:   10
              Class
              age
              menopause
              tumor-size
              inv-nodes
              node-caps
              deg-malig
              breast
              breast-quad
              irradiat
Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===

IB1 instance-based classifier
using 5 nearest neighbour(s) for classification


Time taken to build model: 0 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         220               76.9231 %
Incorrectly Classified Instances        66               23.0769 %
Kappa statistic                          0.181
Mean absolute error                      0.2975
Root mean squared error                  0.4158
Relative absolute error                 81.8349 %
Root relative squared error             97.6636 %
Total Number of Instances              286

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.950    0.809    0.790      0.950    0.863      0.216    0.680     0.867     no
                 0.191    0.050    0.542      0.191    0.283      0.216    0.680     0.427     yes
Weighted Avg.    0.769    0.629    0.731      0.769    0.725      0.216    0.680     0.762

=== Confusion Matrix ===

   a   b   <-- classified as
 207  11 |   a = no
  55  13 |   b = yes
```

The results show that Weka produces better accuracy than the program implementation.

## Enhancements to KNN:

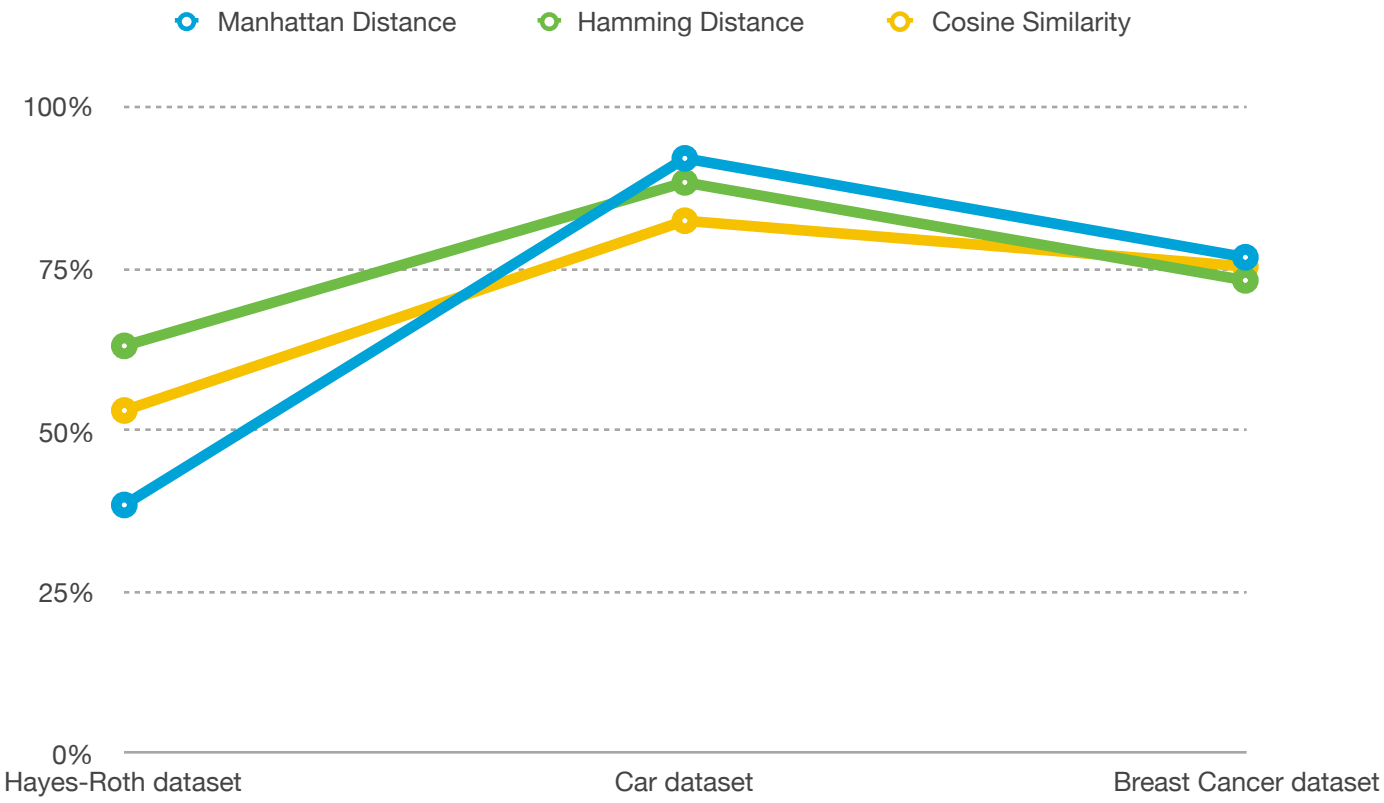
The following enhancements have been made to the KNN implementation:

- **Cosine Similarity measure:** Using the Cosine function & K-Nearest Neighbor algorithm, we can determine how similar or different two sets of items are and use it to determine the classification. The Cosine function is used to calculate the similarity or the distance of the observations in high dimensional space. **cosineSimilarity()** function has been implemented for this purpose.

- **Other Distance measures:** Apart from Euclidean distance, other distance measures - Manhattan, Hamming distance and Cosine Similarity measure have also been implemented and compared for accuracy.

## Accuracies when K = 5 with 10-fold cross validation

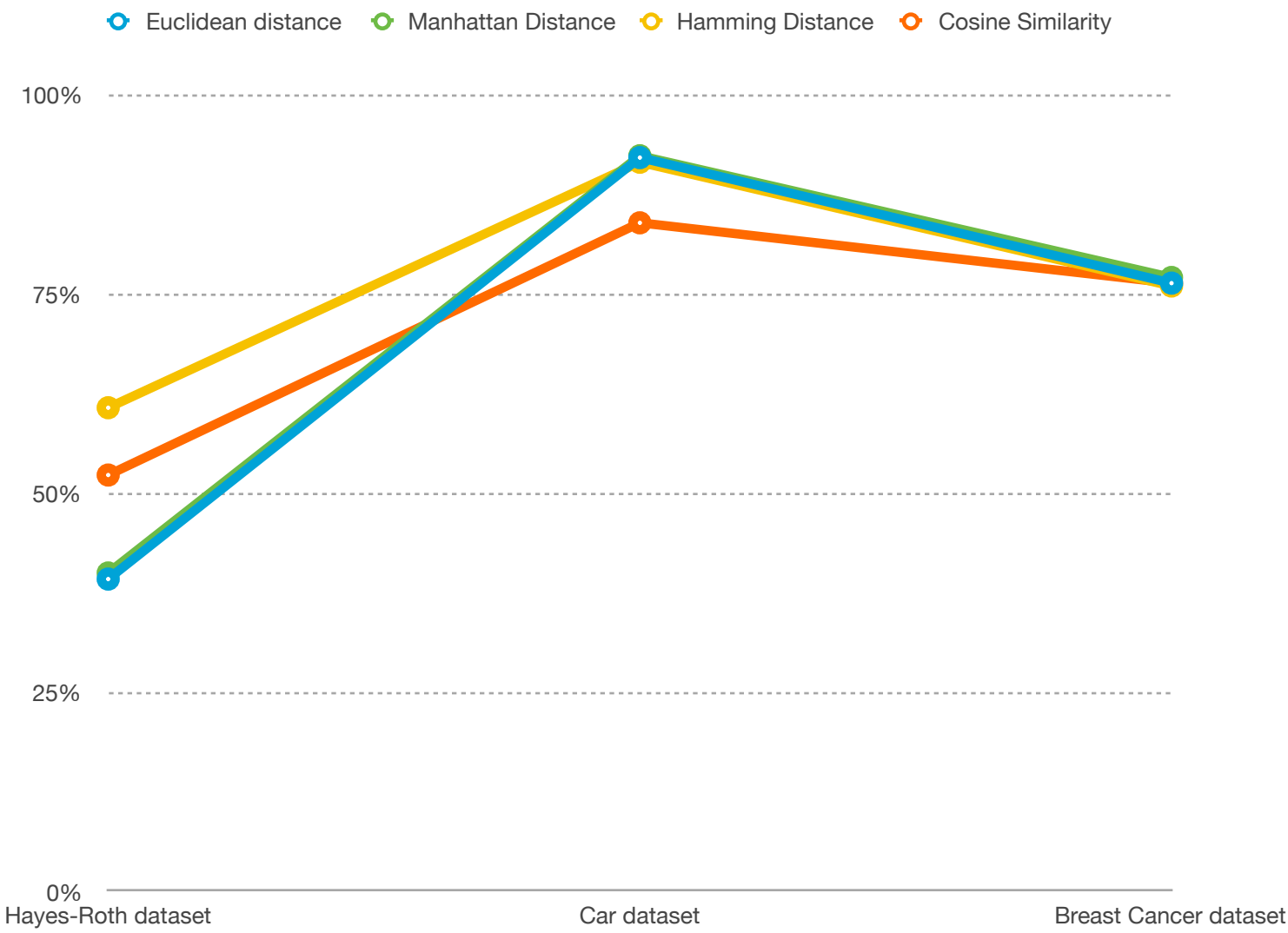
| | Hayes-Roth dataset | Car dataset | Breast Cancer dataset |
|---|---|---|---|
| **Manhattan Distance** | 38.462% | 92.093% | 76.786% |
| **Hamming Distance** | 63.077% | 88.372% | 73.214% |
| **Cosine Similarity** | 53.077% | 82.442% | 75.357% |



- **Tuning KNN:** The program is experimented with larger values of K to see if there is any improvement in performance.

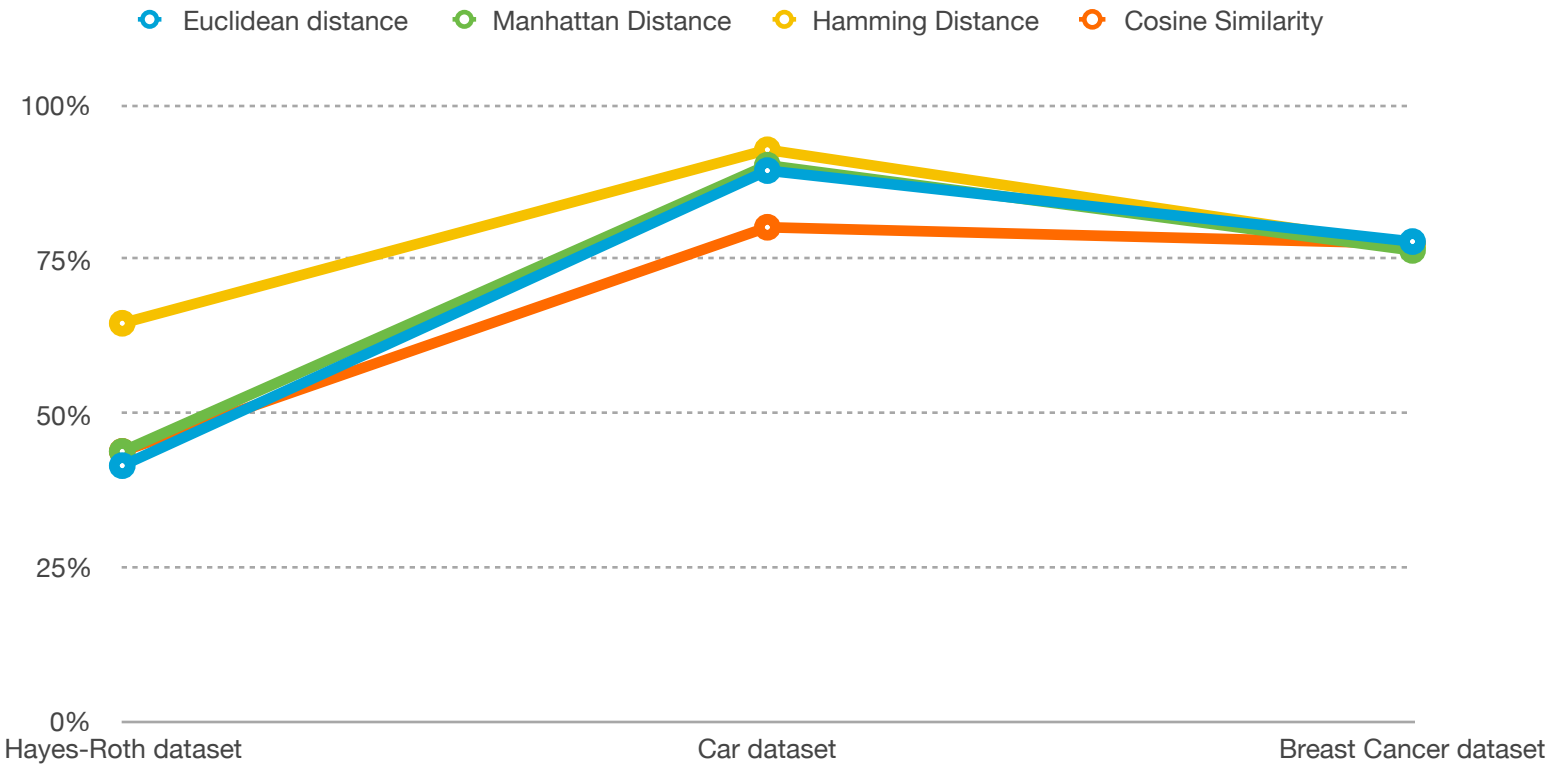
**Case 1: K = 10 with 10-fold cross validation**

| | Hayes-Roth dataset | Car dataset | Breast Cancer dataset |
|---|---|---|---|
| **Euclidean distance** | 39.231% | 92.209% | 76.429% |
| **Manhattan Distance** | 40.000% | 92.442% | 77.143% |
| **Hamming Distance** | 60.769% | 91.628% | 76.071% |
| **Cosine Similarity** | 52.308% | 84.012% | 76.429% |



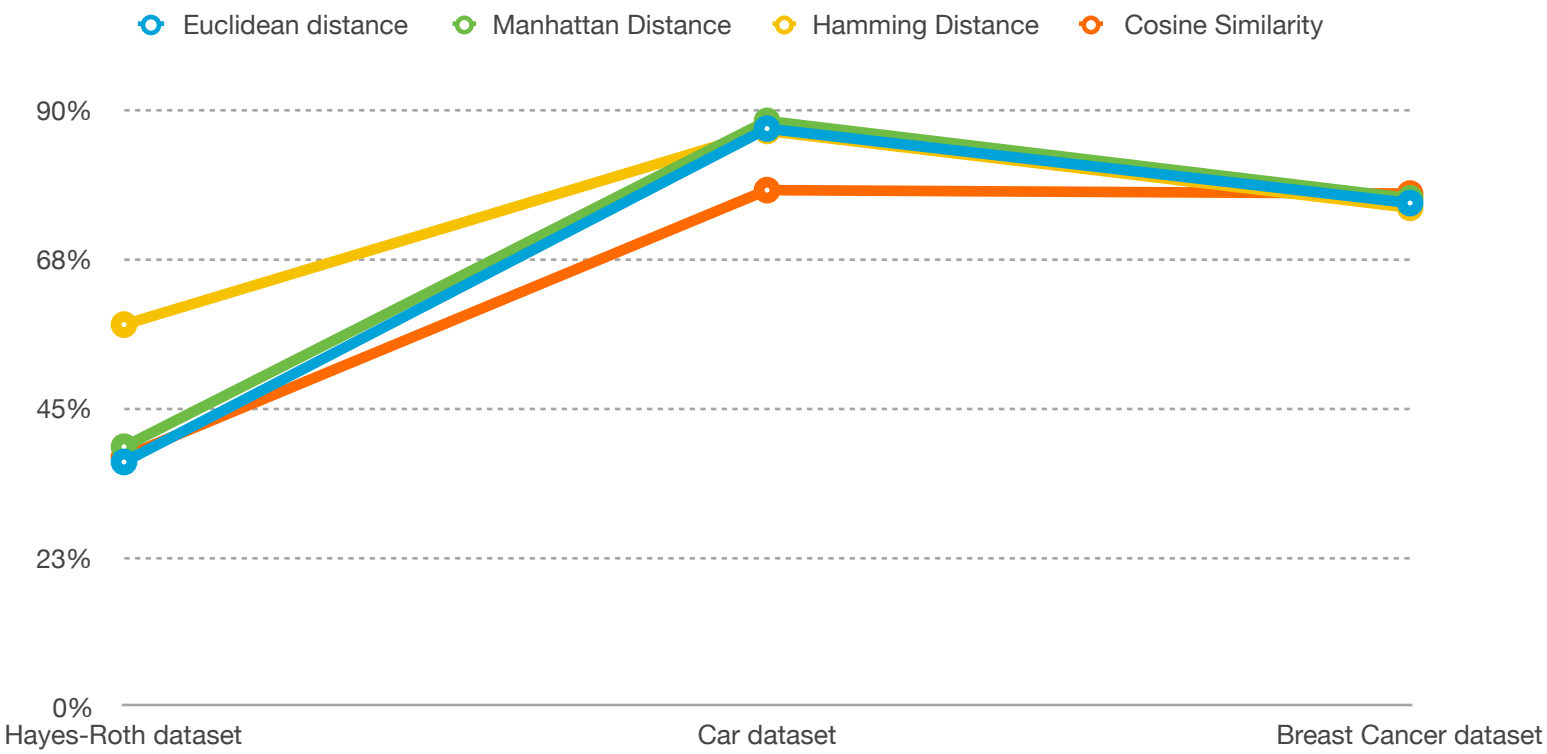
### Case 2: K = 15 with 10-fold cross validation

| | Hayes-Roth dataset | Car dataset | Breast Cancer dataset |
|---|---|---|---|
| **Euclidean distance** | 41.538% | 89.360% | 77.857% |
| **Manhattan Distance** | 43.846% | 90.233% | 76.429% |
| **Hamming Distance** | 64.615% | 92.733% | 77.143% |
| **Cosine Similarity** | 43.846% | 80.174% | 77.500% |



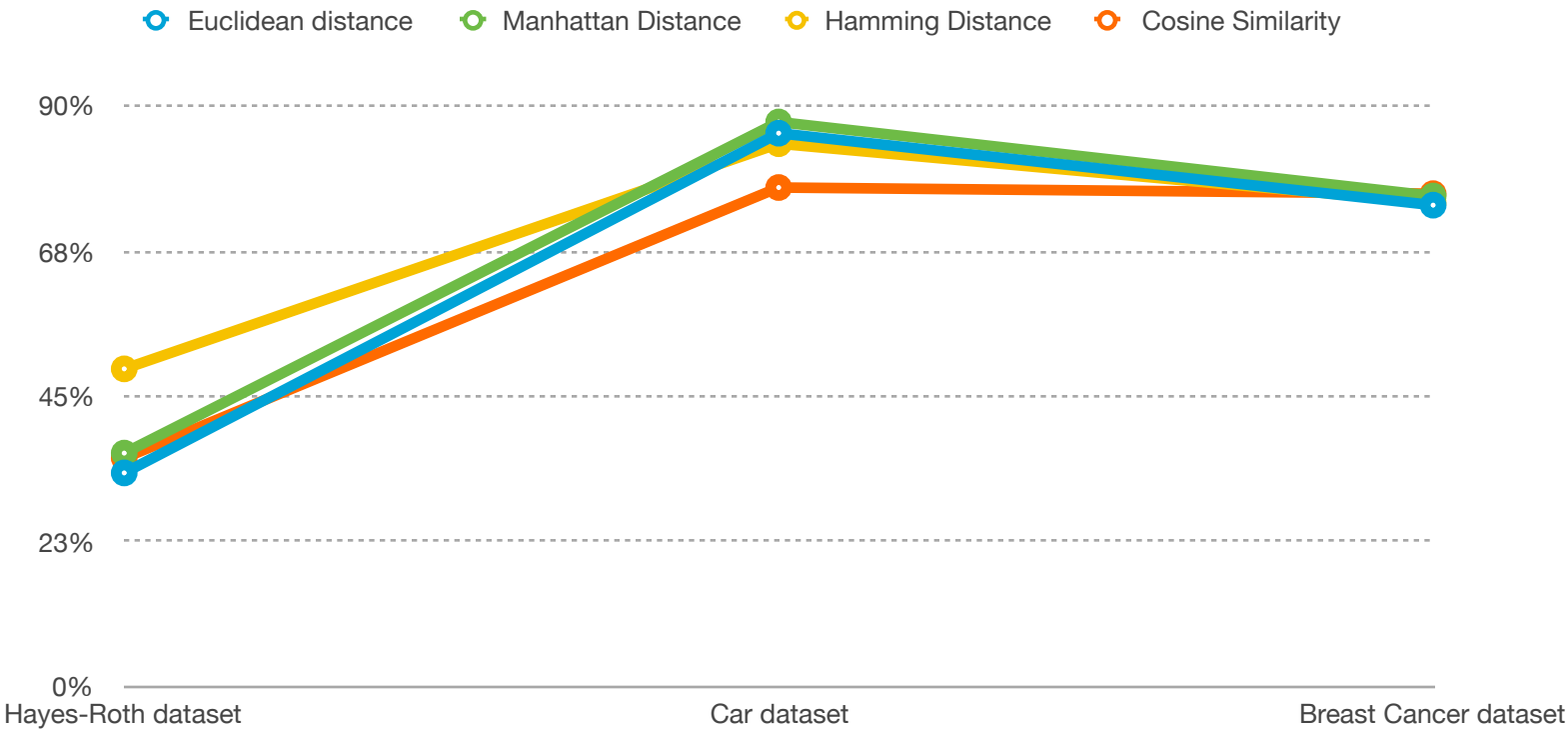

### Case 3: K = 23 with 10-fold cross validation

| | Hayes-Roth dataset | Car dataset | Breast Cancer dataset |
|---|---|---|---|
| **Euclidean distance** | 36.923% | 87.326% | 76.071% |
| **Manhattan Distance** | 39.231% | 88.488% | 76.786% |
| **Hamming Distance** | 57.692% | 86.977% | 75.357% |
| **Cosine Similarity** | 37.692% | 78.023% | 77.500% |



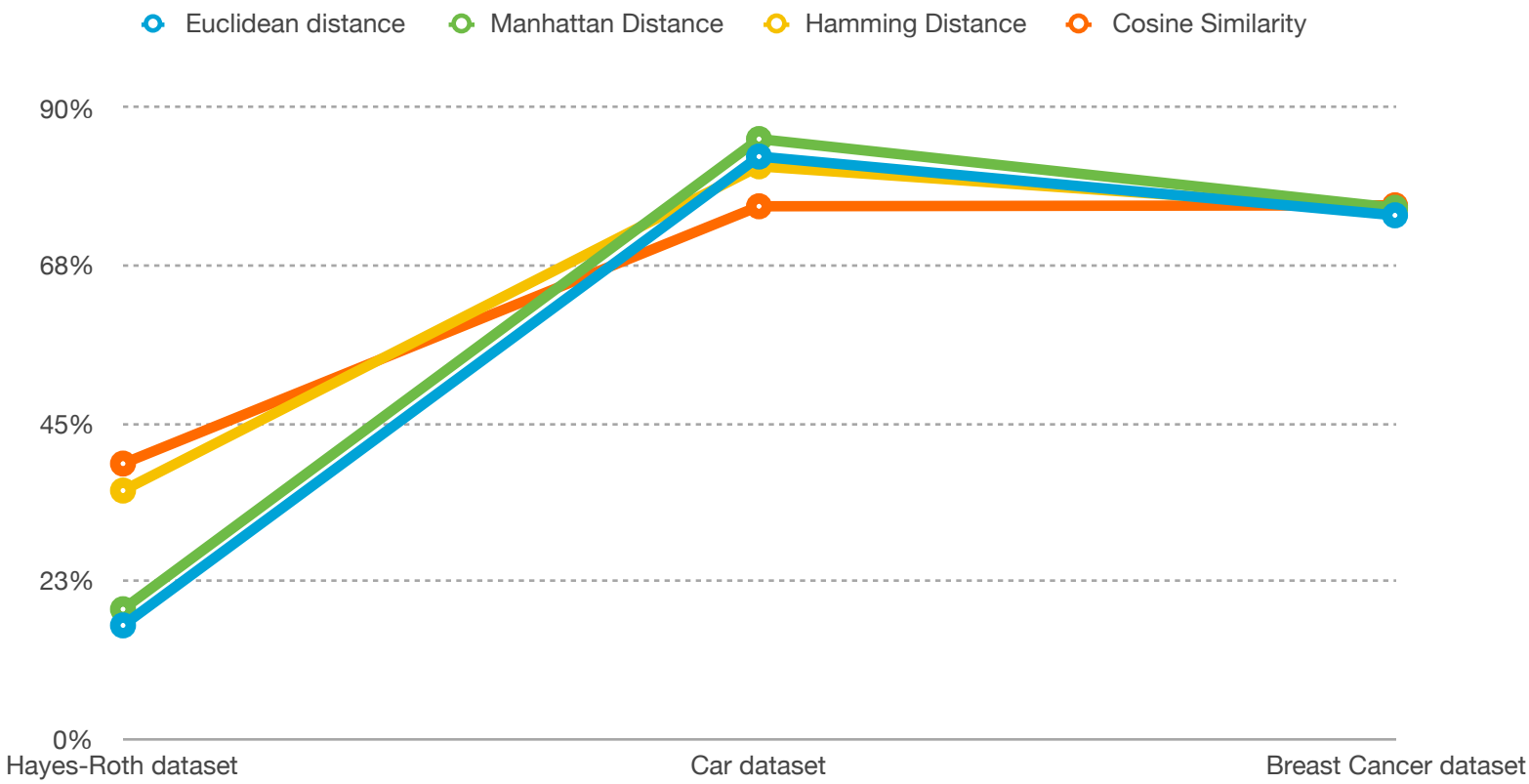

**Case 4: K = 30 with 10-fold cross validation**

| | Hayes-Roth dataset | Car dataset | Breast Cancer dataset |
|---|---|---|---|
| **Euclidean distance** | 33.077% | 85.814% | 74.643% |
| **Manhattan Distance** | 36.154% | 87.558% | 76.071% |
| **Hamming Distance** | 49.231% | 84.186% | 75.000% |
| **Cosine Similarity** | 35.385% | 77.384% | 76.429% |



**Case 5: K = 45 with 10-fold cross validation**

| | Hayes-Roth dataset | Car dataset | Breast Cancer dataset |
|---|---|---|---|
| **Euclidean distance** | 16.154% | 83.023% | 74.643% |
| **Manhattan Distance** | 18.462% | 85.523% | 75.714% |
| **Hamming Distance** | 35.385% | 81.628% | 75.357% |
| **Cosine Similarity** | 39.231% | 75.930% | 76.071% |




**Inferences and Conclusion:**

- If you increase k, the areas predicting each class will be more "smoothed", since it's the majority of the k-nearest neighbors which decide the class of any point.
- With increasing values of K, the accuracy of Hayes_Roth dataset seems to be decreasing.
- Manhattan distance seems to be providing a better accuracy rate compared to other distance measures, for car dataset.

- For Hayes-Roth dataset, Cosine similarity proves to be a good measure for determining similarity and accuracy.
- And for breast cancer dataset, all the similarity measures give almost the same accuracies.

**Files Attached:**
- Hayes_Roth.py: KNN implementation in python for Hayes-Roth dataset
- car.py: KNN implementation in python for car dataset
- breast_cancer.py: KNN implementation in python for breast cancer dataset
- hayes-roth.csv, car.csv, breast-cancer.csv: CSV converted files

## Instructions to run python program:

Open terminal/command prompt, navigate to the folder in which the above files have been saved and type the following command:

python3 <filename>.py where,
filename can be Hayes_Roth, car, breast_cancer.

Can also be run in a Python IDE. I have used Spyder.

NOTE: The path of the file should be changed accordingly in the program.

## References:


- https://machinelearningmastery.com/tutorial-to-implement-k-nearest-neighbors-in-python-from-scratch/
- https://github.com/ghifarozarrr/k-nearest-neighbors-algorithm/blob/master/K-NN%20from%20scratch.py#L125
- https://medium.com/@souravdey/l2-distance-matrix-vectorization-trick-26aa3247ac6c