Amirkabir University of Technology
(Tehran Polytechnic)

Computer Engineering Department

Machine Learning Course
CE5501

Homework 2

Student
Reza Sajedi
400131072
r.sajedi@aut.ac.ir

Teacher
Dr. Nazerfard

Fall 2022

Table of Contents

Problem 1: Why and how

Part a

City Blocks (a.k.a. Manhattan distance) and Euclidean distance are two kinds of Minkowski Distance with p=1 and p=2 respectively.

$$\left(\sum_{i=1}^{n}|u_i - v_i|^p\right)^{1/p}$$

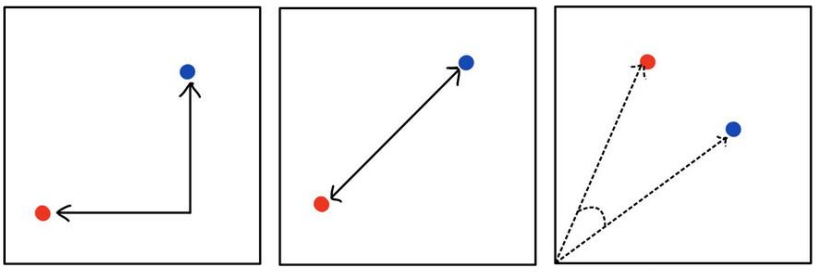
These metrics can be used in a space where distances can be represented as a vector with a non-negative length.

Cosine similarity is measured by the cosine of the angle between two vectors and Cosine distance is calculated as follows:

$$1 - \cos\theta = 1 - \frac{\vec{u}\,.\,\vec{v}}{|\vec{u}||\vec{v}|}$$

It is often used to measure document similarity in text analysis. When used with KNN this distance gives us a new perspective on a business problem and lets us find some hidden information in the data which we didn't see using the above two distance matrices.

The distance values calculated using these metrics are different. For example, two arbitrary points on the line y = x have different non-zero distances using Euclidean and City Blocks, while the cosine distance is zero.



Part b

It depends on the dataset. The best way is to examine various metrics on the desired dataset. Euclidean distance usually may have better results. (Like the results of handwritten digits recognition which is an image classification task)

Part c

After finding the neighbors in the K-NN algorithm, we can simply take the average of target values. The ID3 algorithm can be used to construct a decision tree for regression by replacing Information Gain with Standard Deviation Reduction. The standard deviation reduction is based on the decrease in standard deviation after a dataset is split on an attribute. Constructing a decision tree is all about finding an attribute that returns the highest standard deviation reduction.

Part d

One of the hyperparameters that control the decision tree's complexity is maximum depth. If the depth of the tree gets so large, it tends to overfit which means may perform badly on unseen instances (test data). In such situations, we can remove some nodes to decrease the depth and get a more general model. This process is called pruning.

Part e

$$E(X) = -\sum_{i} p(x_i) \log p\,(x_i)$$

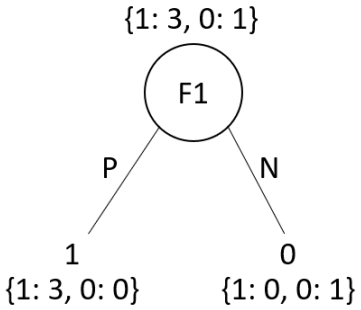
$$G(S, F) = E(S) - \sum_{v \in values(F)} \frac{|S_v|}{|S|} E(S_v)$$

Root:

E(S) = - (1/4 * log(1/4) + 3/4 * log(3/4)) = 0.81

G(F1) = 0.81 - (-3/4 * (3/3 * log(3/3)) -1/4 * (1/1 * log(1/1))) = 0.81

No need to calculate information gain for other features! Because the entropy of F1 is zero, we can not get an information gain higher than 0.81 which is the entropy of the entire set. So, F1 is the best split for the root and after that, we reach the leaves.



Part f

Root:

E(S) = - (2/5 * log(2/5) + 3/5 * log(3/5)) = 0.97

G(F1) = 0.97 - (-4/5 * (3/4 * log(3/4) + 1/4 * log(1/4)) -1/5 * (1/1 * log(1/1))) = 0.32

G(F2) = 0.97 - (-1/5 * (1/1 * log(1/1)) -4/5 * (3/4 * log(3/4) + 1/4 * log(1/4))) = 0.32

G(F3) = 0.97 - (-2/5 * (1/2 * log(1/2) + 1/2 * log(1/2)) -3/5 * (2/3 * log(2/3) + 1/3 * log(1/3))) = 0.02

G(F4) = 0.97 - (-4/5 * (3/4 * log(3/4) + 1/4 * log(1/4)) -1/5 * (1/1 * log(1/1))) = 0.32

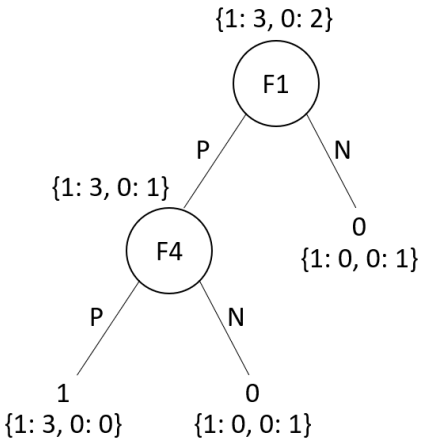
G(F5) = 0.97 - (-4/5 * (2/4 * log(2/4) + 2/4 * log(2/4)) -1/5 * (1/1 * log(1/1))) = 0.17

G(F6) = 0.97 - (-2/5 * (1/2 * log(1/2) + 1/2 * log(1/2)) -3/5 * (2/3 * log(2/3) + 1/3 * log(1/3))) = 0.02

Left Node of Root:

E(S) = - (1/4 * log(1/4) + 3/4 * log(3/4)) = 0.81

G(F4) = 0.81 - (-3/4 * (3/3 * log(3/3)) -1/4 * (1/1 * log(1/1))) = 0.81



Part g

The computational complexity of training a Decision Tree is O(m × n log(n)) because it compares all features on all samples at each node (n=number of instances, m=number of features). So, if we multiply the training set size by 10, the training time will be multiplied by K = (m × 10n × log(10n)) / (m × n × log(n)) = 10 × log(10n) / log(n). If $n = 10^6$, then K ≈ 11.7, so we can expect the training time to be roughly 11.7 hours.

Part h

The maximum possible combinations with M binary features are 2^M. So, if $N < 2^M$, the number of leaves should be equal to N. Otherwise, the maximum number of leaves is 2^M.

Part i

x1	x2	label	d1(x, [5,0])	d1(x, [1,1])	d1(x, [-2,1])	d2(x, [5,0])	d2(x, [1,1])	d2(x, [-2,1])
0	0	T	5	1	2	5	2	3
-5	0	T	10	6	3	10	7	4
-5	5	T	10	6	4	15	10	7
-5	-5	T	10	6	6	15	12	9
0	5	C	5	4	4	10	5	6
0	-5	C	5	6	6	10	7	8
8	3	C	3	7	10	6	9	12
8	-3	C	3	7	10	6	11	14
1-NN Predicted label			C	T	T	T	T	T
4-NN Predicted label			C	T	T	C	C	T

T: Triangle, C: Circle

Part j

These metrics are used to evaluate the performance in the multi-class classification task. Instead of having multiple per-class f1 scores, it would be better to average them to obtain a single number to describe overall performance. Micro and Macro are two methods of averaging.

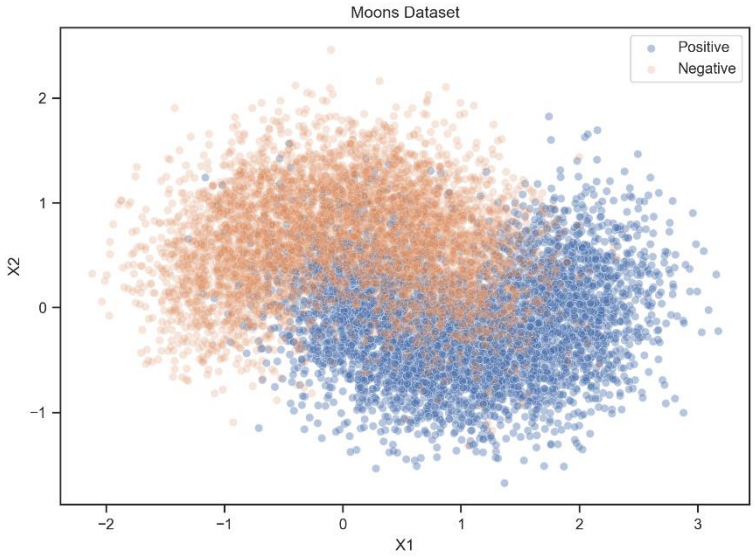
Macro averaging means computing the f1-score for each label and reporting the average without considering the proportion for each label in the dataset.

Micro averaging means computing the f1-score by considering total true positives, false negatives, and false positives (no matter the prediction for each label in the dataset). In multi-class classification, micro f1, micro precision, micro recall, and accuracy share the same value.

If the dataset is imbalanced and all classes are equally important, the macro f1-score should be used. If the dataset is balanced and we want to report an easily understandable metric for overall performance regardless of the class, micro f1-score (overall Accuracy) should be used.

Problem 2: Moons

Part a



Part b

The instruction has been followed.

Part c

Grid search is a model selection technique. It performs an exhaustive search over specified hyperparameter values for an estimator and finds the best combination of these values, leading to the model with the highest performance. We considered a range between 1 to 20 for the maximum depth of the tree and the maximum number of leaf nodes. The optimal hyperparameter values are as follows:

```
{'max_depth': 7, 'max_leaf_nodes': 18}
Accuracy: 0.8669
```

Part d

```
Accuracy: 0.8565
Confusion Matrix:
[[854 162]
 [125 859]]
```

Part e and f

```
Average Accuracy: 0.794 ± 0.0258
Max Accuracy: 0.8505
Min Accuracy: 0.689
```

Part g and h

This kind of ensemble approach which we can call Random Forest has performed a little bit better.

```
Accuracy: 0.8615
The improvement compared to part d: 0.005
```

Problem 3: Lung Cancer

Part a

```
Results for iteration 1 with train size 25%:
Accuracy: 0.8578
Confusion Matrix:
[[ 12  19]
 [ 14 187]]
Depth: 5
Number of nodes: 15
Number of leaves: 8
```

Part b

The results are different because we are choosing a small proportion of the dataset for training and the training instances are chosen randomly.

```
Results for iteration 1 with train size 25%:
Accuracy: 0.875
Confusion Matrix:
[[  0  29]
 [  0 203]]
Depth: 4
Number of nodes: 9
Number of leaves: 5

Results for iteration 2 with train size 25%:
Accuracy: 0.8793
Confusion Matrix:
[[  0  28]
 [  0 204]]
Depth: 4
Number of nodes: 9
Number of leaves: 5

Results for iteration 3 with train size 25%:
Accuracy: 0.8707
Confusion Matrix:
[[  0  29]
 [  1 202]]
Depth: 2
Number of nodes: 5
Number of leaves: 3

Results for iteration 4 with train size 25%:
Accuracy: 0.8793
```

```
Confusion Matrix:
[[  0  28]
 [  0 204]]
Depth: 0
Number of nodes: 1
Number of leaves: 1

Results for iteration 5 with train size 25%:
Accuracy: 0.8534
Confusion Matrix:
[[ 13  15]
 [ 19 185]]
Depth: 4
Number of nodes: 11
Number of leaves: 6

Results for iteration 6 with train size 25%:
Accuracy: 0.7716
Confusion Matrix:
[[ 18  13]
 [ 40 161]]
Depth: 4
Number of nodes: 9
Number of leaves: 5

Results for iteration 7 with train size 25%:
Accuracy: 0.8664
Confusion Matrix:
[[  3  27]
 [  4 198]]
Depth: 6
Number of nodes: 19
Number of leaves: 10
```

Average accuracy of 7 iterations with train size 25%: 0.8565 ± 0.0357

Part c

The results show that when the training data increases the performance improves and also makes the tree bigger.

```
Results for iteration 1 with train size 45%:
Accuracy: 0.8706
Confusion Matrix:
[[  1  20]
 [  2 147]]
Depth: 5
Number of nodes: 13
Number of leaves: 7

Results for iteration 2 with train size 45%:
Accuracy: 0.8647
Confusion Matrix:
[[  9  13]
 [ 10 138]]
Depth: 5
```

```
Number of nodes: 17
Number of leaves: 9

Results for iteration 3 with train size 45%:
Accuracy: 0.8588
Confusion Matrix:
[[  4  18]
 [  6 142]]
Depth: 5
Number of nodes: 11
Number of leaves: 6

Results for iteration 4 with train size 45%:
Accuracy: 0.8588
Confusion Matrix:
[[  0  22]
 [  2 146]]
Depth: 5
Number of nodes: 17
Number of leaves: 9

Results for iteration 5 with train size 45%:
Accuracy: 0.8294
Confusion Matrix:
[[ 14   7]
 [ 22 127]]
Depth: 5
Number of nodes: 21
Number of leaves: 11

Results for iteration 6 with train size 45%:
Accuracy: 0.8824
Confusion Matrix:
[[ 12   7]
 [ 13 138]]
Depth: 6
Number of nodes: 23
Number of leaves: 12

Results for iteration 7 with train size 45%:
Accuracy: 0.8176
Confusion Matrix:
[[  1  26]
 [  5 138]]
Depth: 5
Number of nodes: 13
Number of leaves: 7
```

Average accuracy of 7 iterations with train size 45%: 0.8546 ± 0.0213

```
Results for iteration 1 with train size 65%:
Accuracy: 0.8899
Confusion Matrix:
[[ 0 12]
 [ 0 97]]
Depth: 5
Number of nodes: 13
Number of leaves: 7
```

```
Results for iteration 2 with train size 65%:
Accuracy: 0.7982
Confusion Matrix:
[[12  3]
 [19 75]]
Depth: 5
Number of nodes: 23
Number of leaves: 12

Results for iteration 3 with train size 65%:
Accuracy: 0.8807
Confusion Matrix:
[[ 0 12]
 [ 1 96]]
Depth: 6
Number of nodes: 17
Number of leaves: 9

Results for iteration 4 with train size 65%:
Accuracy: 0.8807
Confusion Matrix:
[[ 5  8]
 [ 5 91]]
Depth: 6
Number of nodes: 19
Number of leaves: 10

Results for iteration 5 with train size 65%:
Accuracy: 0.844
Confusion Matrix:
[[ 0 16]
 [ 1 92]]
Depth: 7
Number of nodes: 27
Number of leaves: 14

Results for iteration 6 with train size 65%:
Accuracy: 0.8807
Confusion Matrix:
[[ 0 12]
 [ 1 96]]
Depth: 3
Number of nodes: 9
Number of leaves: 5

Results for iteration 7 with train size 65%:
Accuracy: 0.8899
Confusion Matrix:
[[ 0 12]
 [ 0 97]]
Depth: 5
Number of nodes: 15
Number of leaves: 8
```

Average accuracy of 7 iterations with train size 65%: 0.8663 ± 0.0313

```
Results for iteration 1 with train size 85%:
```

```
Accuracy: 1.0
Confusion Matrix:
[[47]]
Depth: 8
Number of nodes: 19
Number of leaves: 10

Results for iteration 2 with train size 85%:
Accuracy: 0.8936
Confusion Matrix:
[[ 3  4]
 [ 1 39]]
Depth: 6
Number of nodes: 31
Number of leaves: 16

Results for iteration 3 with train size 85%:
Accuracy: 0.9149
Confusion Matrix:
[[ 4  1]
 [ 3 39]]
Depth: 6
Number of nodes: 27
Number of leaves: 14

Results for iteration 4 with train size 85%:
Accuracy: 0.8936
Confusion Matrix:
[[ 3  3]
 [ 2 39]]
Depth: 6
Number of nodes: 31
Number of leaves: 16

Results for iteration 5 with train size 85%:
Accuracy: 0.8723
Confusion Matrix:
[[ 0  6]
 [ 0 41]]
Depth: 4
Number of nodes: 9
Number of leaves: 5

Results for iteration 6 with train size 85%:
Accuracy: 0.7447
Confusion Matrix:
[[ 0 11]
 [ 1 35]]
Depth: 8
Number of nodes: 29
Number of leaves: 15

Results for iteration 7 with train size 85%:
Accuracy: 0.8936
Confusion Matrix:
[[ 1  4]
 [ 1 41]]
Depth: 7
Number of nodes: 29
```

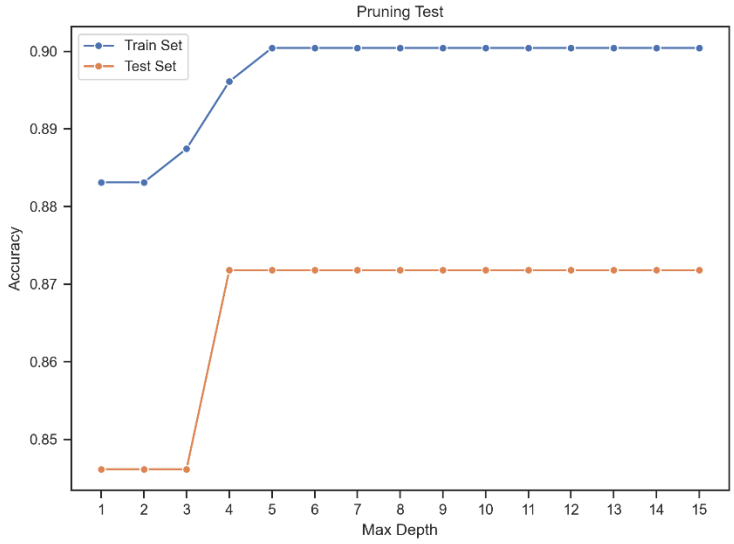
```
Number of leaves: 15
```

Average accuracy of 7 iterations with train size 85%: 0.8875 ± 0.0698

Part d

Maximum depth is a hyperparameter that controls the complexity of the tree. When its value increases, the accuracy increases for the training data most of the time. After some values, it doesn't have any impact on the accuracy because all of the leaves are pure. Increasing the maximum depth doesn't always improve the accuracy of the test data. We should find its optimal value through the chart.

```
Optimal max depth for Train set: 5
Optimal max depth for Test set: 4
Maximum accuracy for Train set: 0.9004
Maximum accuracy for Test set: 0.8718
```

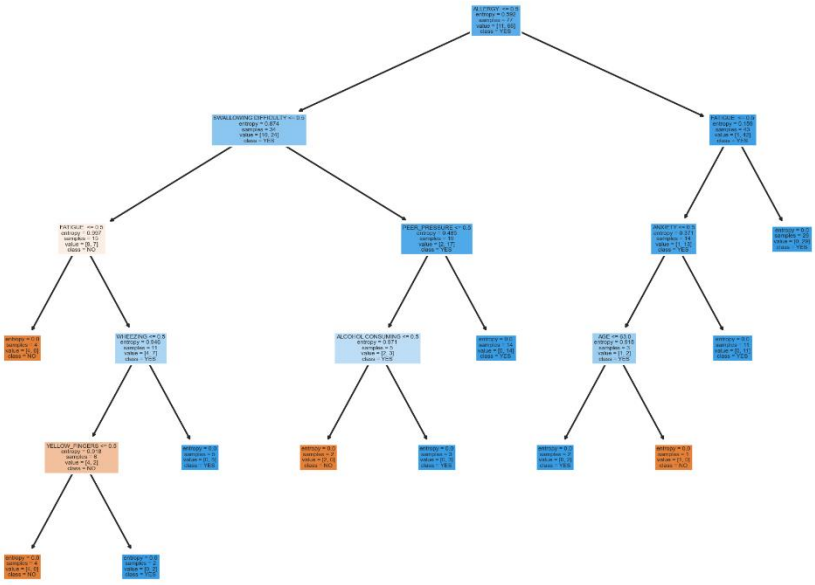


Part e

The results of our algorithm are very similar to sklearn's implementation results. In sklearn the best split for numeric data (Age feature) is found using a good algorithm, but we handled that by simply categorizing them with length 10.

Part e.a

```
Results for iteration 1 with train size 25%:
Accuracy: 0.8578
Confusion Matrix:
[[ 20   8]
 [ 25 179]]
Depth: 5
Number of nodes: 21
Number of leaves: 11
```



Part e.b

```
Results for iteration 1 with train size 25%:
Accuracy: 0.875
Confusion Matrix:
[[ 16  15]
 [ 14 187]]
Depth: 5
Number of nodes: 23
Number of leaves: 12

Results for iteration 2 with train size 25%:
Accuracy: 0.8922
Confusion Matrix:
[[ 16  12]
 [ 13 191]]
Depth: 6
Number of nodes: 23
Number of leaves: 12

Results for iteration 3 with train size 25%:
Accuracy: 0.8319
Confusion Matrix:
[[ 15  16]
 [ 23 178]]
Depth: 6
Number of nodes: 27
Number of leaves: 14

Results for iteration 4 with train size 25%:
Accuracy: 0.8621
Confusion Matrix:
[[ 15  12]
 [ 20 185]]
Depth: 5
```

```
Number of nodes: 17
Number of leaves: 9

Results for iteration 5 with train size 25%:
Accuracy: 0.8879
Confusion Matrix:
[[ 13  19]
 [  7 193]]
Depth: 5
Number of nodes: 15
Number of leaves: 8

Results for iteration 6 with train size 25%:
Accuracy: 0.8707
Confusion Matrix:
[[ 18   9]
 [ 21 184]]
Depth: 8
Number of nodes: 33
Number of leaves: 17

Results for iteration 7 with train size 25%:
Accuracy: 0.8405
Confusion Matrix:
[[ 17  17]
 [ 20 178]]
Depth: 5
Number of nodes: 25
Number of leaves: 13
```

Average accuracy of 7 iterations with train size 25%: 0.8658 ± 0.021

Part e.c

```
Results for iteration 1 with train size 45%:
Accuracy: 0.7765
Confusion Matrix:
[[  6  17]
 [ 21 126]]
Depth: 8
Number of nodes: 35
Number of leaves: 18

Results for iteration 2 with train size 45%:
Accuracy: 0.8647
Confusion Matrix:
[[  8  15]
 [  8 139]]
Depth: 10
Number of nodes: 43
Number of leaves: 22

Results for iteration 3 with train size 45%:
Accuracy: 0.8647
Confusion Matrix:
[[  9  12]
 [ 11 138]]
Depth: 11
```

```
Number of nodes: 39
Number of leaves: 20

Results for iteration 4 with train size 45%:
Accuracy: 0.8882
Confusion Matrix:
[[ 13   8]
 [ 11 138]]
Depth: 8
Number of nodes: 39
Number of leaves: 20

Results for iteration 5 with train size 45%:
Accuracy: 0.8529
Confusion Matrix:
[[  7  15]
 [ 10 138]]
Depth: 7
Number of nodes: 31
Number of leaves: 16

Results for iteration 6 with train size 45%:
Accuracy: 0.9
Confusion Matrix:
[[  9  10]
 [  7 144]]
Depth: 9
Number of nodes: 43
Number of leaves: 22

Results for iteration 7 with train size 45%:
Accuracy: 0.8647
Confusion Matrix:
[[ 11  14]
 [  9 136]]
Depth: 9
Number of nodes: 27
Number of leaves: 14
```

Average accuracy of 7 iterations with train size 45%: 0.8588 ± 0.0368

```
Results for iteration 1 with train size 65%:
Accuracy: 0.8532
Confusion Matrix:
[[11  5]
 [11 82]]
Depth: 7
Number of nodes: 49
Number of leaves: 25

Results for iteration 2 with train size 65%:
Accuracy: 0.7798
Confusion Matrix:
[[ 8 14]
 [10 77]]
Depth: 8
Number of nodes: 45
Number of leaves: 23
```

```
Results for iteration 3 with train size 65%:
Accuracy: 0.8257
Confusion Matrix:
[[ 5  9]
 [10 85]]
Depth: 7
Number of nodes: 43
Number of leaves: 22

Results for iteration 4 with train size 65%:
Accuracy: 0.8349
Confusion Matrix:
[[ 9 10]
 [ 8 82]]
Depth: 10
Number of nodes: 55
Number of leaves: 28

Results for iteration 5 with train size 65%:
Accuracy: 0.8624
Confusion Matrix:
[[ 8  4]
 [11 86]]
Depth: 11
Number of nodes: 55
Number of leaves: 28

Results for iteration 6 with train size 65%:
Accuracy: 0.8624
Confusion Matrix:
[[ 2  4]
 [11 92]]
Depth: 10
Number of nodes: 61
Number of leaves: 31

Results for iteration 7 with train size 65%:
Accuracy: 0.844
Confusion Matrix:
[[ 8 10]
 [ 7 84]]
Depth: 7
Number of nodes: 53
Number of leaves: 27
```

Average accuracy of 7 iterations with train size 65%: 0.8375 ± 0.0267

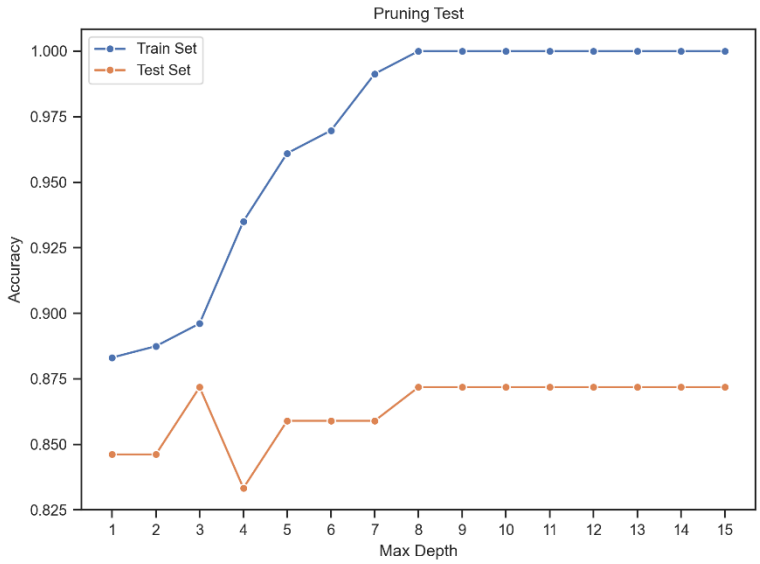
```
Results for iteration 1 with train size 85%:
Accuracy: 0.9149
Confusion Matrix:
[[ 4  3]
 [ 1 39]]
Depth: 9
Number of nodes: 65
Number of leaves: 33

Results for iteration 2 with train size 85%:
```

```
Accuracy: 0.8511
Confusion Matrix:
[[ 4  2]
 [ 5 36]]
Depth: 13
Number of nodes: 73
Number of leaves: 37

Results for iteration 3 with train size 85%:
Accuracy: 0.9149
Confusion Matrix:
[[ 4  1]
 [ 3 39]]
Depth: 8
Number of nodes: 51
Number of leaves: 26

Results for iteration 4 with train size 85%:
Accuracy: 0.9149
Confusion Matrix:
[[ 4  3]
 [ 1 39]]
Depth: 10
Number of nodes: 69
Number of leaves: 35

Results for iteration 5 with train size 85%:
Accuracy: 0.9149
Confusion Matrix:
[[ 5  1]
 [ 3 38]]
Depth: 9
Number of nodes: 75
Number of leaves: 38

Results for iteration 6 with train size 85%:
Accuracy: 0.8936
Confusion Matrix:
[[ 5  3]
 [ 2 37]]
Depth: 9
Number of nodes: 61
Number of leaves: 31

Results for iteration 7 with train size 85%:
Accuracy: 0.9149
Confusion Matrix:
[[ 5  2]
 [ 2 38]]
Depth: 10
Number of nodes: 73
Number of leaves: 37
```

Average accuracy of 7 iterations with train size 85%: 0.9027 ± 0.0223

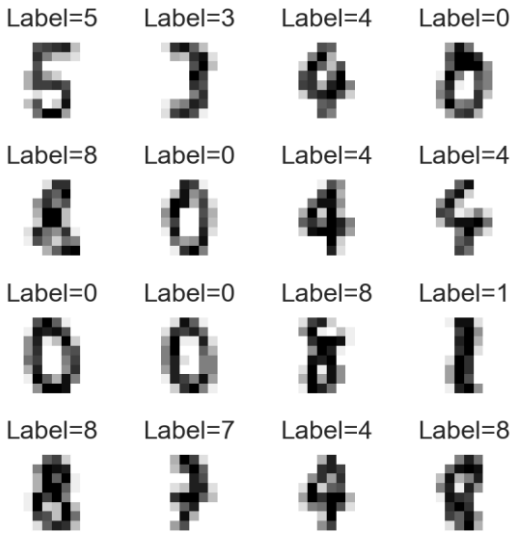
Part e.d

```
Optimal max depth for Train set: 8
Optimal max depth for Test set: 3
Maximum accuracy for Train set: 1.0
Maximum accuracy for Test set: 0.8718
```



Problem 4: Optical Recognition of Handwritten Digits

Part a



Part b

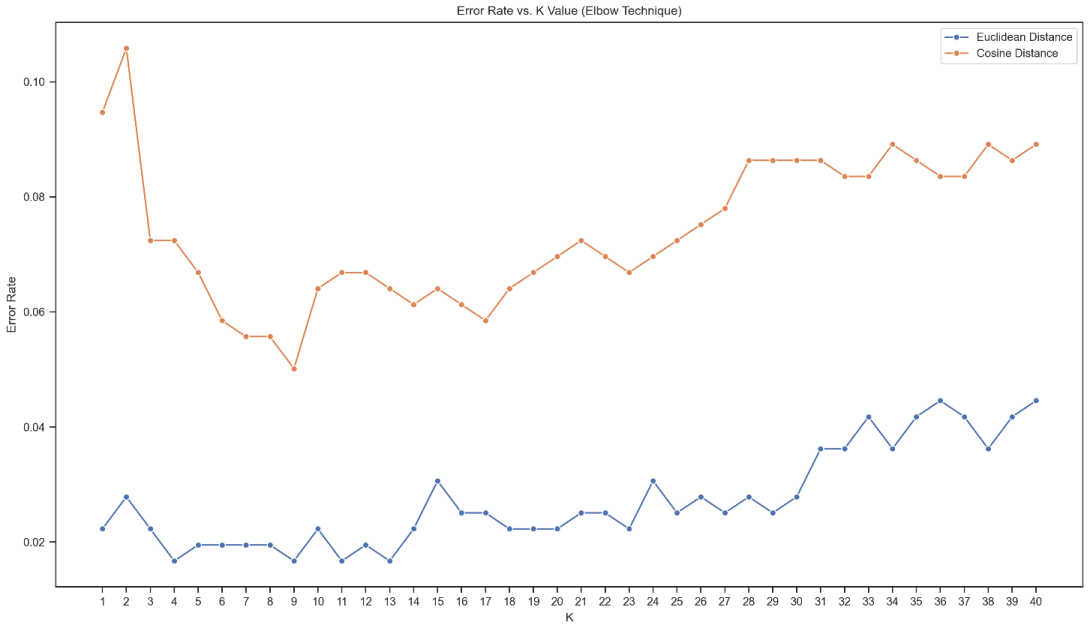
The implementation code has been attached.

Part c and d

According to the chart, the overall error rate of Euclidean distance is lower than the cosine distance in this application. The best values for k are as follows:

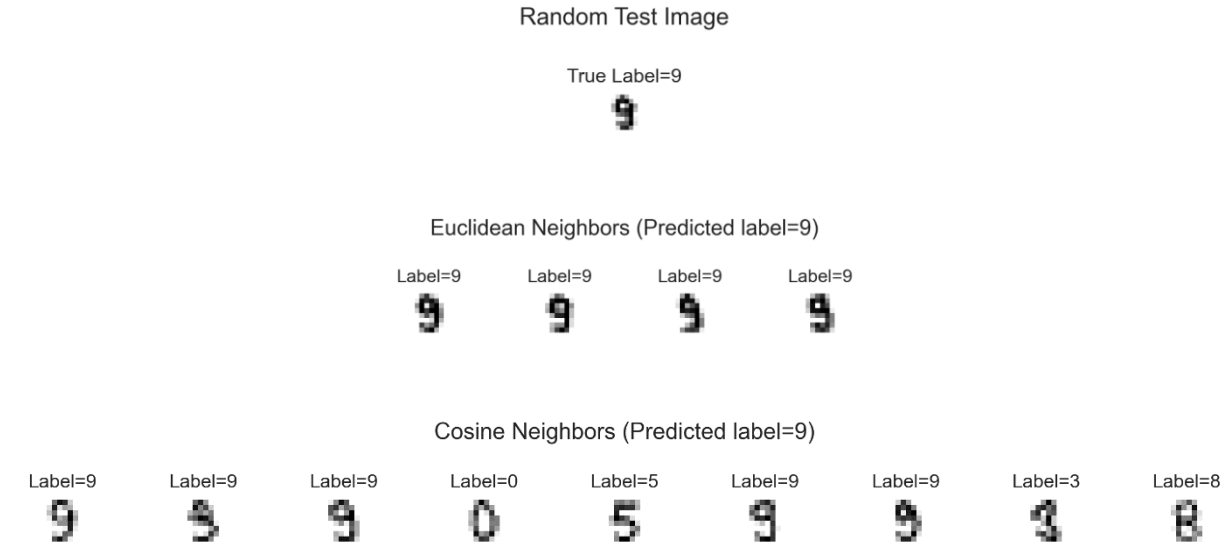
- Best k for Euclidean distance: 4
- Best k for Cosine distance: 9

Values higher than these lead to an overall error rate increment.



Part e

The randomly selected sample is 9. All 4 neighbors found using Euclidean distance have the correct label. Using the cosine distance, 4 of 9 found neighbors are indicating the wrong labels. Although the predicted label using these two metrics for this sample is correct, Euclidean distance is preferred. Because it selects the lower number of neighbors with higher similarities and the model is simpler.



Part f

```
Confusion Matrix for Euclidean Distance:
[[35  0  0  0  0  0  0  0  0  0]
 [ 0 37  0  0  0  0  0  0  0  0]
 [ 0  0 43  0  0  0  0  0  0  0]
 [ 0  0  0 37  0  0  0  0  0  0]
 [ 0  0  0  0 35  0  0  0  0  0]
 [ 0  0  0  0  0 29  0  0  0  1]
 [ 0  0  0  0  0  0 43  0  0  0]
 [ 0  0  0  0  0  0  0 31  0  0]
 [ 0  2  0  0  0  0  0  0 30  0]
 [ 0  0  0  0  0  1  0  0  2 33]]


Confusion Matrix for Cosine Distance:
[[35  0  0  0  0  0  0  0  0  0]
 [ 0 36  0  0  0  0  0  0  1  0]
 [ 0  0 43  0  0  0  0  0  0  0]
 [ 1  0  0 36  0  0  0  0  0  0]
 [ 0  0  0  0 34  0  0  1  0  0]
 [ 0  0  0  0  0 29  0  0  0  1]
 [ 1  0  0  0  0  0 42  0  0  0]
 [ 0  0  0  0  0  0  0 31  0  0]
 [ 1  1  0  5  0  0  0  0 24  1]
 [ 1  0  0  0  0  2  0  1  1 31]]
```

Part g

```
Euclidean Distance Performance for target 8:
TP: 30
TN: 329
FP: 2
FN: 2
F1-score: 0.9375
Precision: 0.9375
TPR (Recall, Sensitivity): 0.9375
TNR (Specificity): 0.994


Euclidean Distance Performance for target 3:
TP: 37
TN: 322
FP: 0
FN: 0
F1-score: 1.0
Precision: 1.0
TPR (Recall, Sensitivity): 1.0
TNR (Specificity): 1.0
```

Part h

```
Cosine Distance Performance for target 6:
TP: 42
TN: 317
FP: 0
FN: 1
F1-score: 0.9882
Precision: 1.0
TPR (Recall, Sensitivity): 0.9767
TNR (Specificity): 1.0

Cosine Distance Performance for target 4:
TP: 34
TN: 325
FP: 0
FN: 1
F1-score: 0.9855
Precision: 1.0
TPR (Recall, Sensitivity): 0.9714
TNR (Specificity): 1.0
```