Agreement 1) This assignment represents my own work. I did not work on this assignment with others. All coding was done by myself.
Agreement 2) I understand that if I struggle with this assignment that I will reevaluate whether this is the correct class for me to take. I understand that the homework only gets harder.

Q1

(1) Classification (Classify the players into different skill levels. Players are matched in the same level.)

(2) Clustering (This situation is image segmentation which can be solved by clustering.)

(3) Conditional probability estimation (The goal of this example is to find possible harmonies and avoid the most common one. The conditional probability estimation can achieve this goal.)

(4) Regression (Price for cat food is a continuous value, which can be predicted by the Regression method.)

(5) Density estimation (The goal of this example is to build a density distribution of standard transactions and monitor the new transactions. This can be done by Density estimation method.)

(6) Classification (In this case, the aim is to divide voters into four classes based on demographic data. This can be done by Classification method.)

(7) Ranking (The goal of this example is to build a search algorithm that ranks the search results based on a relevance score. Therefore, ranking method is suitable for this example.)

(8) Regression (The aim is to determine the quantity of vegetables to buy.

Because the quantity of vegetables is a continuous value, Regression method can be used.)

(9) Classification (The aim is to classify the posts into positive or negative. Classification is a suitable method to divide the posts into different classes.)

(10) Pattern mining (The goal of this example is to find correlations of genres and songs that users listening to from a large dataset. Pattern mining method is suitable for finding correlations in large dataset.)

Q2

(a) M1. Because M1 introduces non-linear components, which can increase model's accuracy.

(b) When using M3's model, overfitting occurs due to its complexity.

(c) Reduce the parameters of M3's model directly. This will reduce M3's model complexity.

Or add an additional regularization term to the objective function of M3's model. Thus, the complexity can be reduced automatically.

(d)

Calculate sensitivity, specificity, accuracy for M1, M2.

For M1:

Sensitivity = $\frac{TP}{\#Pos} = \frac{85}{85+15} = 85\%$

Specificity = $\frac{TN}{\#Neg} = \frac{890}{10+890} = 98.89\%$

Accuracy = $1 - \frac{FP+FN}{n} = 1 - \frac{10+15}{1000} = 97.5\%$

For M2:

Sensitivity = $\frac{TP}{\#Pos} = \frac{70}{70+30} = 70\%$

Specificity = $\frac{TN}{\#Neg} = \frac{892}{8+892} = 99.11\%$

Accuracy = $1 - \frac{FP+FN}{n} = 1 - \frac{8+30}{1000} = 96.2\%$

Would your answer to the first question change? Why or why not?

The answer is same as the first question. Because the sensitivity of M1 is significantly larger than that of M2, and the specificity and accuracy of the two are similar.

How to deal with imbalanced dataset?

Firstly, F1-score instead of accuracy can be employed to evaluate the model. Secondly, weighting method like introducing $C_{\text{imbalanced}}$ can be used to deal with the imbalanced dataset.

Q3

1.

$$N = \frac{1}{nm(\theta_0)}$$

If $\boldsymbol{\lambda < N}$:

Because $m(\theta_0) \geq 0$,

$$\lambda m(\theta_0) < \frac{1}{n} \tag{1}$$

We also have $\mathcal{L}(\theta_0) \leq \mathcal{L}(\theta)$, and $\mathcal{L}(\theta_\lambda) + \lambda m(\theta_\lambda) \leq \mathcal{L}(\theta) + \lambda m(\theta)$, for all θ. Therefore,

$$\mathcal{L}(\theta_0) \leq \mathcal{L}(\theta_\lambda) \tag{2}$$

$$\mathcal{L}(\theta_\lambda) + \lambda m(\theta_\lambda) \leq \mathcal{L}(\theta_0) + \lambda m(\theta_0) \tag{3}$$

Apply (1) and (3), we can get:

$$\mathcal{L}(\theta_\lambda) + \lambda m(\theta_\lambda) < \mathcal{L}(\theta_0) + \frac{1}{n} \tag{4}$$

Because $\lambda m(\theta_\lambda) > 0$, we can obtain:

$$\mathcal{L}(\theta_\lambda) < \mathcal{L}(\theta_0) + \frac{1}{n} \tag{5}$$

Combine (2) and (5):

$$\mathcal{L}(\theta_0) \leq \mathcal{L}(\theta_\lambda) < \mathcal{L}(\theta_0) + \frac{1}{n} \tag{6}$$

Because the 0-1 loss function, the smallest difference in $\mathcal{L}(\theta)$ is $\frac{1}{n}$. Therefore, $\mathcal{L}(\theta_0) = \mathcal{L}(\theta_\lambda)$.

If $\boldsymbol{\lambda \geq N}$:

Because $m(\theta_0) \geq 0$,

$$\lambda m(\theta_0) \geq \frac{1}{n} \tag{1}$$

Same as the case $\lambda < N$, we can get:

$$\mathcal{L}(\theta_0) \leq \mathcal{L}(\theta_\lambda) \tag{2}$$

$$\mathcal{L}(\theta_\lambda) + \lambda m(\theta_\lambda) \leq \mathcal{L}(\theta_0) + \lambda m(\theta_0) \tag{3}$$

From (1), we know that $\mathcal{L}(\theta_0) + \lambda m(\theta_0) \geq \mathcal{L}(\theta_0) + \frac{1}{n}$. Therefore, $\mathcal{L}(\theta_\lambda)$ can be larger than $\mathcal{L}(\theta_0) + \frac{1}{n}$. Because the smallest difference in $\mathcal{L}(\theta)$ is $\frac{1}{n}$, $\mathcal{L}(\theta_\lambda)$ can be different from $\mathcal{L}(\theta_0)$.

2.

We have $\mathcal{L}(\theta_1) + \lambda m(\theta_1) = \mathcal{L}(\theta_2) + \lambda m(\theta_2)$.

We also have $m(\theta_2) + 2 = m(\theta_1)$

Therefore, $\mathcal{L}(\theta_1) + \lambda(m(\theta_2) + 2) = \mathcal{L}(\theta_2) + \lambda m(\theta_2)$

Therefore, $\mathcal{L}(\theta_1) = \mathcal{L}(\theta_2) - 2\lambda$

3.

From the question, we know these equations below:

(1)$\lambda_3 > \lambda_4$

(2)$\mathcal{L}(\theta_4) < \mathcal{L}(\theta_3) + \epsilon$

(3)$\mathcal{L}(\theta_3) + \lambda_3 m(\theta_3) = \mathcal{L}(\theta_4) + \lambda_4 m(\theta_4)$

(4)$m(\theta_3) + z \geq m(\theta_4) \Rightarrow m(\theta_3) \geq m(\theta_4) - z$

From (2) and (3), we can derive:

(5) $\lambda_3 m(\theta_3) - \lambda_4 m(\theta_4) < \epsilon \Rightarrow m(\theta_3) < \frac{\epsilon+\lambda_4 m(\theta_4)}{\lambda_3}$

From (4) and (5), we can derive:

$$m(\theta_4) - z \leq m(\theta_3) < \frac{\epsilon + \lambda_4 m(\theta_4)}{\lambda_3}$$

Therefore, $m(\theta_4) - z < \frac{\epsilon+\lambda_4 m(\theta_4)}{\lambda_3} \Rightarrow (\lambda_3 - \lambda_4) m(\theta_4) < \epsilon + \lambda_3 z$

From (1), we know that $\lambda_3 - \lambda_4 > 0$

Therefore, $m(\theta_4) < \frac{\epsilon+\lambda_3 z}{\lambda_3-\lambda_4}$

Therefore, $function(\lambda_3, \lambda_4, \epsilon, z) = \frac{\epsilon+\lambda_3 z}{\lambda_3-\lambda_4}$

Q4

(a)

The value of g(x) is shown in the code below. The largest threshold value is 1.922.

(b)

The value of f(x) is shown in the code below. The largest threshold value is 0.95808178. The confusion matrix is $\begin{bmatrix} 3 & 0 \\ 2 & 5 \end{bmatrix}$, the precision is 1, the recall is 0.6, and the F1 score is 0.75.

(c)

Plotted in the code below.

(d)

AUC of $g(x)$ is 0.76.

AUC of $f(x)$ is 0.76.

The intact code and answers for Q4 are shown below.

HW1_Q4

September 9, 2023

1 Q4 Classifiers and Metrics

1.1 Initialization

[1]:

```
import numpy as np
import matplotlib.pyplot as plt

# feature vector x including Age, likeRowing, Experience, and Income
x = np.array([
    [20, 1, 0, 20],
    [18, 1, 1, 33],
    [11, 0, 1, 21],
    [31, 0, 0, 9],
    [22, 1, 1, 7],
    [21, 1, 0, 10],
    [13, 1, 0, 23],
    [15, 1, 1, 16],
    [16, 0, 1, 15],
    [17, 1, 0, 6]
])

# binary label y
y = np.array([
    [0],
    [0],
    [1],
    [1],
    [1],
    [0],
    [1],
    [0],
    [1],
    [0]
])

# parameters
theta = np.array([[0.05, -3, 2.1, 0.008]])
theta0 =  0.3
```

1.2 (a)

```
[2]: # linear classifier g(x)
g = x @ theta.T + theta0
print('g(x)=\n', g)

# select the largest threshold value
g_binary = np.zeros((10, 1))
threshold = -2
error_min_g = 10
threshold_max = -2
while threshold <= 4:
    g_binary[g < threshold] = 0
    g_binary[g >= threshold] = 1
    error = np.sum(np.abs(g_binary - y))
    if error_min_g >= error:
        error_min_g = error
        threshold_max = threshold
    threshold += 0.001
print('minimized (mis)classification error for g(x) =', error_min_g, '; largest␣
 ↪threshold value for g(x) =', format(threshold_max, '.3f'))
```

```
g(x)=
 [[-1.54 ]
 [ 0.564]
 [ 3.118]
 [ 1.922]
 [ 0.556]
 [-1.57 ]
 [-1.866]
 [ 0.278]
 [ 3.32 ]
 [-1.802]]
minimized (mis)classification error for g(x) = 2.0 ; largest threshold value for
g(x) = 1.922
```

1.3 (b)

```
[3]: # define tanh
def tanh(z):
    return (np.exp(z) - np.exp(-z)) / (np.exp(z) + np.exp(-z))

# non-linear classifier f(x)
f = tanh(x @ theta.T + theta0)
print('f(x)=\n', f)

# select the largest threshold value
f_binary = np.zeros((10, 1))
```

```
f_binary_for_max_threshold = np.zeros((10, 1))
threshold = -1
error_min_f = 10
threshold_max = -1
while threshold <= 1:
    f_binary[f < threshold] = 0
    f_binary[f >= threshold] = 1
    error = np.sum(np.abs(f_binary - y))
    if error_min_f >= error:
        error_min_f = error
        threshold_max = threshold
        f_binary_for_max_threshold = f_binary.copy()
    threshold += 0.001
print('minimized (mis)classification error for f(x) =', error_min_f, '; largest␣
 ↪threshold value for f(x) =', format(threshold_max, '.3f'))

# confusion matrix
confusion_matrix = np.zeros((2, 2))
for i in range(10):
    # print(y[i], f_binary_for_max_threshold[i])
    if y[i] == 1 and f_binary_for_max_threshold[i] == 1:
        confusion_matrix[0][0] += 1
    elif y[i] == 1 and f_binary_for_max_threshold[i] == 0:
        confusion_matrix[1][0] += 1
    elif y[i] == 0 and f_binary_for_max_threshold[i] == 1:
        confusion_matrix[0][1] += 1
    elif y[i] == 0 and f_binary_for_max_threshold[i] == 0:
        confusion_matrix[1][1] += 1
print('confusion matrix=\n', confusion_matrix)
# precision
precision = confusion_matrix[0][0] / (confusion_matrix[0][0] +␣
 ↪confusion_matrix[0][1])
print('precision =', precision)
# recall
recall = confusion_matrix[0][0] / (confusion_matrix[0][0] +␣
 ↪confusion_matrix[1][0])
print('recall =', recall)
# F1 score
F1 = 2 * precision * recall / (precision + recall)
print('F1 score =', F1)
```

```
f(x)=
 [[-0.91212037]
 [ 0.51093923]
 [ 0.99609231]
 [ 0.95808178]
 [ 0.50500357]
 [-0.91702576]
 [-0.9532301 ]
 [ 0.27105303]
 [ 0.99738936]
 [-0.94701274]]
minimized (mis)classification error for f(x) = 2.0 ; largest threshold value for
f(x) = 0.958
confusion matrix=
 [[3. 0.]
 [2. 5.]]
precision = 1.0
recall = 0.6
F1 score = 0.7499999999999999
```

1.4 (c)

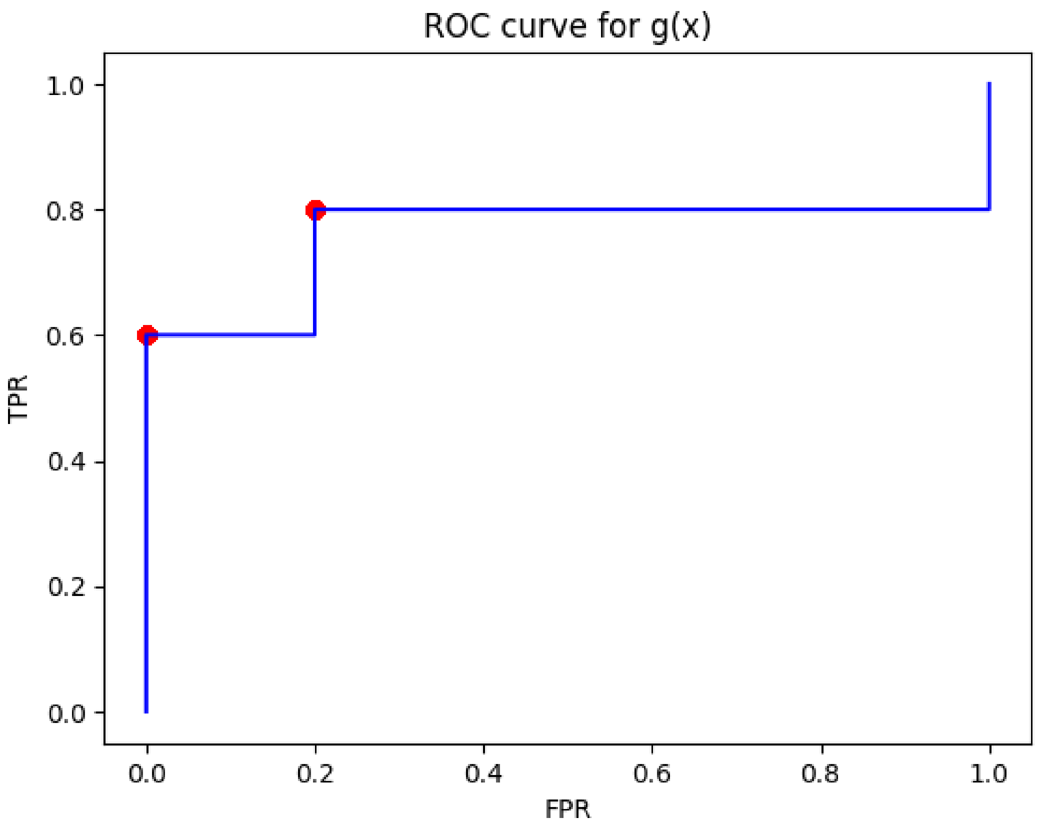
[4]:
```
# plot ROC curve for g(x)
decision_boundary = -2
g_predicted = np.zeros((10, 1))
FPRs_g = []
TPRs_g = []
FPRs_min_err_g = []
TPRs_min_err_g = []
while decision_boundary <= 4:
    g_predicted[g < decision_boundary] = 0
    g_predicted[g >= decision_boundary] = 1
    confusion_matrix_g = np.zeros((2, 2))
    for i in range(10):
        if y[i] == 1 and g_predicted[i] == 1:
            confusion_matrix_g[0][0] += 1
        elif y[i] == 1 and g_predicted[i] == 0:
            confusion_matrix_g[1][0] += 1
        elif y[i] == 0 and g_predicted[i] == 1:
            confusion_matrix_g[0][1] += 1
        elif y[i] == 0 and g_predicted[i] == 0:
            confusion_matrix_g[1][1] += 1
    decision_boundary += 0.001
    FPR_g = confusion_matrix_g[0][1] / (confusion_matrix_g[0][1] +␣
 ↪confusion_matrix_g[1][1])
    TPR_g = confusion_matrix_g[0][0] / (confusion_matrix_g[0][0] +␣
 ↪confusion_matrix_g[1][0])
    FPRs_g.append(FPR_g)
    TPRs_g.append(TPR_g)
    err = np.sum(np.abs(g_predicted - y))
    if err == error_min_g:
        FPRs_min_err_g.append(FPR_g)
        TPRs_min_err_g.append(TPR_g)
```

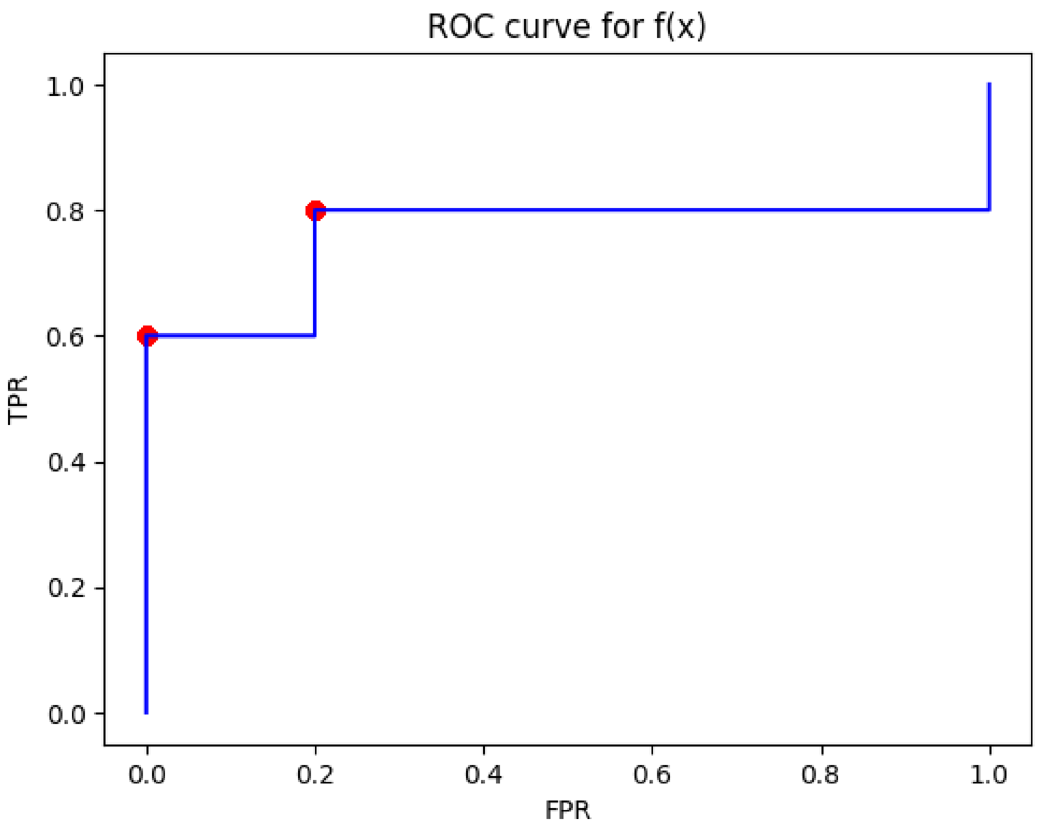
```
# plot the points that represent decision points with the minimum␣
 ↪classification error
plt.scatter(FPRs_min_err_g, TPRs_min_err_g, color='r')
# plot ROC curve
plt.plot(FPRs_g, TPRs_g, color='b')
plt.title('ROC curve for g(x)')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.show()

###########################################
# plot ROC curve for f(x)
decision_boundary = -1
f_predicted = np.zeros((10, 1))
FPRs_f = []
TPRs_f = []
FPRs_min_err_f = []
TPRs_min_err_f = []
while decision_boundary <= 1:
    f_predicted[f < decision_boundary] = 0
    f_predicted[f >= decision_boundary] = 1
    confusion_matrix_f = np.zeros((2, 2))
    for i in range(10):
        if y[i] == 1 and f_predicted[i] == 1:
            confusion_matrix_f[0][0] += 1
        elif y[i] == 1 and f_predicted[i] == 0:
            confusion_matrix_f[1][0] += 1
        elif y[i] == 0 and f_predicted[i] == 1:
            confusion_matrix_f[0][1] += 1
        elif y[i] == 0 and f_predicted[i] == 0:
            confusion_matrix_f[1][1] += 1
    decision_boundary += 0.001
    FPR_f = confusion_matrix_f[0][1] / (confusion_matrix_f[0][1] +␣
 ↪confusion_matrix_f[1][1])
    TPR_f = confusion_matrix_f[0][0] / (confusion_matrix_f[0][0] +␣
 ↪confusion_matrix_f[1][0])
    FPRs_f.append(FPR_f)
    TPRs_f.append(TPR_f)
    err = np.sum(np.abs(f_predicted - y))
    if err == error_min_f:
        FPRs_min_err_f.append(FPR_f)
        TPRs_min_err_f.append(TPR_f)

# plot the points that represent decision points with the minimum␣
 ↪classification error
plt.scatter(FPRs_min_err_f, TPRs_min_err_f, color='r')
```

```
# plot ROC curve
plt.plot(FPRs_f, TPRs_f, color='b')
plt.title('ROC curve for f(x)')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.show()
```





1.5 (d)

[5]:
```python
# AUC for g(x) and f(x)
y_pos_ind = np.where(y == 1)[0]
y_neg_ind = np.where(y == 0)[0]
AUC_g = 0
AUC_f = 0
for i in y_pos_ind:
    for k in y_neg_ind:
        if g[i][0] > g[k][0]:
            AUC_g += 1
        if f[i][0] > f[k][0]:
            AUC_f += 1
AUC_g /= len(y_pos_ind) * len(y_neg_ind)
AUC_f /= len(y_pos_ind) * len(y_neg_ind)
print('AUC of g(x) = ', AUC_g)
print('AUC of f(x) = ', AUC_f)
```

```
AUC of g(x) =  0.76
AUC of f(x) =  0.76
```

Q5

(a)

F1 score. Because F1 score takes into account precision and recall. F1 score can provide a more useful measure than accuracy when the data is imbalanced.

(b)

The k-NN algorithm is implemented in the code below. The F1 score on the test set is 0.933.

(c)

Plotted in the code below.

When k is 3 and distance function is Euclidean distance, the performance of the model is the best.

(d)

Using the best parameters determined in (c), the F1 score on the test set is 0.964, which is better than the one in (b). This result meets my expectation.

The intact code and answers for Q5 are shown below.

HW1_Q5

September 11, 2023

1 Q5 K-Nearest Neighbors with Parameter Tuning

1.1 Initialization

[1]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# get train and test data from file using pandas
train_data_pd = pd.read_csv('breast_cancer_train.csv')
test_data_pd = pd.read_csv('breast_cancer_test.csv')
train_data = train_data_pd.to_numpy()
test_data = test_data_pd.to_numpy()
train_x = train_data[:, :30]
train_y = train_data[:, 30:]
test_x = test_data[:, :30]
test_y = test_data[:, 30:]
```

1.2 (b)

[2]:
```python
# normalization
train_x_max = np.max(train_x, axis=0)
train_x_min = np.min(train_x, axis=0)
normalized_train_x = (train_x - train_x_min) / (train_x_max - train_x_min)
normalized_test_x = (test_x - train_x_min) / (train_x_max - train_x_min)

# KNN
k = 31
predicted_y = np.zeros(test_y.shape)
n = 0
for i in normalized_test_x:
    # Euclidean distance
    dists = [np.sqrt(np.sum((i - j) ** 2)) for j in normalized_train_x]
    # print(len(dists))
    knn_ind = np.argsort(dists)[:k]
    knn_y = train_y[knn_ind]
    if len(knn_y[knn_y == 1]) > len(knn_y[knn_y == 0]):
        predicted_y[n, 0] = 1
    else:
        predicted_y[n, 0] = 0
    n += 1
# print(predicted_y)

# confusion matrix
confusion_matrix = np.zeros((2, 2))
for i in range(test_y.shape[0]):
    if test_y[i] == 1 and predicted_y[i] == 1:
        confusion_matrix[0][0] += 1
    elif test_y[i] == 1 and predicted_y[i] == 0:
        confusion_matrix[1][0] += 1
    elif test_y[i] == 0 and predicted_y[i] == 1:
        confusion_matrix[0][1] += 1
    elif test_y[i] == 0 and predicted_y[i] == 0:
        confusion_matrix[1][1] += 1
# precision
precision = confusion_matrix[0][0] / (confusion_matrix[0][0] +␣
 ↪confusion_matrix[0][1])
# recall
recall = confusion_matrix[0][0] / (confusion_matrix[0][0] +␣
 ↪confusion_matrix[1][0])
# F1 score
F1 = 2 * precision * recall / (precision + recall)
print('F1 score =', F1)
```

```
F1 score = 0.9333333333333333
```

1.3 (c)

[3]:

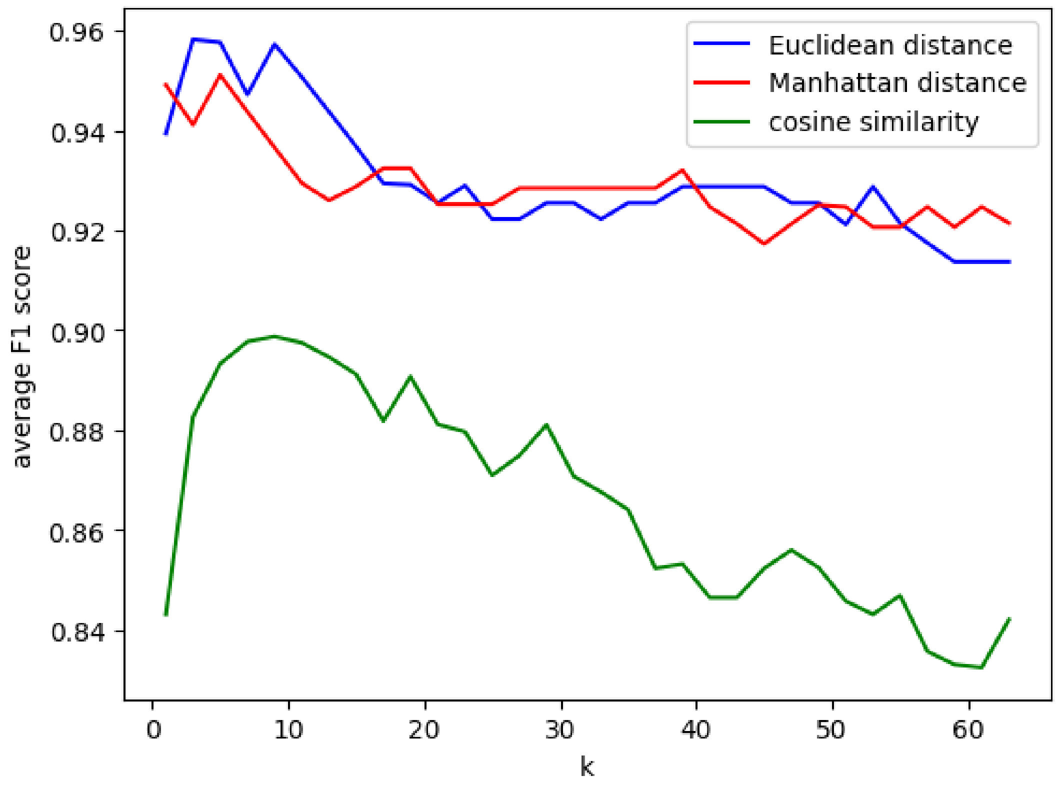
```
# split normalized_train_x into five folds
folds_x = np.array_split(train_x, 5)
folds_y = np.array_split(train_y, 5)

# select best pair of parameters
F1_best = -1
k_best = -1
distance_method_best = -1

ks = np.linspace(1, 63, 32, dtype='int32')
# ks = np.linspace(1, 63, 4, dtype='int32')
# distance_methods = [0]
distance_methods = [0, 1, 2]
# tuning distance method
for distance_method in distance_methods:
    F1_avgs = []
    # tuning k
    for k in ks:
        F1_sum = 0;
        for episode in range(5):
            eval_x = folds_x[episode]
            eval_y = folds_y[episode]
            tra_x = []
            tra_y = []
            predicted_y = np.zeros(eval_y.shape)
            for episode2 in range(5):
                if episode != episode2:
                    tra_x.append(folds_x[episode2])
                    tra_y.append(folds_y[episode2])
            tra_x = np.array(tra_x).reshape(4 * eval_x.shape[0], eval_x.
↪shape[1])
            tra_y = np.array(tra_y).reshape(4 * eval_y.shape[0], eval_y.
↪shape[1])
            # normalization
            tra_x_max = np.max(tra_x, axis=0)
            tra_x_min = np.min(tra_x, axis=0)
            normalized_tra_x = (tra_x - tra_x_min) / (tra_x_max - tra_x_min)
            normalized_eval_x = (eval_x - tra_x_min) / (tra_x_max - tra_x_min)
            n = 0
            for i in normalized_eval_x:
                if distance_method == 0:
                    # Euclidean distance
                    dists = [np.sqrt(np.sum((i - j) ** 2)) for j in␣
↪normalized_tra_x]
                elif distance_method == 1:
                    # Manhattan distance
                    dists = [np.sum(np.abs(i - j)) for j in normalized_tra_x]
                elif distance_method == 2:
                    # cosine similarity
                    dists = [1 - (i.T @ j) / (np.sqrt(np.sum(i ** 2)) * np.
↪sqrt(np.sum(j ** 2))) for j in normalized_tra_x]
                knn_ind = np.argsort(dists)[:k]
                knn_y = tra_y[knn_ind]
                if len(knn_y[knn_y == 1]) > len(knn_y[knn_y == 0]):
                    predicted_y[n, 0] = 1
                else:
                    predicted_y[n, 0] = 0
                n += 1
            # print(predicted_y.shape)
            # confusion matrix
            confusion_matrix = np.zeros((2, 2))
            for i in range(eval_y.shape[0]):
                if eval_y[i] == 1 and predicted_y[i] == 1:
                    confusion_matrix[0][0] += 1
                elif eval_y[i] == 1 and predicted_y[i] == 0:
                    confusion_matrix[1][0] += 1
                elif eval_y[i] == 0 and predicted_y[i] == 1:
                    confusion_matrix[0][1] += 1
                elif eval_y[i] == 0 and predicted_y[i] == 0:
                    confusion_matrix[1][1] += 1
            precision = confusion_matrix[0][0] / (confusion_matrix[0][0] +␣
 ↪confusion_matrix[0][1])
            recall = confusion_matrix[0][0] / (confusion_matrix[0][0] +␣
 ↪confusion_matrix[1][0])
            F1 = 2 * precision * recall / (precision + recall)
            F1_sum += F1
            # print(F1)
        F1_avg = F1_sum / 5
        F1_avgs.append(F1_avg)
        if F1_best < F1_avg:
            F1_best = F1_avg
            k_best = k
            distance_method_best = distance_method
        # print('average F1 score =', F1_avg)
    # print(ks, F1_avgs)
    if distance_method == 0:
        plt.plot(ks, F1_avgs, color='b')
    if distance_method == 1:
        plt.plot(ks, F1_avgs, color='r')
    if distance_method == 2:
        plt.plot(ks, F1_avgs, color='g')

print('best average F1 score = ', F1_best)
print('best k = ', k_best)
if distance_method_best == 0:
    print('best distance method = Euclidean distance')
elif distance_method_best == 1:
    print('best distance method = Manhattan distance')
elif distance_method_best == 2:
    print('best distance method = cosine similarity')
plt.xlabel('k')
plt.ylabel('average F1 score')
plt.legend(['Euclidean distance', 'Manhattan distance', 'cosine similarity'])
plt.show()
```

```
best average F1 score =  0.95818197149844
best k =  3
best distance method = Euclidean distance
```



1.4 (d)

[4]:
```
# KNN
k = 3 # best k is 3
predicted_y = np.zeros(test_y.shape)
n = 0
for i in normalized_test_x:
    # best distance method is Euclidean distance
    dists = [np.sqrt(np.sum((i - j) ** 2)) for j in normalized_train_x]
    knn_ind = np.argsort(dists)[:k]
    knn_y = train_y[knn_ind]
    if len(knn_y[knn_y == 1]) > len(knn_y[knn_y == 0]):
        predicted_y[n, 0] = 1
    else:
        predicted_y[n, 0] = 0
    n += 1

# confusion matrix
confusion_matrix = np.zeros((2, 2))
for i in range(test_y.shape[0]):
    if test_y[i] == 1 and predicted_y[i] == 1:
        confusion_matrix[0][0] += 1
    elif test_y[i] == 1 and predicted_y[i] == 0:
        confusion_matrix[1][0] += 1
    elif test_y[i] == 0 and predicted_y[i] == 1:
        confusion_matrix[0][1] += 1
    elif test_y[i] == 0 and predicted_y[i] == 0:
        confusion_matrix[1][1] += 1
precision = confusion_matrix[0][0] / (confusion_matrix[0][0] +␣
 ↪confusion_matrix[0][1])
recall = confusion_matrix[0][0] / (confusion_matrix[0][0] +␣
 ↪confusion_matrix[1][0])
F1 = 2 * precision * recall / (precision + recall)
print('F1 score =', F1)
```

```
F1 score = 0.9636363636363636
```

```
[ ]:
```