HW1_Q5

September 10, 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
```

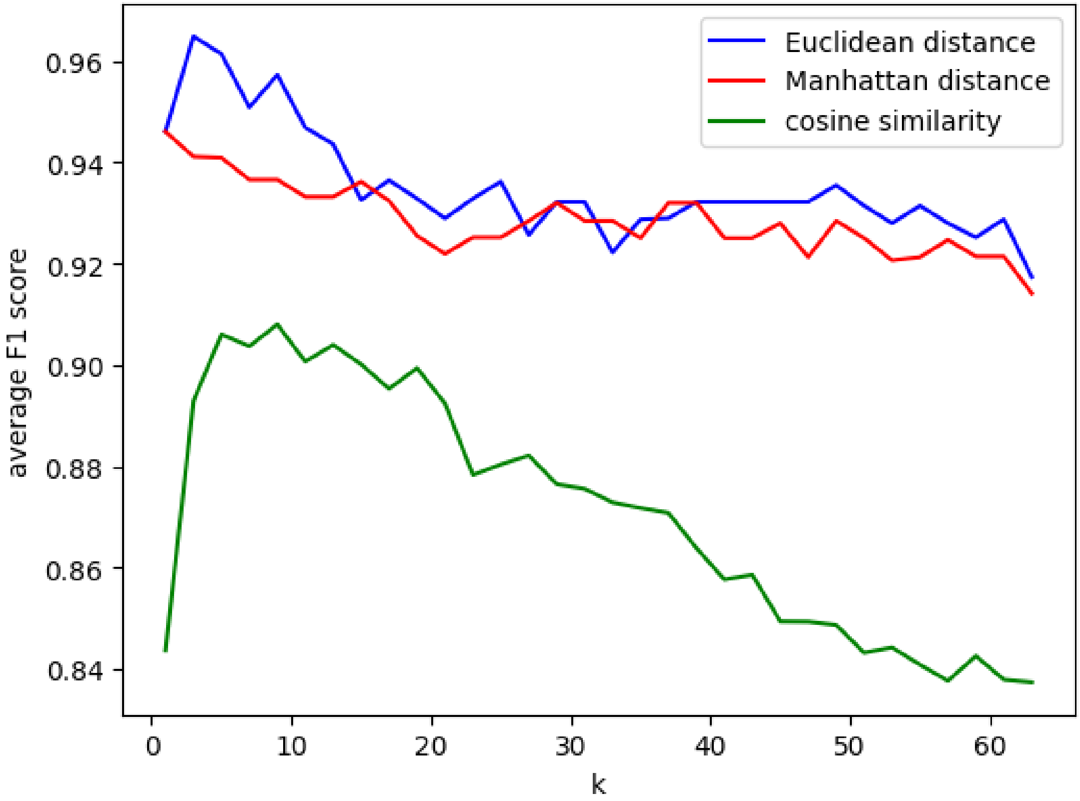
1.3 (c)

[3]:

```
# split normalized_train_x into five folds
folds_x = np.array_split(normalized_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')
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])
            n = 0
            for i in eval_x:
                if distance_method == 0:
                    # Euclidean distance
                    dists = [np.sqrt(np.sum((i - j) ** 2)) for j in tra_x]
                elif distance_method == 1:
                    # Manhattan distance
                    dists = [np.sum(np.abs(i - j)) for j in 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 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 =  0.9648454636091724
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
```