k_nn_classifier

May 5, 2022

1 Libraries

[68]:
```
import pandas as pd
import os
import numpy as np
from collections import Counter
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
```

2 Read dataset

feature vector X = {f1, f2, f3, f4, f5, f6}

class vector C = {"satisfied", "unsatisfied"}

[69]:
```
# load satisfaction_data.csv from the local path
directory_data = './'
filename_data = 'satisfaction_data.csv'
df = pd.read_csv(os.path.join(directory_data, filename_data), header=None)
dataset = df.to_numpy() # pandas dataframe -> numpy array

print("Examples of dataset : \n", dataset[0:3])
```

```
Examples of dataset :
 [[40 2 1 86872 25 9 'unsatisfied']
 [40 2 1 259323 54 10 'satisfied']
 [40 2 1 256813 43 14 'satisfied']]
```

3 Generate 10 different train/test dataset pairs (k fold cross validation)

train data : test data = 9 : 1

[70]:
```
fold_size = int(len(dataset) / 10)
fold_num = 10
```

```
X_dataset = dataset[:,:6]
y_dataset = dataset[:,6]

X_train = []
y_train = []
X_test = []
y_test = []

for i in range(fold_num-1):
    X_test.append(X_dataset[i*fold_size:(i+1)*fold_size])
    y_test.append(y_dataset[i*fold_size:(i+1)*fold_size])

    xa = X_dataset[:i*fold_size]
    xb = X_dataset[(i+1)*fold_size:]

    ya = y_dataset[:i*fold_size]
    yb = y_dataset[(i+1)*fold_size:]

    X_train.append(np.concatenate((xa,xb)))
    y_train.append(np.concatenate((ya,yb)))

X_test.append(X_dataset[9*fold_size:])
y_test.append(y_dataset[9*fold_size:])

X_train.append(X_dataset[:9*fold_size])
y_train.append(y_dataset[:9*fold_size])


print("Shape of X_test", X_test[8].shape)
print("Shape of y_test", y_test[8].shape)

print("Shape of X_train", X_train[8].shape)
print("Shape of y_train", y_train[8].shape)

print("Shape of X_test", X_test[9].shape)
print("Shape of y_test", y_test[9].shape)

print("Shape of X_train", X_train[9].shape)
print("Shape of y_train", y_train[9].shape)
```

4 Data preprocessing (normalize)

[71]:
```
col_means = []
col_std = []

for i in range(fold_num):
    col_means.append(X_train[i].sum(axis = 0) / len(X_train[i]))
    col_std.append(np.std(X_train[i], dtype=np.float64, axis = 0))

    normalized_train_data = (X_train[i] - col_means[i])/col_std[i]
    X_train[i] = normalized_train_data

    normalized_test_data = (X_test[i] - col_means[i])/col_std[i]
    X_test[i] = normalized_test_data

print("Examples of normalized train dataset : \n", X_train[i][0:2])
print("Examples of normalized test dataset : \n", X_test[i][0:2])
```

```
Examples of normalized train dataset :
 [[-0.2666224404400283 -0.31300994728093356 -0.3067523904747627
  -0.9654140175900487 -1.1204620079779688 -0.5308974550386716]
 [-0.2666224404400283 -0.31300994728093356 -0.3067523904747627
  0.7326949552864844 1.1328335502574762 -0.12397554264316428]]
Examples of normalized test dataset :
 [[-0.8577094026190303 -1.1899249746827711 -2.077612887121886
  -1.3669511945038344 0.2004353882290163 1.0967901945433576]
 [0.5777875055299745 1.4408201075227416 -0.3067523904747627
  -0.43912637487802 1.3659330907645912 -0.5308974550386716]]
```

5 kNN model

distance between two data is defined as Euclidean(L2 norm)

[72]:
```
class KNN:
  def __init__(self, k):
    self.k = k

  def fit(self, X, y):
    self.X_train = X
    self.y_train = y

  # distance
  def distance(self, data1, data2) :
    sub = data1 - data2
    dis = np.sum(np.square(sub)) ** 0.5
```

```
    # print("data1 : ", data1)
    # print("data2 : ", data2)
    # print("sub", sub)
    # print("dis", dis)
    return dis


  def predict(self, _X_test):
    final_output = []

    for i in range(len(_X_test)):
        if i % 300 == 0 :
          print("   Loading : ", i/len(_X_test)*100, "%")

        d = []
        votes = []

        for j in range(len(self.X_train)):
            # get distance with every data samples
            dist = self.distance(_X_test[i] , self.X_train[j])
            d.append([dist, j])

        d.sort()
        d = d[0:self.k]

        # vote
        for d, j in d:
            votes.append(self.y_train[j])
        ans = Counter(votes).most_common(1)[0][0]
        final_output.append(ans)

    return final_output
```

6 Predict satisfaction with 10 dataset pairs

[73]:
```
# set k-NN model
window_size = 5
clf = KNN(window_size)

prediction_arr = []

for i in range(fold_num):
    print("Predict with train/test pair ", i)

    clf.fit(X_train[i], y_train[i])
```

```
    # predict
    prediction = clf.predict(X_test[i])
    prediction_arr = np.concatenate((prediction_arr,prediction))

    ground_truth = y_test[i]
    # print(prediction)
    # print(ground_truth)
    # prediction loss
    accuracy_score = np.sum(prediction == ground_truth) / len(ground_truth)
    print("       Accuracy_score : ", accuracy_score)


# export preiction result .csv
df = pd.DataFrame(X_dataset)
df.insert(6,"class" ,prediction_arr)
df.to_csv(r'./20174089.csv', index = False)
```

```
Predict with train/test pair  0
   Loading :  0.0 %
   Loading :  15.0 %
   Loading :  30.0 %
   Loading :  45.0 %
   Loading :  60.0 %
   Loading :  75.0 %
   Loading :  90.0 %
       Accuracy_score :  0.753
Predict with train/test pair  1
   Loading :  0.0 %
   Loading :  15.0 %
   Loading :  30.0 %
   Loading :  45.0 %
   Loading :  60.0 %
   Loading :  75.0 %
   Loading :  90.0 %
       Accuracy_score :  0.749
Predict with train/test pair  2
   Loading :  0.0 %
   Loading :  15.0 %
   Loading :  30.0 %
   Loading :  45.0 %
   Loading :  60.0 %
   Loading :  75.0 %
   Loading :  90.0 %
       Accuracy_score :  0.7505
Predict with train/test pair  3
   Loading :  0.0 %
```

```
   Loading :  15.0 %
   Loading :  30.0 %
   Loading :  45.0 %
   Loading :  60.0 %
   Loading :  75.0 %
   Loading :  90.0 %
       Accuracy_score :  0.736
Predict with train/test pair  4
   Loading :  0.0 %
   Loading :  15.0 %
   Loading :  30.0 %
   Loading :  45.0 %
   Loading :  60.0 %
   Loading :  75.0 %
   Loading :  90.0 %
       Accuracy_score :  0.75
Predict with train/test pair  5
   Loading :  0.0 %
   Loading :  15.0 %
   Loading :  30.0 %
   Loading :  45.0 %
   Loading :  60.0 %
   Loading :  75.0 %
   Loading :  90.0 %
       Accuracy_score :  0.7605
Predict with train/test pair  6
   Loading :  0.0 %
   Loading :  15.0 %
   Loading :  30.0 %
   Loading :  45.0 %
   Loading :  60.0 %
   Loading :  75.0 %
   Loading :  90.0 %
       Accuracy_score :  0.757
Predict with train/test pair  7
   Loading :  0.0 %
   Loading :  15.0 %
   Loading :  30.0 %
   Loading :  45.0 %
   Loading :  60.0 %
   Loading :  75.0 %
   Loading :  90.0 %
       Accuracy_score :  0.75
Predict with train/test pair  8
   Loading :  0.0 %
   Loading :  15.0 %
   Loading :  30.0 %
   Loading :  45.0 %
```

```
   Loading :  60.0 %
   Loading :  75.0 %
   Loading :  90.0 %
       Accuracy_score :  0.7445
Predict with train/test pair  9
   Loading :  0.0 %
   Loading :  15.0 %
   Loading :  30.0 %
   Loading :  45.0 %
   Loading :  60.0 %
   Loading :  75.0 %
   Loading :  90.0 %
       Accuracy_score :  0.759
```

[74]:
```
final_accuracy_score = np.sum(prediction_arr == y_dataset) / len(prediction_arr)
print("Fianl accuracy_score : ", final_accuracy_score)
```

```
Fianl accuracy_score :  0.75095
```