HomeWork 04

Deep Dand

May 15, 2018

Abstract

KNN Regressor

1 Problem 1

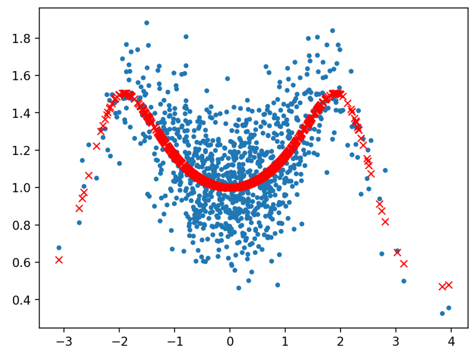
1.1 a

You will modify the python code below to generate 1000 data points.
The code below generates 1000 data points.
Code Begins

```
import numpy as np
from matplotlib import pyplot as plt
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import KFold

def genDataSet(N):
    x = np.random.normal(0, 1, N)
    ytrue = (np.cos(x) + 2) / (np.cos(x*1.4) +2)
    noise = np.random.normal(0, 0.2, N)
    y = ytrue + noise
    return x, y, ytrue
N = 1000
x, y, ytrue = genDataSet(N)
plt.plot(x,y,'.')
plt.plot(x,ytrue,'rx')
plt.savefig('HW4_PartA', dpi=200, bbox_inches='tight')
```

Code ends

In the code, I have introduced a variable N to change the number of data points to be generated which will further be used to perform regression.
Output



The chart above shows the distribution of the data points x and y across the plane.

1.2 b,c

Using 10-fold CV, you will report the three best values of k-neighbors that yield the best CV Eout. You will vary the values of k in the following range:

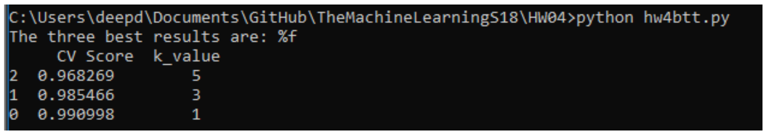
$$k = 2 * ((N + 1)/2) - 1$$

You will report the best CV Eout. Code Begins

```
import numpy as np
from matplotlib import pyplot as plt
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
import pandas as pd
def genDataSet(N):
    x = np.random.normal(0, 1, N)
    ytrue = (np.cos(x) + 2) / (np.cos(x*1.4) +2)
    noise = np.random.normal(0, 0.2, N)
    y = ytrue + noise
    return x, y, ytrue
N = 1000 #defining N
x, y, ytrue = genDataSet(N)

#converting x, y, ytrue to NP Arrays
x = np.array(x)
y = np.array(y)

#using KFold to get the splits
kf = KFold(n_splits=10)
kf.get_n_splits(x)
```

```
#dividing the data in train and test sets
for train_index,test_index in kf.split(x):
    X_train, X_test = x[train_index], x[test_index]
    y_train, y_test = y[train_index], y[test_index]

i=1
k_val = []
scores = []
cv_scores = []
for i in xrange(1, len(train_index),2):
    k_val.append(i)
    #performing KNN Regressions
    neigh = KNeighborsRegressor(n_neighbors=i)
    x = x.reshape((-1,1))
    y = y.reshape((-1,1))
    ytrue = ytrue.reshape((-1,1))
    neigh.fit(x, ytrue)
    #checking the R square (goodness of fit) for the knn
    score = neigh.score(x, ytrue)
    #checking cross validation score
    cv_score = cross_val_score(neigh, x, ytrue, cv=10)
    scores.append(score)
    cv_scores.append(np.mean(cv_score))

d = {'k_value': k_val,'CV Score': cv_scores}
results = pd.DataFrame(data=d)
print "The three best results are: %f \n ",results.loc[results["
    ↪ CV Score"].argsort()[-3:]]
```

Code ends.
Output



```
C:\Users\deepd\Documents\GitHub\TheMachineLearningS18\HW04>python hw4btt.py
The three best results are: %f
    CV Score  k_value
2  0.968269       5
1  0.985466       3
0  0.990998       1
```

The code above has 2 parts to have odd values of "k" for number of neighbors. In part 1, we divide the data in train and test so as to calculate the cross validation on test data in part 2.

The main reason for doing so is, while the other for loop varies the value of k as,

$$k = 2 * ((N + 1)/2) - 1$$

The Knn-regressor function gives error for indexes because it divides the data in test and train if not done explicitly and that doesn't match with the index that for loo tries to use.

At the end, we display the three best results of cross validation and its corresponding ”k” value which answers part C of the question as well.

2 Problem 2

Here is my attempt to iterate the code 100 times and save results. Somehow, the data-frame is getting over-written and so I am unable to store and generate histogram of the same. The code looks like,

```
import numpy as np
from matplotlib import pyplot as plt
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
import pandas as pd
def genDataSet(N):
    x = np.random.normal(0, 1, N)
    ytrue = (np.cos(x) + 2) / (np.cos(x*1.4) +2)
    noise = np.random.normal(0, 0.2, N)
    y = ytrue + noise
    return x, y, ytrue
N = 1000
x, y, ytrue = genDataSet(N)
x = np.array(x)
y = np.array(y)
kf = KFold(n_splits=10)
kf.get_n_splits(x)

for train_index,test_index in kf.split(x):
    #print("Train: ",train_index,"Test: ",test_index)
    X_train, X_test = x[train_index], x[test_index]
    y_train, y_test = y[train_index], y[test_index]

i=1
k_val = []
scores = []
cv_scores = []
#print("Train Index: ",len(train_index))
#print("Test Index: ",len(test_index))
k_each = []
k_all = []

for j in xrange(1,100,1):
    for i in xrange(1, len(train_index),2):
        #print("K IS: ",i)
        k_val.append(i)
        neigh = KNeighborsRegressor(n_neighbors=i)
        x = x.reshape((-1,1))
        y = y.reshape((-1,1))
        ytrue = ytrue.reshape((-1,1))
        neigh.fit(x, ytrue)
        score = neigh.score(x, ytrue)
        cv_score = cross_val_score(neigh, x, ytrue, cv=10)
        scores.append(score)
        cv_scores.append(np.mean(cv_score))
        #print cv_score

#print ("R^2 scores: %s" %scores)
#print ("CV scores: %s" %cv_scores)
    d = {'k_value': k_val,'CV Score': cv_scores}
    results = pd.DataFrame(data=d)
    max_e = max(results["CV Score"])
    #print "The best results are: %f \n ",results.loc[(results["CV
        ↪  Score"]== max_e)]

    print "The three best results are: %f \n ",results.loc[results
        ↪ ["CV Score"].argsort()[-3:]]
    print "J is ", j
    #print results.loc[results["CV Score"].argsort()[-3:],'k_value
        ↪ ']
    k_each = results.loc[results["CV Score"].argsort()[-3:],'
        ↪ k_value'].iloc[0]
    k_each = k_each.append(results.loc[results["CV Score"].argsort
        ↪ ()[-2:],'k_value'].iloc[0])
        #k_each = results.loc[results["CV Score"].argsort()[-3:],'
            ↪ k_value'].iloc[0]
    k_all.append(k_each)
    results = results.reset_index(drop=True, inplace=True)
    #del results['index']
    #results.drop(results.index, inplace=True)

print k_all
```