ps1_problem1_eduardo_beltrame

April 21, 2020

0.1 IDS/ACM/CS 158: Fundamentals of Statistical Learning

0.1.1 PS1, Problem 1: K-NN and Linear Regression for Regression.

Name: Beltrame, Eduardo

Email address: edaveiga@caltech.edu

0.1.2 Problem 1 part a) Write a function `knn_regression`, which implements the k-NN method

[199]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.spatial.distance import pdist, squareform
from numpy.linalg import inv
from numpy import matmul
```

[167]:
```python
def knn_regression(K, D, X):
    df = pd.DataFrame(D)
    #get number of dimensions in our training data
    dims = np.shape(df)[1] -1

    # check that the dimensions of D and X match
    assert dims==np.shape(X)[0]
    #concatenate the new point with the training data
    concat = df.iloc[:,0:dims].append(pd.Series([1,2,dims], index=df.iloc[:,0:
 ↪dims].columns),ignore_index=True)

    # calculate distance matrix
    distance_matrix=squareform(pdist(df))


    # put the indices of the K lowest entries in the first K entries of␣
 ↪`partition`
    partition = np.argpartition(distance_matrix[-1],K)
    # calculate predicted Y by doing the mean of the Y's of the fetched indices
    kmean = df.loc[partition[:K]]['Y']
    Y = kmean
```

```
    return kmean
```

[ ]:

0.1.3 Problem 1 part b) Write a function `linreg_regression`, which implements the linear regression method

I used this tutorial to make the OLS using classes

[273]:
```
class OLS:
    def __init__(self):
        self.coefficients=[]
    def fit(self, X, y):
        ones = np.ones(shape=X.shape[0]).reshape(-1,1)
        X = np.concatenate((ones, X), 1)
        self.coefficients = matmul(matmul(inv(matmul(X.T, X)), X.T), y)

    def predict(self, entry):
        b0 = self.coefficients[0]
        other_betas = self.coefficients[1:]
        prediction = b0

        for xi, bi in zip(entry, other_betas): prediction = prediction + (bi*xi)
        return prediction

def linreg_regression(D, X):
    df = pd.DataFrame(D)
    dims = np.shape(df)[1] -1
    # check that the dimensions of D and X match
    assert dims==np.shape(X)[0]

    model.fit(df.iloc[:,0:dims].values,df.iloc[:,dims:dims+1].values)
    return model.predict(X)
```

0.1.4 Problem 1 part c) Write a script `ps1problem1` that compares the two methods on two different data sets, dataset1 and dataset2

[296]:
```
# compute linreg error on dataset 1
D = train1 = pd.read_csv('./data/dataset1_train.csv')
test1 = pd.read_csv('./data/dataset1_test.csv')

linreg_square_errors1 = []
for idx, row in test1.iterrows():
    X = row[0:3]
    prediction = linreg_regression(D=D, X=X)
```

```
    truth = row[3]
    error = prediction - truth
    linreg_square_errors1.append(error*error)

print('The mean squared error of linreg on test1 dataset is:')
linreg_mse1=np.mean(linreg_square_errors1)

print(linreg_mse1)

# compute linreg error on dataset 2
D = train2 = pd.read_csv('./data/dataset2_train.csv')
test2 = pd.read_csv('./data/dataset2_test.csv')

linreg_square_errors2 = []
for idx, row in test2.iterrows():
    X = row[0:3]
    prediction = linreg_regression(D=D, X=X)
    truth = row[3]
    error = prediction - truth
    linreg_square_errors2.append(error*error)
linreg_mse2=np.mean(linreg_square_errors2)
print('The mean squared error of linreg on test2 dataset is:')
print(linreg_mse2)
```

```
The mean squared error of linreg on test1 dataset is:
0.04266437566774965
The mean squared error of linreg on test2 dataset is:
2.3068086726109702
```

[297]:

```
# compute linreg error on dataset 1
D = train1 = pd.read_csv('./data/dataset1_train.csv')
test1 = pd.read_csv('./data/dataset1_test.csv')

knn_square_errors1 = []
for idx, row in test1.iterrows():
    X = row[0:3]
    prediction = knn_regression(K=5,D=D, X=X)
    truth = row[3]
    error = prediction - truth
    knn_square_errors1.append(error*error)

print('The mean squared error of knn on test1 dataset is:')
knn_mse1=np.mean(knn_square_errors1)
print(knn_mse1)

# compute linreg error on dataset 2
D = train2 = pd.read_csv('./data/dataset2_train.csv')
```

```
test2 = pd.read_csv('./data/dataset2_test.csv')

knn_square_errors2 = []
for idx, row in test2.iterrows():
    X = row[0:3]
    prediction = knn_regression(K=5,D=D, X=X)
    truth = row[3]
    error = prediction - truth
    knn_square_errors2.append(error*error)

knn_mse2=np.mean(knn_square_errors2)
print('The mean squared error of linreg on test2 dataset is:')
print(knn_mse2)
```

```
The mean squared error of knn on test1 dataset is:
14.5486140645868
The mean squared error of linreg on test2 dataset is:
6.4935661873427275
```

[298]:
```
print('The R for dataset 1 is:')
print(knn_mse1/linreg_mse1)

print('The R for dataset 2 is:')
print(knn_mse2/linreg_mse2)
```

```
The R for dataset 1 is:
341.0014523096424
The R for dataset 2 is:
2.814956552071984
```

Apparently regression outperforms knn by a lot…

[ ]:
```

```