ps1_problem1

April 20, 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: Li, Michael

Email address: mlli@caltech.edu

Notes: Please use python 3.6

You are required to properly comment and organize your code.

- Helper functions (add/remove part label according to the specific question requirements)

[1]:
```
import numpy as np
import numpy.matlib

def load_data(filename):
    """
    filename - filename to open and load

    Returns file as matrix where last column is
    y_i and columns up to last one is x_i
    """
    res = np.loadtxt(open(filename, "rb"), delimiter=",", skiprows=1)
    return res

def average_error(ys, y_preds):
    """
    ys - vector of real outputs
    y_preds - vector of predicted outputs

    Returns L2 loss between vectors
    """
    return np.mean((ys - y_preds)**2)
```

- Part A

[2]:
```
def knn_regression(K, D, X):
    """
```

```
    K - number of neighbors
    D - training data consisting of pairs of p-dimensional vectors and outputs
    X - a column p-vector that represents a new input

    Returns the K-NN regression of X using D
    """
    train_x = D[:,:-1]
    train_y = D[:,-1]

    # find distances to X and sort points in D by that
    dists = np.sqrt(np.sum((train_x - np.matlib.repmat(X, len(train_x), 1))**2,␣
↪axis=1))
    inds = dists.argsort()

    # return the mean of the outputs of the first K observations
    return np.mean(train_y[inds][:K])
```

- Part B

[3]:
```
def linreg_regression(D, X):
    """
    D - training data consisting of pairs of p-dimensional vectors and output
    X - a column p-vector that represents a new input

    Returns the linear regression of X using D
    """

    x = D[:,:-1]
    y = D[:,-1]

    # add bias term to training data
    bias = np.matlib.repmat(1, len(x), 1)
    x = np.concatenate((bias, x), axis=1)

    # calculate beta
    intermediate = np.matmul(x.transpose(), x)
    inverse_intermediate = np.linalg.inv(np.array(intermediate))
    pseudo_x = np.matmul(inverse_intermediate, x.transpose())

    beta = np.matmul(pseudo_x, y)

    # apply beta weight to X
    return np.matmul(np.insert(X, 0, 1), beta)
```

- Part C

[4]:
```
def knn_vs_linear_reg(train_filename, test_filename, dataset):
    """
```

```python
    train_filename - filename of training data to load
    test_filename - filename of test data to load
    dataset - number of dataset

    Prints Results for KNN vs LinReg
    """
    K = 5
    training_data = load_data(train_filename)
    test_data = load_data(test_filename)

    test_x = test_data[:,:-1]
    test_y = test_data[:,-1]

    # run KNN and Linear Regression on all points in test dataset
    knn = [knn_regression(K, training_data, test_x[i]) for i in␣
 ↪range(len(test_x))]
    lr = [linreg_regression(training_data, test_x[i]) for i in␣
 ↪range(len(test_x))]

    # compute the L2 loss of both models
    Err_knn = average_error(test_y, np.array(knn))
    Err_lr = average_error(test_y, np.array(lr))
    R = Err_knn / Err_lr

    print('For dataset {}\n Err_knn is {:1.4f} \n Err_lr is {:1.4f} \n R = {:1.
 ↪4f}'.format(dataset, Err_knn, Err_lr, R))
    if R > 1:
        print(' Linear regression is better.')
    else:
        print(' k-NN is better.')
```

[5]:

```python
knn_vs_linear_reg('dataset1_train.csv', 'dataset1_test.csv', 1)
print()
knn_vs_linear_reg('dataset2_train.csv', 'dataset2_test.csv', 2)
```

```
For dataset 1
 Err_knn is 0.3416
 Err_lr is 0.0427
 R = 8.0073
 Linear regression is better.

For dataset 2
 Err_knn is 0.4928
 Err_lr is 2.3068
 R = 0.2136
 k-NN is better.
```