HW1 - Linear regression & k-NN

September 18, 2018

```
In [ ]: import numpy as np
        import matplotlib.pyplot as pl
        from numpy import linalg as LA
        from operator import itemgetter
```

```
In [ ]: # Import the data from the text file
        train_data_2 = np.loadtxt("/Users/qinghongxu/Documents/MATH895/HW1/train.2.txt",
                                  delimiter=',')
        train_data_3 = np.loadtxt("/Users/qinghongxu/Documents/MATH895/HW1/train.3.txt",
                                  delimiter=',')
        test_data = np.loadtxt("/Users/qinghongxu/Documents/MATH895/HW1/test.txt",
                                  delimiter=' ')
```

```
In [ ]: # Edit training data
        label_2 = np.zeros((train_data_2.shape[0]))
        label_3 = np.ones((train_data_3.shape[0]))
        label_23 = np.append(label_2, label_3, axis=0)
        train_data_23 = np.append(train_data_2, train_data_3, axis=0)
```

```
In [ ]: # Edit test data
        test_data_2 = test_data[test_data[:,0] == 2,1:257]
        test_data_3 = test_data[test_data[:,0] == 3,1:257]
        test_label_2 = np.zeros((test_data_2.shape[0]))
        test_label_3 = np.ones((test_data_3.shape[0]))
        test_label_23 = np.append(test_label_2, test_label_3, axis=0)
        test_data_23 = np.append(test_data_2, test_data_3, axis=0)
        #test_label_23 = np.concatenate((test_label_2,test_label_3))
        #test_data_23 = np.concatenate((test_data_2,test_data_3))
```

```
In [ ]: def computeDistance(instance1, instance2):
            distance = LA.norm(instance1 - instance2, 2)
            return distance
```

```
In [ ]: def computeDistanceSet(traindataSet, testdataInstance):
            distances = []
            for x in range(len(traindataSet)):
                dist = computeDistance(testdataInstance, traindataSet[x])
                distances = np.append(distances, dist)
            traindataNew = np.insert(traindataSet, 0, distances, axis = 1 )
            return traindataNew
```

```
In [ ]: def findNeighbours(traindataNew, label, k):
            traindataNew = np.insert(traindataNew, 1, label, axis = 1)
            traindataSort = traindataNew[np.argsort(traindataNew[:, 0])]
            neighbours = traindataSort[0:k]
            return neighbours
```

```
In [ ]: def getResponse(neighbours):
            k = neighbours.shape[0]
            response = float(np.sum(neighbours, axis=0)[1])/k
            if response >= 0.5:
                response = 1
            else:
                response = 0
            return response
```

```
In [ ]: # k - Nearest neighbours
        def getAccuracykNN(traindata, testdata, k, trainlabel, testlabel):
            prediction = []
            for x in range(testdata.shape[0]):
                traindataNew = computeDistanceSet(traindata, testdata[x])
                neighbours = findNeighbours(traindataNew, trainlabel, k)
                result = getResponse(neighbours)
                prediction = np.append(prediction, result)
            incorrect_index = np.where(testlabel!=prediction)[0]
            error = float(incorrect_index.size)/(testdata.shape[0])
            return(error)
```

```
In [ ]: # Linear regression fit
        def getAccuracyLR(traindata, testdata, trainlabel, testlabel):
            prediction = []
            traindata = np.insert(traindata, 0, 1, axis=1)
            testdata = np.insert(testdata, 0, 1, axis=1)
            beta_hat = np.linalg.lstsq(traindata, trainlabel, rcond=None)[0]
            prediction = np.matmul(testdata, beta_hat)
            prediction[np.where(prediction < 0.5)[0]] = 0
            prediction[np.where(prediction >= 0.5)[0]] = 1
            incorrect_index = np.where(testlabel!=prediction)[0]
            error = float(incorrect_index.size)/(testdata.shape[0])
            return(error)
```

```
In [ ]: # Perform k-nn on training data
        k = [1, 3, 5, 7, 15]
        for x in k:
            error = getAccuracykNN(train_data_23, train_data_23, x, label_23, label_23)
            print 'when k = %d , the error for training data is %f' % (x, 100*error)
```

```
In [ ]: # Perform k-nn on test data
        k = [1, 3, 5, 7, 15]
        for x in k:
            error = getAccuracykNN(train_data_23, test_data_23, x, label_23, test_label_23)
            print 'when k = %d , the error for test data is %f' % (x, 100*error)
```

```
In [ ]: # Perform linear regression on training/test data
        data = [train_data_23, test_data_23]
        dataname = ['test data', 'training data']
        label = [label_23, test_label_23]
        for i in [0, 1]:
            error = getAccuracyLR(data[0], data[i], label[0], label[i])
            print 'the error for %s is %f' % (dataname[i], error)
```