q2

April 3, 2020

1 Q2 - Logistic Regression After applying PCA

[15]:

```
import os
import sys
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from skimage import transform,io
from sklearn.metrics import accuracy_score

url_train_csv = "/home/abhishek/dev/Semester_2/SMAI/Assignments/Assignment_3/q2/
 ↪sample_train_2.txt"
url_test_csv = "/home/abhishek/dev/Semester_2/SMAI/Assignments/Assignment_3/q2/
 ↪sample_test_2.txt"
sample_output_csv = "/home/abhishek/dev/Semester_2/SMAI/Assignments/
 ↪Assignment_3/q2/output_of_sample_test_2.txt"

class LogisticRegression:
    def read_csv_file(self,marker):
        image_files = []
        f = None
        if(marker == "train"):
            f = open(url_train_csv,'r')
        elif(marker == "test"):
            f = open(url_test_csv,'r')
        else:
            f = open(sample_output_csv,'r')

        lines = f.readlines()
        for line in lines:
            image_files.append(line.strip())
        return image_files

    def split_image_dir_and_label(self,images_directory,marker):
        if(marker == "train"):
            # print("splitting directory and labels")
            label = []
            directory = []
            for l in images_directory:
                x = l.split(" ")
                label.append(x[1])
                directory.append(x[0])
            # print("splitting directory and labels done ....")
            return label, directory
        elif(marker == "test"):
            # print("read lines...")
            lines = []
            for l in images_directory:
                lines.append(l)
            # print("reading lines done...")
            return lines

    def read_image_from_directory_to_grayscale(self,directory):
        # print("reading images from directory and downscaling images")
        faces = []
        for i in directory:
            img = io.imread(i)
            img = img.astype(np.uint8)
            # converting to grayscale
            rgb_weights = [0.2989, 0.5870, 0.1141]
            grayscale_image = np.dot(img[...,:3], rgb_weights)
            small_grey = transform.resize(grayscale_image, (64,64),␣
↪mode='symmetric', preserve_range=True)
            reshape_img = small_grey.reshape(1, 4096)
            faces.append(reshape_img[0])
        X = np.asarray(faces)
        # print("reading images from directory and downscaling images done...")
        return X

    def apply_pca(self,X):
        # print("Applying PCA on the image set")
        eig_val, eig_mat = np.linalg.eig(np.cov(X))
        idx = eig_val.argsort()[::-1]
        eig_val = eig_val[idx]
        eig_mat = eig_mat[:,idx]
        eigen_coeff = eig_mat[:,range(50)]
        X_PCA = np.dot(eigen_coeff.T,X)
        # print("Applying PCA on the image set done .......")
        return X_PCA.T

    def sigmoid(self,z):
        return 1/(1+np.exp(-z))

    def cost(self,w,b,X,y,lmd=10):
        # print("Calculating Cost")
        m = X.shape[0]
        z = np.matmul(X,w) + b
        hx = self.sigmoid(z)
        J = (-1/m)*( np.sum( y * np.log(hx) + (1. - y) * np.log(1.- hx) ) )
        J += (lmd/(2*m))* np.matmul(w,w)
        # print("Calculating Cost done ..")
        return J

    def gradient_descent(self,w,b,X,y,learning_rate=0.01,lmd=10,no_of_iteration=10000):
        # print("Applying Gradient Descent")
        m = X.shape[0]
        # print("Initial cost: {}".format( self.cost(w,b,X,y) ))
        for i in range(no_of_iteration):
            z = np.matmul(X,w) + b
            hx = self.sigmoid(z)
            dw = (1/m)*np.matmul(X.T,hx-y)
            db = (1/m)*np.sum(hx-y)
            factor = 1-( (learning_rate * lmd)/m)
            w = w*factor - learning_rate*dw
            b = b - learning_rate*db
            # if i % 500 == 0:
                # print("Iteration {} cost :{}".format(i,self.cost(w,b,X,y)))
        # print("Final cost: {}".format( self.cost(w,b,X,y) ))
        # print("Applying Gradient Descent done ....")
        return w,b

    def accuracy(self,w,b,X,y):
        # print("Calculating Accuracy")
        m = X.shape[0]
        z = np.matmul(X,w) + b
        hx = self.sigmoid(z)
        pred = np.round(hx)
        correct_pred = (pred==y)
        total = np.sum(correct_pred)
        # print("Calculating Accurcy done .....")
        return (total*100)/m

    def test(self,w,b,X):
        # m = X.shape[0]
        z = np.matmul(X,w)+b
        hx = self.sigmoid(z)
        pred = np.round(hx)
        return pred

lr = LogisticRegression()
```

[ ]:

```
## Applying PCA on the above dataset
```

[2]:

```
lines = lr.read_csv_file("train")
# print(images_directory)
label, directory = lr.split_image_dir_and_label(lines,"train")
# label = np.asarray(label)
# print(label)
unique_labels = np.unique(np.asarray(label))
# print(unique_labels)
X_train = lr.read_image_from_directory_to_grayscale(directory)
# normalizing
X_train = X_train/255
#applying PCA
X_PCA = lr.apply_pca(X_train.T)
# print(X_PCA.shape)
# print(X_PCA[0])
m = X_train.shape[0]
```

[3]:

```
print(X_PCA.shape)
```

```
(520, 50)
```

[7]:

```
## Training Multi-Class Classifier for each unique class in Tranining set
```

[4]:

```
weight_label_map = {}
i = 0
THETA = []
BIAS = []

for unique in unique_labels:
    y_train = []
    for l in label:
        if(unique == l):
            y_train.append(1)
        else:
            y_train.append(0)

    weight_label_map[i] = unique
    i = i+1
    y_train = np.asarray(y_train)
    # print(y_train)
    w = np.zeros(X_PCA.shape[1],dtype=np.float64)
    b = 0.0
    w,b = lr.gradient_descent(w,b,X_PCA,y_train)
    THETA.append(w)
    BIAS.append(b)
```

[5]:

```
## ALL Unique Labels in Training set
```

[6]:

```
print(weight_label_map)
```

```
{0: '000', 1: '001', 2: '002', 3: '003', 4: '004', 5: '005', 6: '006', 7: '007'}
```

[12]:

```
## Applying PCA on Test Data set
```

[7]:

```
THETA = np.asarray(THETA)
BIAS = np.asarray(BIAS)

lr1 = LogisticRegression()
lines = lr1.read_csv_file("test")
directory = lr1.split_image_dir_and_label(lines,"test")
X_test = lr1.read_image_from_directory_to_grayscale(directory)
X_test = X_test/255
X_Test_PCA = lr1.apply_pca(X_test.T)
# print(X_Test_PCA.shape)
```

[8]:

```
## Predicting The labels of Test Data by taking maximum sigmoid value of label␣
 ↪along all classifier
```

[9]:

```
y_pred = []
for X in X_Test_PCA:
    i = 0
    max_hx = 0.0
    index = 0
    for w,b in zip(THETA,BIAS):
        pred = lr1.test(w,b,X)
        if(pred > max_hx):
            max_hx = pred
            index = i
        i = i+1
    y_pred.append(weight_label_map[index])

y_pred = np.asarray(y_pred)
for prediction in y_pred:
    print(prediction)
```

```
000
000
000
002
001
002
003
005
```

```
000
000
000
002
000
002
000
000
007
000
000
002
007
000
000
000
000
000
006
006
001
006
000
000
000
003
006
002
007
000
002
000
000
000
004
000
000
003
003
001
000
006
006
001
005
000
000
001
```

```
002
000
000
003
002
001
000
000
000
004
000
000
004
007
006
006
007
007
000
000
000
006
000
006
000
006
004
007
000
007
007
002
000
003
005
002
000
003
000
005
000
003
001
007
000
000
005
000
```

```
007
000
000
000
000
000
000
000
000
007
002
003
007
000
000
000
003
007
001
000
000
000
007
004
000
000
002
000
002
006
000
005
000
000
001
000
004
000
005
003
003
000
000
006
007
001
000
000
```

```
005
000
000
000
004
000
007
005
000
000
000
002
000
000
004
003
001
000
005
000
004
000
000
000
006
000
000
003
000
000
000
000
000
005
005
000
005
007
000
000
000
002
006
007
000
007
004
003
```

```
005
002
000
003
000
000
004
000
003
002
004
004
000
000
006
007
000
004
000
001
003
000
000
000
007
003
002
000
003
000
000
000
000
000
005
002
000
006
000
000
005
000
004
000
000
002
005
002
```

```
000
000
000
000
000
000
000
002
000
000
005
004
005
000
000
001
007
007
001
000
000
003
003
000
000
000
004
003
007
003
000
002
007
000
007
005
000
000
000
006
004
000
000
007
000
000
007
000
```

```
003
000
001
005
005
000
000
005
004
002
000
000
000
004
006
004
000
005
004
007
002
000
007
000
005
002
000
007
005
000
007
002
007
004
003
000
000
000
003
002
000
000
000
003
000
001
003
000
```

```
000
001
000
000
003
000
000
000
007
000
004
007
000
000
004
000
002
001
000
000
000
000
000
000
007
000
000
000
004
000
000
007
000
000
002
000
005
000
000
000
000
000
003
005
000
004
000
002
```

```
000
000
000
000
005
004
007
000
002
000
000
004
003
005
007
004
004
001
006
004
004
007
005
007
000
000
003
007
000
004
000
004
003
004
007
002
005
003
006
002
000
000
002
003
005
000
004
000
```

```
000
000
003
005
007
007
000
000
000
000
000
003
000
002
002
006
000
003
003
003
001
000
000
000
006
000
003
007
001
000
000
000
004
000
002
000
005
000
000
000
006
002
004
000
000
001
007
005
```

```
007
000
004
003
006
000
000
000
004
005
001
002
002
007
003
000
004
007
000
000
000
003
004
004
002
000
000
000
002
001
007
003
```

```
[16]: labels = lr.read_csv_file("label")
```

```
[18]: print(len(labels))
```

```
520
```

```
[20]: labels = np.asarray(labels)
```

```
[21]: print(labels.shape)
```

```
(520,)
```

```
[ ]: ## Accuracy Score for predicting the labels
```

```
[22]: accuracy_score(labels,y_pred)
```

```
[22]: 0.5846153846153846
```

```
[ ]:
```