SMAI ASSIGNMENT 2

2019115007

Question 1 : Eigen Values and Eigen Vectors

A) Singular value decomposition is more generalizable because, it can be applied for both rectangle and square matrices. But Eigen value decomposition is applied for only Square matrices.

B)

Given Matrix $M = \begin{bmatrix} 4 & 8 \\ 11 & 7 \\ 14 & -2 \end{bmatrix}$, we have to decompose it using singular value decomposition. The splitting will be as follows

$$M = U\Sigma V^T \tag{1}$$

Where

Now let us solve for every one of them.

<u>For V and Σ</u>

- First we have to calculate eigen vectors and values of M^TM

$$M^TM = \begin{bmatrix} 4 & 11 & 14 \\ 8 & 7 & -2 \end{bmatrix} * \begin{bmatrix} 4 & 8 \\ 11 & 7 \\ 14 & -2 \end{bmatrix} = \begin{bmatrix} 333 & 81 \\ 81 & 117 \end{bmatrix}$$

- M^TM is always square, hence we can find the eigenvalues λ_1, λ_2. After solving for eigen vectors and values, we get

  eigen values as

$$\lambda_1 = 360,$$
$$\lambda_2 = 90$$

  corresponding square roots of non-zero eigen values are

$$\sigma_1 = 6\sqrt{10},$$
$$\sigma_2 = 3\sqrt{10}$$

corresponding eigen vectors are

$$w1 = \begin{bmatrix} 3 \\ 1 \end{bmatrix}, w2 = \begin{bmatrix} \frac{-1}{3} \\ 1 \end{bmatrix}$$

After normalizing the eigen vectors we get

$$v1 = \begin{bmatrix} \frac{3}{\sqrt{10}} \\ \frac{1}{\sqrt{10}} \end{bmatrix}, v2 = \begin{bmatrix} \frac{-1}{\sqrt{10}} \\ \frac{3}{\sqrt{10}} \end{bmatrix}$$

- with this information we can say our matrices,

$$V = \begin{bmatrix} \frac{3}{\sqrt{10}} & \frac{-1}{\sqrt{10}} \\ \frac{1}{\sqrt{10}} & \frac{3}{\sqrt{10}} \end{bmatrix}, \qquad \Sigma = \begin{bmatrix} 6\sqrt{10} & 0 \\ 0 & 3\sqrt{10} \\ 0 & 0 \end{bmatrix}$$

Calculation of U

- Now let us calculate u_i , using $u_i = \frac{1}{\sigma_i} M v_i$

$$u_1 = \frac{1}{6\sqrt{10}} \begin{bmatrix} 4 & 8 \\ 11 & 7 \\ 14 & -2 \end{bmatrix} \begin{bmatrix} \frac{3}{\sqrt{10}} \\ \frac{1}{\sqrt{10}} \end{bmatrix} = \begin{bmatrix} \frac{1}{3} \\ \frac{2}{3} \\ \frac{2}{3} \end{bmatrix}$$

$$u_2 = \frac{1}{3\sqrt{10}} \begin{bmatrix} 4 & 8 \\ 11 & 7 \\ 14 & -2 \end{bmatrix} \begin{bmatrix} \frac{-1}{\sqrt{10}} \\ \frac{3}{\sqrt{10}} \end{bmatrix} = \begin{bmatrix} \frac{2}{3} \\ \frac{1}{3} \\ \frac{-2}{3} \end{bmatrix}$$

- We will use Gram-Schmidt process for calculating third vector, let us start with a vector which is not in the plane of u_1, u_2. Let it be y. Now we can calculate u_3 by using

$$u_3 = y - \frac{y.u_1}{u_1.u_1} u_1 - \frac{y.u_2}{u_2.u_2} u_2 \tag{2}$$

For making our lives simple, consider

$$y = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

after solving u_3 using equation 2, we get

$$u_3 = \begin{bmatrix} \frac{4}{9} \\ \frac{-4}{9} \\ \frac{2}{9} \end{bmatrix}$$

Normalizing the obtained vector u_3, we get

$$u3 = \begin{bmatrix} \frac{2}{3} \\ \frac{-2}{3} \\ \frac{1}{3} \end{bmatrix}$$

Now we know that,

$$U = [u_1 \quad u_2 \quad u_3]$$

$$\implies U = \begin{bmatrix} \frac{1}{3} & \frac{2}{3} & \frac{2}{3} \\ \frac{2}{3} & \frac{1}{3} & \frac{-2}{3} \\ \frac{2}{3} & \frac{-2}{3} & \frac{1}{3} \end{bmatrix}$$

- We got all the required values, To sum up ,

$$M = U\Sigma V^T$$

Where

$$U = \begin{bmatrix} \frac{1}{3} & \frac{2}{3} & \frac{2}{3} \\ \frac{2}{3} & \frac{1}{3} & \frac{-2}{3} \\ \frac{2}{3} & \frac{-2}{3} & \frac{1}{3} \end{bmatrix}, \Sigma = \begin{bmatrix} 6\sqrt{10} & 0 \\ 0 & 3\sqrt{10} \\ 0 & 0 \end{bmatrix}, V = \begin{bmatrix} \frac{3}{\sqrt{10}} & \frac{-1}{\sqrt{10}} \\ \frac{1}{\sqrt{10}} & \frac{3}{\sqrt{10}} \end{bmatrix},$$

Question 2 : LDA and PCA

A)

Statement	**True / False**
(a) PCA can be useful if all elements of D are equal	False
(b) PCA can be useful if all elements of D are not equal	True
(c) D is not full-rank if all points in X lie on a straight line	True
(d) V is not full-rank if all points in X lie on a straight line	False
(e) D is not full-rank if all points in X lie on a circle	False

B)

FALSE

Question 3 : Bayes Theorem

A)

Prior Probability : It is the probability that an observation will fall into a class or group before we have any knowledge or data on the observation.

Posterior Probability: It is the probability that an observation will fall into a group after having some data on the observation.

B)

Let us represent flu with F, symptoms with S

- Given An individual has flu he will have sore throat and headache 90$\%$ of the times.

$$P(\frac{S}{F}) = 0.9$$

- Only 5$\%$ of the population have the flu at any given time of the year.

$$P(F) = 0.05$$

- 20% of the population have headache and sore throat at any given time.

$$P(S) = 0.2$$

- We are asked to the probability that I have flu given that I have symptoms. That is $P(\frac{F}{S})$

- Using Bayes Probability Theorem:

$$P(\frac{F}{S}) = \frac{P[\frac{S}{F}] * P[F]}{P[S]}$$

$$\implies P[\frac{F}{S}] = \frac{0.045}{0.2} = 0.225$$

Question 5 : K Nearest neighbors

```python
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
```

```python
#Load data
iris = pd.read_csv('Iris.csv')
#data cleaning
iris.drop(columns="Id",inplace=True)
iris
```

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 5 columns

```python
#features and labels
X=iris.iloc[:,0:4].values
y=iris.iloc[:,4].values

#Train and Test split
x_train,x_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=0)

y_pred = np.zeros([len(y_test),])
```

```python
#Giving lables for flowers in dataset test dataset
for i in range(0,len(y_test)):
    if(y_test[i]=='Iris-setosa'):
        y_test[i]=1
    elif (y_test[i]=='Iris-virginica'):
        y_test[i]=2
    elif (y_test[i]=='Iris-versicolor'):
        y_test[i]=3
    else:
        print("You are a fool")


for i in range(0,len(y_train)):
    if(y_train[i]=='Iris-setosa'):
        y_train[i]=1
    elif (y_train[i]=='Iris-virginica'):
        y_train[i]=2
    elif (y_train[i]=='Iris-versicolor'):
        y_train[i]=3
    else:
        print("You are a fool")

y_train= y_train.astype('int16')
y_test= y_test.astype('int16')
y_pred= y_pred.astype('int16')
```

```python
# KNN K
KNN=5
for i in range(0,len(x_test)):
    nbrs=np.zeros([len(x_train),2])
    for j in range(0,len(x_train)):
        distance=0
        for k in range(0,len(x_train[0])):
            distance += (x_train[j][k]-x_test[i][k])**2
        nbrs[j]=[distance,y_train[j]]

    nbrs = nbrs[nbrs[:,0].argsort()]
    nbrs = nbrs[:KNN]
    c1=c2=c3=0;
    for e in range(0,len(nbrs)):
        if(nbrs[e][1]==1):
            c1+=1
        elif(nbrs[e][1]==2):
            c2+=1
        elif(nbrs[e][1]==3):
            c3+=1
        else:
            print("You are fooled")
    if(c1==max(c1,c2,c3)):
        y_pred[i]=1
    elif(c2==max(c1,c2,c3)):
        y_pred[i]=2
    elif(c3==max(c1,c2,c3)):
        y_pred[i]=3
    else:
        print("Fooled UP")
```

```python
print(y_pred)
print(y_test)
```

```
[2 3 1 2 1 2 1 3 3 3 2 3 3 3 2 1 3 3 1 1 2 3 1 1 2 1 1 3 3 1]
[2 3 1 2 1 2 1 3 3 3 2 3 3 3 3 1 3 3 1 1 2 3 1 1 2 1 1 3 3 1]
```

```python
## Accuracy Priting
import sklearn.metrics as metrics
print(np.sqrt(metrics.accuracy_score(y_test, y_pred)))

```

```
0.983192080250175
```

Accuracy

- AS you can see above accuracy score is 0.98

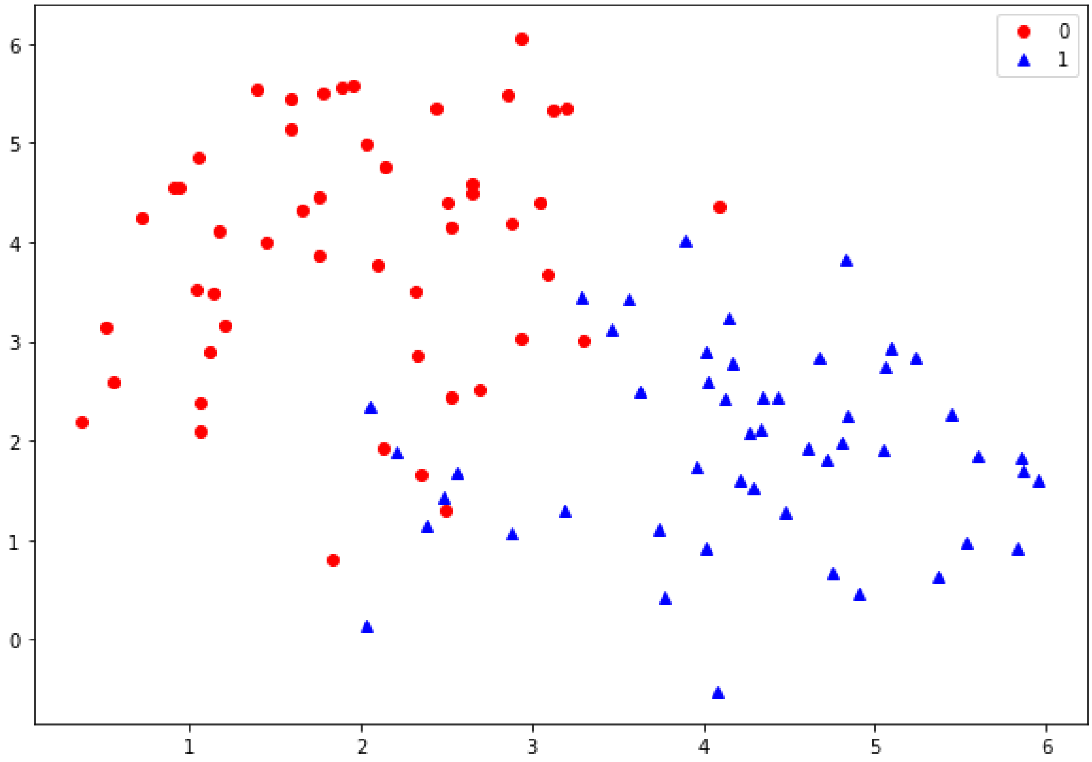
Question 5 : Logistic Regression Using Gradient Descent

Dataset

```
import numpy as np
import matplotlib.pyplot as plt
```

```
from sklearn.datasets import make_blobs
X, y =  make_blobs(n_samples=100, centers=[[2,4],[4,2]], random_state=20)
```

```
plt.figure(figsize=(10, 7))
#Visualize dataset
plt.plot(X[:,0][y==0],X[:,1][y==0],'o',color='r', label='0')
plt.plot(X[:,0][y==1],X[:,1][y==1],'^',color='b', label='1')
plt.legend();
```



```python
class Logreg:
    def __init__(self, num_iter,lr):
        self.num_iter = num_iter
        self.lr = lr

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


    def __loss(self, h, y):
        return (-y * np.log(h) - (1 - y) * np.log(1 - h)).mean()

    def fit(self, X, y):
        X= np.concatenate((np.ones((X.shape[0], 1)), X), axis=1)
        self.W = np.zeros(X.shape[1])

        for i in range(self.num_iter):

            h = self.__sigmoid(np.dot(X, self.W))
            self.W -= self.lr * (np.dot(X.T, (h - y)) / y.size)

            #update
            h = self.__sigmoid(np.dot(X, self.W))
            loss = self.__loss(h, y)

    def predict_prob(self, X):
        X= np.concatenate((np.ones((X.shape[0], 1)), X), axis=1)
        return self.__sigmoid(np.dot(X, self.W))

    def predict(self, X):
        return self.predict_prob(X).round()
```

```python
model = Logreg(num_iter=400000,lr=0.2)
model.fit(X, y)
```

```python
plt.figure(figsize=(10, 7))
plt.plot(X[:,0][y==0],X[:,1][y==0],'o',color='r', label='0')
plt.plot(X[:,0][y==1],X[:,1][y==1],'^',color='b', label='1')
plt.legend()

#inspired from github
x1_min, x1_max = X[:,0].min(), X[:,0].max(),
x2_min, x2_max = X[:,1].min(), X[:,1].max(),
xx1, xx2 = np.meshgrid(np.linspace(x1_min, x1_max), np.linspace(x2_min, x2_max))
grid = np.c_[xx1.ravel(), xx2.ravel()]
probs = model.predict_prob(grid).reshape(xx1.shape)
plt.contour(xx1, xx2, probs, [0.55], linewidths=1, colors='green')
```