HW 3

Juan Vila

October 28, 2019

Part 1

A.

As $r_i = y_i - Xw$, also we know that for any $\hat{w}$ we can form the following equation $y = X\hat{w} + \hat{r}$, then we replace this equation into the first one and we get $r = X\hat{w} + \hat{r} - Xw$, now if we reoder the terms, we get; $r_i = \hat{r} - X(\hat{w} - w)$

B.

Now if we construct the norm -2 function we get:

$$\|r\|^2 = (\hat{r} - X(\hat{w} - w))^T(\hat{r} - X(\hat{w} - w))$$

Now if we multiply both expresion we get:

$$\|r\|^2 = \hat{r}^T\hat{r} + \hat{r}^T X(\hat{w} - w) + (\hat{w} - w)^T X^T\hat{r} + (\hat{w} - w)^T X^T X(\hat{w} - w)$$

C.

As we know that $\hat{r}$ is orthogonal to X, we know that the product of $\hat{r}^T X = 0$, then if we replace:

$$\|r\|^2 = \hat{r}^T\hat{r} + (\hat{w} - w)^T X^T X(\hat{w} - w)$$

D.

As we know that $X^T X$ is positive definite matrix, which implies that $X^T X > 0$, and that $\|\hat{r}\|^2 = \hat{r}^T\hat{r}$. Then, the only way that $\|\hat{r}\|^2 = \|r\|^2$ is that $\hat{w} = w$ and for any w different that $\hat{w}$, as in a cuadratic form implies that $\|\hat{r}\|^2 < \|r\|^2$

Part 2

A.

We select the first vector $x_1 = \begin{pmatrix} 3 \\ 0 \\ 0 \end{pmatrix}$ and $\|x_1\| = \sqrt{3^2 + 0 + 0} = 3$, then $u_1 = \frac{x_1}{\|x_1\|} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$. Now have to obtain u_2, which could be defined as the residual between u_1 and x_2, then this is $x_2^{'} = x_2 - u_1(u_1^T x_2)$

$$x_2^{'} = x_2 - u_1(u_1^T x_2)$$

$$x_2^{'} = \begin{pmatrix} 1 \\ 3 \\ 4 \end{pmatrix} - \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} ((\begin{pmatrix} 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 3 \\ 4 \end{pmatrix})$$

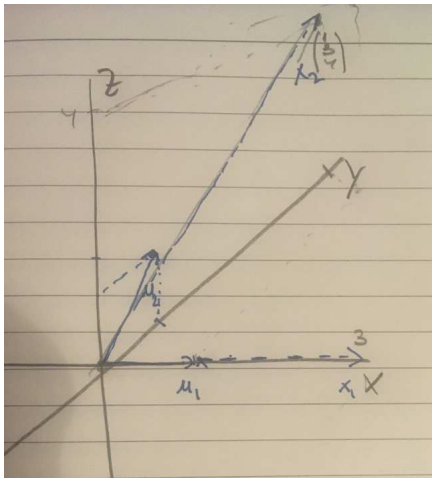
$$x_2^{'} = \begin{pmatrix} 1 \\ 3 \\ 4 \end{pmatrix} - \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

$$x_2^{'} = \begin{pmatrix} 0 \\ 3 \\ 4 \end{pmatrix}$$

Then the norm is $\left\|x_2^{'}\right\| = \sqrt{0 + 9 + 16} = 5$, then $u_2 = \frac{x_2^{'}}{\|x_2^{'}\|} = \begin{pmatrix} 0 \\ \frac{3}{5} \\ \frac{4}{5} \end{pmatrix}$.

Then $U = \left\{ \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ \frac{3}{5} \\ \frac{4}{5} \end{pmatrix} \right\}$

B.



C.

Now if we do the projection we got:

$$\hat{y} = X\left(X^T X\right)^{-1} X^T y$$

We use U insted of X and we know as U is a orthonormal matrix $U^T U = I$

$$\hat{y} = U\left(U^T U\right)^{-1} U^T y$$

$$\hat{y} = U U^T y$$

Now we replace for U and get:

$$\hat{y} = \begin{bmatrix} 1 & 0 \\ 0 & \frac{3}{5} \\ 0 & \frac{4}{5} \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \frac{3}{5} & \frac{4}{5} \end{bmatrix} \begin{pmatrix} 1 \\ 3 \\ 1 \end{pmatrix}$$

$$\hat{y} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \frac{9}{25} & \frac{12}{25} \\ 0 & \frac{12}{25} & \frac{16}{25} \end{bmatrix} \begin{pmatrix} 1 \\ 3 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ \frac{39}{25} \\ \frac{52}{25} \end{pmatrix}$$

Part 3

In the final part.

```
import numpy as np

def normalize(x):
rv = x/np.sqrt(np.dot(x.T,x))
return rv
def find_zero(x):
idx = np.argwhere(np.all(x[..., :] == 0, axis=0))
rv = np.delete(x, idx, axis=1)
return rv

def projection(U,X):
p1 = np.dot(U.T,X)
rv = np.dot(U,p1) return rv

def gs_algorithm(A):
a_nonzero = find_zero(A)
U = normalize(a_nonzero[:,0])
A_j=a_nonzero[:,1:]
n = np.shape(A_j)[1]
for i in range(n):
x_j= A_j[:,i]
x_j_prime = x_j - projection(U,x_j)
if x_j_prime.sum() == 0:
continue
U = np.c_[ U, normalize(x_j_prime)]
return U
```

Part 4

We know that U is nxn orthogonal matrix, $\sum$ is a nxp, which have to have rank of A, in this case implies that p=2. Then V^Tshould be also orthogonal of nxn. Also, we know that are the basis coeficient needed to represent each column of A. As the matrix A is only a diagonal matrix i will use an identity matrix as U, then as V is the matrix are coefficient to represent each column i will also use a identiy matrix because the value of A are positive. The the value matrix is the same as A:

$$A = U \sum V^T$$

$$A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Part 5

We know that U is nxn orthogonal matrix, $\sum$ is a nxp, which have to have rank of A, in this case implies that p=2. Then V^Tshould be also orthogonal of nxn.

Also, we know that are the basis coeficient needed to represent each column of A. As the matrix A is only a diagonal matrix i will use an identity matrix as U, then as V is the matrix are coefficient to represent each column i will also use a negative identiy matrix because the value of A are negative. The the value matrix is the same as A:

$$A = U \sum V^T$$

$$A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}$$

Part 6

A.

We can see, it's possible to decompose A into a SDV

$$A = U \sum V^T$$

$$A = \begin{bmatrix} | & & | \\ U_1 & \cdots & U_n \\ | & & | \end{bmatrix} \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_d \end{bmatrix} \begin{bmatrix} - & V_1 & - \\ & \vdots & \\ - & V_d & - \end{bmatrix}$$

We know that the product of $U \sum$ is :$\sum_{i=1} \begin{bmatrix} | \\ U_i \\ | \end{bmatrix} \begin{bmatrix} \sigma_i & 0 & ... & 0 \end{bmatrix}$, where the dimensions are nx1 and 1xd(rank- 1). But as the elements of $\sum$ that are not in the diagonal provokes that much of the n by d matrix are zero. Then taking into consideration we can show that, $U \sum = \begin{bmatrix} | & & | \\ \sigma_1 U_1 & \cdots & \sigma_n U_n \\ | & & | \end{bmatrix}$, then if we use the outer product distribution again we transform the SVD decompomspotion in the following form:

$$A = \sum_{i=1}^{min(n,d)} \sigma_i \begin{bmatrix} | \\ U_i \\ | \end{bmatrix} \begin{bmatrix} - & V_i & - \end{bmatrix}$$

B.

Using the same reasoning we can argue the same for a aproximation, as we know in a approximation of rank K we have only use K columns of U, a KxK matrix from $\sum$ and K rows of $V^T$of the SVD. Meaning the expresion is the same as in the previous question but the sumation for i until $min(d, n)$ is until K.

$$A = \sum_{i=1}^{k} \sigma_i \begin{bmatrix} | \\ U_i \\ | \end{bmatrix} \begin{bmatrix} - & V_i & - \end{bmatrix}$$

Part 7

A.

I will run a one against all model, saying that as we have three lables, we are train a model for each label. After that as value of the projection is a real value we are going to say if that is over 0.5, we are going to predict the label in the test set. Then we are going to compare this results with the original data.

Hw3 coding part

October 28, 2019

1 HW : 3 - Coding Part

Juan Vila

1.1 Part 3

[6]:
```
import numpy as np

def normalize(x):
    rv =  x/np.sqrt(np.dot(x.T,x))
    return rv

def find_zero(x):
    idx = np.argwhere(np.all(x[..., :] == 0, axis=0))
    rv = np.delete(x, idx, axis=1)
    return rv


def projection(U,X):
    p1 = np.dot(U.T,X)
    rv = np.dot(U,p1)
    return rv

def gs_algorithm(A):
    a_nonzero = find_zero(A)
    U = normalize(a_nonzero[:,0])
    A_j=a_nonzero[:,1:]
    n = np.shape(A_j)[1]
    for i in range(n):
        x_j= A_j[:,i]
        x_j_prime = x_j - projection(U,x_j)
        if x_j_prime.sum() == 0:
            continue
        U = np.c_[ U, normalize(x_j_prime)]

    return U
```

```
#checking with Part 2
x_2=np.array([[3,1],[0,3],[0,4]])
gs_algorithm(x_2)
```

[6]:
```
array([[1. , 0. ],
       [0. , 0.6],
       [0. , 0.8]])
```

1.2 Part 7

[12]:
```
import random
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt

iris=load_iris()
X = iris['data']
y = iris['target']


def beta_est(Y,x,cons=True):
    if cons:
        X = np.column_stack((x,np.ones([len(x),1])))
    else:
        X=x.copy()
    return np.dot(np.linalg.inv(np.dot(X.T,X)),np.dot(X.T,Y))

def proyection_2(Y,x, cons=True):
    if cons:
        X = np.column_stack((x,np.ones([len(x),1])))
    else:
        X=x.copy()
    y_hat = np.dot(X,beta_est(Y,x,cons))
    return y_hat

def normalization(X):
    rv = X.copy()
    for i in range(np.shape(X)[1]):
        mean = X[:,i].mean()
```

```
        sd = X[:,i].std()
        rv[:,i] = (X[:,i] - mean)/sd
    return rv


def cross_val(y,X,n_train, n_test, n):
    '''
    n_train: Size train set
    n_test: Size test set
    n: number of repetitions
    '''

    data = np.column_stack((y,X))

    #data_cv = np.split(data,n)
    rv =[]

    for i in range(n):
        np.random.shuffle(data)
        test_set  = np.array(random.sample(data.tolist(),n_test))
        train_set = np.array(random.sample(data.tolist(),n_train))
        y_test_orig, x_test_orig =  test_set[:,[0]], test_set[:,1:]
        y_train_orig, x_train_orig = train_set[:,[0]], train_set[:,1:]
        unique_train, counts_train = np.unique(y_test_orig, return_counts=True)
        unique_test, counts_test = np.unique(y_train_orig, return_counts=True)

        if len(unique_train) != 3 or len(unique_test) != 3:
            while len(unique_train) != 3 or len(unique_test) != 3:
                np.random.shuffle(data)
                test_set  = np.array(random.sample(data.tolist(),n_test))
                train_set = np.array(random.sample(data.tolist(),n_train))
                y_test_orig, x_test_orig =  test_set[:,[0]], test_set[:,1:]
                y_train_orig, x_train_orig = train_set[:,[0]], train_set[:,1:]
                unique_train, counts_train = np.unique(y_test_orig,␣
↪return_counts=True)
                unique_test, counts_test = np.unique(y_train_orig,␣
↪return_counts=True)

        count = 0
        for i in range(3):
            y_train = np.where(y_train_orig == i, 1, 0)
            y_test = np.where(y_test_orig == i, 1, 0)
            w = beta_est(y_train, x_train_orig)
            x_test = np.column_stack((x_test_orig,np.
↪ones([len(x_test_orig),1])))
            y_hat = np.dot(x_test,w)
            y_label_assig = np.where(y_hat>.5,1,0)
```

```
            for i,j in enumerate(y_test):
                if y_label_assig[i] != y_test[i]:
                    count+=1

        rv.append(count/(3 * n_test))
    rv=np.array(rv)

    return rv.mean()
```

1.2.1 7b

[13]:
```
random.seed(1234)
x_norm = normalization(X)
cross_val(y,x_norm,40,10,100)
```

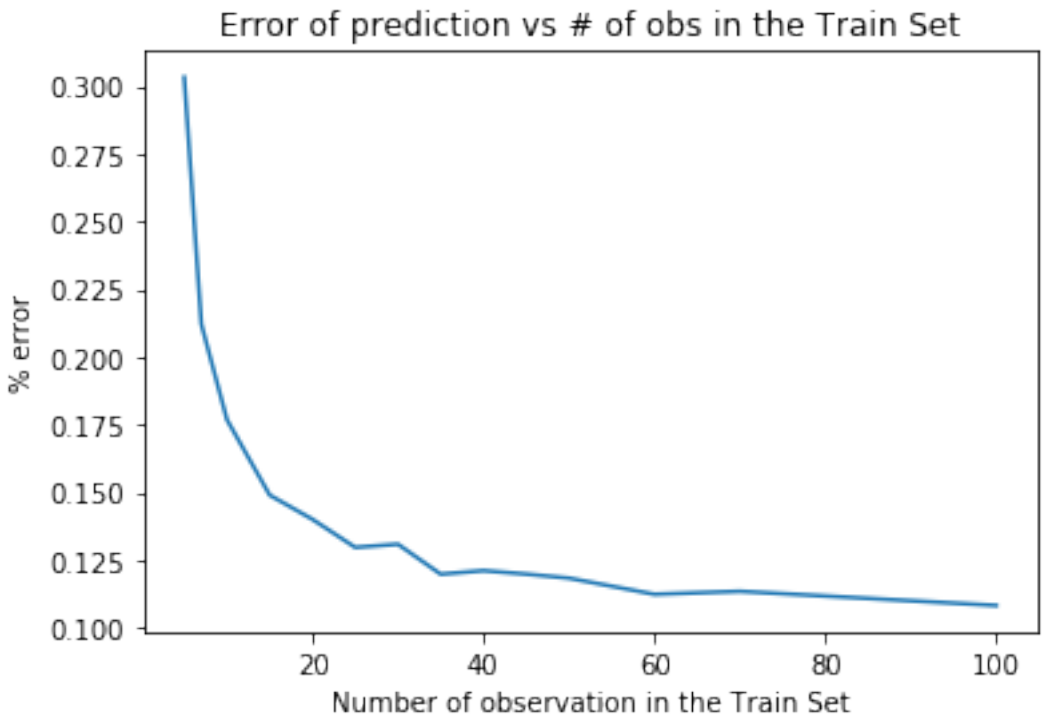
[13]:
```
0.13333333333333333
```

Comment: We can see that error is 13.33% on average of a trainning set of 40 obs and a test set of 10 obs, with 100 repetitions.

1.2.2 7c

[14]:
```
random.seed(1234)
train_nobs_set = [5,7,10,15,20,25,30,35,40,45,50,60,70,100]
test_result = []
for i in train_nobs_set:
    test_result.append(cross_val(y,x_norm,i,10,1000))
train_nobs_set=np.array(train_nobs_set)
test_result=np.array(test_result)
plt.plot(train_nobs_set,test_result)
plt.ylabel('% error')
plt.xlabel('Number of observation in the Train Set')
plt.title('Error of prediction vs # of obs in the Train Set')
```

[14]:
```
Text(0.5, 1.0, 'Error of prediction vs # of obs in the Train Set')
```



Comment: we can see as the size of the train set increase the error diminish.

1.2.3 7d

[15]:
```
x_d = x_norm[:,[0,1,2]]
random.seed(1234)
cross_val(y,x_d,40,10,100)
```

[15]:
```
0.13899999999999998
```

Comment: We see that if we run the model only with the three first variables the error increase.