hw4

November 5, 2022

[1]:
```
import numpy as np
import numpy.linalg as la
import matplotlib.pyplot as plt
import scipy.io as sio
import sys
```

1 1.

[2]:
```
def gram_schmidt(X):
    for i in range(X.shape[1]):
        if np.all(X[:, i] == 0):
            X = np.delete(X, i, axis=1)

    U = np.zeros(X.shape)
    U[:,0] = X[:,0] / sum(X[:,0]*X[:,0])**0.5
    to_delete = []
    for i in range(1, X.shape[1]):
        x_p = X[:,i] - U[:,:i]@U[:,:i].T@X[:,i]
        if np.allclose(x_p, 0):
            to_delete.append(i)
            continue
        U[:,i] = x_p / sum(x_p*x_p)**0.5


    return np.delete(U, to_delete, axis=1)
```

[3]:
```
X = np.array([[-1,1,2],[1,-1,2]])

gram_schmidt(X)
```

[3]:
```
array([[-0.70710678,  0.70710678],
       [ 0.70710678,  0.70710678]])
```

2 2.

2.1 a)

$u_1 = \begin{bmatrix} \sqrt{2} \\ 0 \end{bmatrix} / \sqrt{2} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$

$x_2' = \begin{bmatrix} 0 \\ \sqrt{7} \end{bmatrix} - \begin{bmatrix} 1 \\ 0 \end{bmatrix} \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ \sqrt{7} \end{bmatrix} = \begin{bmatrix} 0 \\ \sqrt{7} \end{bmatrix}$

$u_2 = \begin{bmatrix} 0 \\ \sqrt{7} \end{bmatrix} / \sqrt{7} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$

$u = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$

2.2 b)

2.2.1 i.

$P = v(v^t v)^{-1} v^t$

Since the middle term is a scalar

$P = \frac{vv^t}{v^t v}$

2.2.2 ii.

$||d_i||_2^2 = ||x_i - Px_i||_2^2$

2.2.3 iii.

$argmin_v \sum_{i=1}^{3} ||X - Px_i||_f^2$

if $v = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ or $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$ then the distance is 11. We can actually see that the magnitude in either direction (looking at the values in the 2 rows) is the same so if $v = \begin{bmatrix} 1 \\ 1 \end{bmatrix} / \sqrt{2}$ the distance would only be 5. The negative version of that would work as well in this case, but since P would be the same in either case that is unique while v is unique up to the sign.

2.3 c)

Since we know u_1 already, u_2 should just be orthogonal to it so:

$U = \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix} / \sqrt{2}$

Then

$\sigma_1 = ||Xu_1||_2 = || \begin{bmatrix} \sqrt{2} & 3 & 0 \\ 0 & 2 & \sqrt{7} \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} / \sqrt{2} ||_2 = || \begin{bmatrix} 1 \\ \frac{5}{\sqrt{2}} \\ \frac{7}{\sqrt{2}} \end{bmatrix} ||_2 = \sqrt{17}$

And

$$\sigma_2 = ||Xu_2||_2 = || \begin{bmatrix} \sqrt{2} & 3 & 0 \\ 0 & 2 & \sqrt{7} \end{bmatrix} \begin{bmatrix} -1 \\ 1 \end{bmatrix} / \sqrt{2}||_2 = || \begin{bmatrix} -1 \\ -\frac{1}{\sqrt{2}} \\ \frac{7}{\sqrt{2}} \end{bmatrix} ||_2 = \sqrt{5}$$

So

$$\Sigma = \begin{bmatrix} \sqrt{17} & 0 \\ 0 & \sqrt{5} \end{bmatrix}$$

3 3.

3.1 a)

$X = \sum_{i=1}^{r} \sigma_i u_i v_i^t$

3.2 b)

$X_k = \sum_{i=1}^{k} \sigma_i u_i v_i^t$ where $k < r$

4 4.

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 5 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

5 5.

$$\begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 3 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

6 7.

6.1 a) and b)

[4]:
```
d = sio.loadmat('face_emotion_data.mat')
X = d['X']
y = d['y']

X_chunks = np.split(X, 8)
y_chunks = np.split(y, 8)
svd_errors = []
rls_errors = []
for i in range(8):
    for j in range(8):
        if i == j:
            continue
        x_valid = X_chunks[i]
        y_valid = y_chunks[i]
        x_test = X_chunks[j]
        y_test = y_chunks[j]
```

```
        X_train = np.concatenate(np.delete(X_chunks, [i, j], axis=0))
        y_train = np.concatenate(np.delete(y_chunks, [i, j], axis=0))

        U, svs, V = la.svd(X_train)

        best_error = 1
        best_w = None
        for k in range(1, 10):
            n, p = np.shape(X_train)
            S = np.zeros((n,p))
            for s in range(k):
                S[s,s] = svs[s]
            S_pi = la.pinv(S)
            w = V@S_pi@U.T@y_train

            y_tilde = x_valid@w
            y_hat = np.where(y_tilde > 0, 1, -1)
            v_error = ((y_hat - y_valid)!=0).sum()/16

            if v_error < best_error:
                best_w = w

        y_tilde = x_test@best_w
        y_hat = np.where(y_tilde > 0, 1, -1)
        error = ((y_hat - y_test)!=0).sum()/16
        svd_errors.append(error)

        best_error = 1
        best_w = None
        n, p = np.shape(X_train)
        S = np.zeros((n,p))
        for s in range(len(svs)):
            S[s,s] = svs[s]

        S_pi = la.pinv(S)
        for lam in [0, 0.5, 1, 2, 4, 8, 16]:
            w = V@la.inv(S.T@S + (lam * np.identity(p)))@S.T@U.T@y_train
            y_tilde = x_valid@w
            y_hat = np.where(y_tilde > 0, 1, -1)
            v_error = ((y_hat - y_valid)!=0).sum()/16

            if v_error < best_error:
                best_w = w

        y_tilde = x_test@best_w
        y_hat = np.where(y_tilde > 0, 1, -1)
        error = ((y_hat - y_test)!=0).sum()/16
```

```
        rls_errors.append(error)

print(f"SVD Error Rate: {np.array(svd_errors).mean()}")
print(f"RLS Error Rate: {np.array(rls_errors).mean()}")
```

```
SVD Error Rate: 0.5223214285714286
RLS Error Rate: 0.5345982142857143
```

6.2 c)

No we wouldn't expect these new features to be helpful since they are made from already existing features. They aren't providing any new information.

[5]:
```
val1 = (X * np.random.randint(-5, 5, (9))).sum(axis=1)
val2 = (X * np.random.randint(-5, 5, (9))).sum(axis=1)
val3 = (X * np.random.randint(-5, 5, (9))).sum(axis=1)
for val in [val1, val2, val3]:
    X = np.append(X, val.reshape(-1,1), axis=1)
```

[6]:
```
X_chunks = np.split(X, 8)
y_chunks = np.split(y, 8)
svd_errors = []
rls_errors = []
for i in range(8):
    for j in range(8):
        if i == j:
            continue
        x_valid = X_chunks[i]
        y_valid = y_chunks[i]
        x_test = X_chunks[j]
        y_test = y_chunks[j]
        X_train = np.concatenate(np.delete(X_chunks, [i, j], axis=0))
        y_train = np.concatenate(np.delete(y_chunks, [i, j], axis=0))

        U, svs, V = la.svd(X_train)

        best_error = 1
        best_w = None
        for k in range(1, 10):
            n, p = np.shape(X_train)
            S = np.zeros((n,p))
            for s in range(k):
                S[s,s] = svs[s]
            S_pi = la.pinv(S)
            w = V@S_pi@U.T@y_train

            y_tilde = x_valid@w
            y_hat = np.where(y_tilde > 0, 1, -1)
```

```
            v_error = ((y_hat - y_valid)!=0).sum()/16

            if v_error < best_error:
                best_w = w

        y_tilde = x_test@best_w
        y_hat = np.where(y_tilde > 0, 1, -1)
        error = ((y_hat - y_test)!=0).sum()/16
        svd_errors.append(error)

        best_error = 1
        best_w = None
        n, p = np.shape(X_train)
        S = np.zeros((n,p))
        for s in range(len(svs)):
            S[s,s] = svs[s]
        S_pi = la.pinv(S)
        for lam in [0, 0.5, 1, 2, 4, 8, 16]:
            w = V@la.inv(S.T@S + (lam * np.identity(p)))@S.T@U.T@y_train
            y_tilde = x_valid@w
            y_hat = np.where(y_tilde > 0, 1, -1)
            v_error = ((y_hat - y_valid)!=0).sum()/16

            if v_error < best_error:
                best_w = w

        y_tilde = x_test@best_w
        y_hat = np.where(y_tilde > 0, 1, -1)
        error = ((y_hat - y_test)!=0).sum()/16
        rls_errors.append(error)

print(f"SVD Error Rate: {np.array(svd_errors).mean()}")
print(f"RLS Error Rate: {np.array(rls_errors).mean()}")
```

```
SVD Error Rate: 0.47767857142857145
RLS Error Rate: 0.4497767857142857
```

[ ]: