# Linear Classifier

December 2, 2020

[2]:
```
import numpy as np
from scipy.io import loadmat
import matplotlib.pyplot as plt
from sklearn.svm import LinearSVC

in_data = loadmat('CardioDataUpdatedFile.mat')
# 11 features age, height, weight, gender, systolic blood pressure, diastolic␣
 ↪blood pressure, cholesterol, glucose level, smoking, alcohol intake, and␣
 ↪physical activity

x = in_data['X']

A = x[:,1:12] # Matrix A with all the features
A[:,1] = A[:,1] / 365 # Change age from days to years.
d = x[:,12] # Target variable d
print('Matrix rank = ',np.linalg.matrix_rank(A))
print('Matrix is full rank, unique solution exist.')
# Now the Least squares
```

```
Matrix rank =  11
Matrix is full rank, unique solution exist.
```

[3]:
```
# Simple Least Square Sum using all features
w = np.linalg.inv((A.T@A))@A.T@d
error = np.mean(np.sign(A@w)!=d)
print('Without any altercation to our data and using a simple least squares␣
 ↪with all the features we get an error rate of', error*100)

x_sub = A[:,0:8] # Using the first 8 features
w_sub = np.linalg.inv((x_sub.T@x_sub))@x_sub.T@d
error_sub = np.mean(np.sign(x_sub@w_sub)!=d)
print('\nRemoving the data the features that are provided by the patient we get␣
 ↪a similar error rate: ', error_sub*100)
```

```
Without any altercation to our data and using a simple least squares with all
the features we get an error rate of 50.029999999999994

Removing the data the features that are provided by the patient we get a similar
```

error rate:  50.029999999999994

[4]:

```
# Using singular value decomposition
A = A[:,0:8]
U,s,VT = np.linalg.svd(A, full_matrices = False)
w_pred = (1/s)*VT.transpose()@U.transpose()@d
error = np.mean(np.sign(A@w_pred)!=d)
print('Using a simple SVD we get the same error', error)

# Know use training and validation sets
# Sets of 70000
x_train = np.array(list(range(0,56000)))
hold_1 = np.array(list(range(56000,63000)))
hold_2 = np.array(list(range(63000,70000)))
x_train = np.vstack((x_train, (x_train+7000)%70000))
hold_1 = np.vstack((hold_1, (hold_1+7000)%70000))
hold_2 = np.vstack((hold_2, (hold_2+7000)%70000))


for x in range(8):
    x_train = np.vstack((x_train, (x_train[x+1]+7000)%70000))
    hold_1 = np.vstack((hold_1, (hold_1[x+1]+7000)%70000))
    hold_2 = np.vstack((hold_2, (hold_2[x+1]+7000)%70000))


# Now have different training and hold out sets
err_list2 = []
err_list3 = []
r_prime = 0; # Assume r_prime = 0


for i in range(10):
    for r in range(8):
        X_train = A[x_train[i]]
        U,s,VT = np.linalg.svd(X_train, full_matrices = False)
        w_pred = (1/s[0:r+1])*VT.transpose()[:,0:r+1]@U.transpose()[0:r+1,:
 ↪]@d[x_train[i]]
        y_pred = np.sign(A[hold_1[i]]@w_pred)
        err_list2.append(np.mean(d[hold_1[i]]!=y_pred))
        if r > 0:
            if err_list2[r_prime] > err_list2[r]:
                r_prime = r
        if r > 6: # Should only be ran once after finding optimal r
            w_pred = VT.transpose()[:,0:r_prime+1]*(1/s[0:r_prime+1])@U.
 ↪transpose()[0:r_prime+1,:]@d[x_train[i]]
            y_pred = np.sign(A[hold_2[i]]@w_pred)
            err_list3.append(np.mean(d[hold_2[i]]!=y_pred))
```

```
    r_prime = 0
    err_list2 = []
    for r in range(8): # repeat for different combination of hold outs
        X_train = A[x_train[i]]
        U,s,VT = np.linalg.svd(X_train, full_matrices = False)
        w_pred = (1/s[0:r+1])*VT.transpose()[:,0:r+1]@U.transpose()[0:r+1,:
↪]@d[x_train[i]]
        y_pred = np.sign(A[hold_2[i]]@w_pred)
        err_list2.append(np.mean(d[hold_2[i]]!=y_pred))
        if r > 0:
            if err_list2[r_prime] > err_list2[r]:
                r_prime = r

# Find error on hold out set 2 given ideal r
        if r > 6: # Should only be ran once after finding optimal r
            w_pred = VT.transpose()[:,0:r_prime+1]*(1/s[0:r_prime+1])@U.
↪transpose()[0:r_prime+1,:]@d[x_train[i]]
            y_pred = np.sign(A[hold_1[i]]@w_pred)
            err_list3.append(np.mean(d[hold_1[i]]!=y_pred))

print(err_list3)
print(len(err_list3))

avg_error = np.mean(err_list3)
print("Average error rate for truncated SVD is ", avg_error*100)
```

```
Using a simple SVD we get the same error 0.5003
[0.49685714285714283, 0.502, 0.4948571428571429, 0.49685714285714283, 0.515,
0.4948571428571429, 0.509, 0.515, 0.49328571428571427, 0.509,
0.4948571428571429, 0.49328571428571427, 0.49814285714285716,
0.4948571428571429, 0.4997142857142857, 0.49814285714285716, 0.4992857142857143,
0.4997142857142857, 0.502, 0.4992857142857143]
20
Average error rate for truncated SVD is  50.029999999999994
```

[5]:
```
# Method 2 = ridge regression

lambdas = [0,0.5,1,2,4,8,16]



err_list2 = []
err_list3 = []
r_prime = 0; # Assume r_prime = 0

# Find optimum r for w by testing on hold out set 1
```

```
for i in range(10):
    for r in range(7):
        X_train = A[x_train[i]]
        U,s,VT = np.linalg.svd(X_train, full_matrices = False)
        sigma_inv = s / (s*s + lambdas[r])
        w_pred = sigma_inv*VT.transpose()@U.transpose()@d[x_train[i]]
        y_pred = np.sign(A[hold_1[i]]@w_pred)
        err_list2.append(np.mean(d[hold_1[i]]!=y_pred))
        if r > 0:
            if err_list2[r_prime] > err_list2[r]:
                r_prime = r
                print('r prime', r_prime)
        # Find error on hold out set 2 given ideal r
        if r > 5: # Should only be ran once after finding optimal r
            sigma_inv = s / (s*s + lambdas[r_prime])
            w_pred = sigma_inv*VT.transpose()@U.transpose()@d[x_train[i]]
            y_pred = np.sign(A[hold_2[i]]@w_pred)
            err_list3.append(np.mean(d[hold_2[i]]!=y_pred))

    r_prime = 0
    err_list2 = []

    for r in range(7): # repeat for different combination of hold outs
        X_train = A[x_train[i]]
        U,s,VT = np.linalg.svd(X_train, full_matrices = False)
        sigma_inv = s / (s*s + lambdas[r])
        w_pred = sigma_inv*VT.transpose()@U.transpose()@d[x_train[i]]
        y_pred = np.sign(A[hold_2[i]]@w_pred)
        err_list2.append(np.mean(d[hold_2[i]]!=y_pred))
        if r > 0:
            if err_list2[r_prime] > err_list2[r]:
                r_prime = r

# Find error on hold out set 2 given ideal r
        if r > 5: # Should only be ran once after finding optimal r
            sigma_inv = s / (s*s + lambdas[r])
            w_pred = sigma_inv*VT.transpose()@U.transpose()@d[x_train[i]]
            y_pred = np.sign(A[hold_1[i]]@w_pred)
            err_list3.append(np.mean(d[hold_1[i]]!=y_pred))

print(err_list3)
print(len(err_list3))
avg_error = np.mean(err_list3)
print("Average error rate for ridge regression is ", avg_error*100)
```

```
[0.49685714285714283, 0.502, 0.4948571428571429, 0.49685714285714283, 0.515,
0.4948571428571429, 0.509, 0.515, 0.49328571428571427, 0.509,
```

```
0.4948571428571429, 0.49328571428571427, 0.49814285714285716,
0.4948571428571429, 0.4997142857142857, 0.49814285714285716, 0.4992857142857143,
0.4997142857142857, 0.502, 0.4992857142857143]
20
Average error rate for ridge regression is  50.029999999999994
```

[7]:
```
#SVM
from sklearn import svm
# Train classifier using linear SVM from SK Learn library
# 90% f data to train
# 10% of data to test

x_train = np.array(list(range(0,63000)))
x_val = np.array(list(range(63000,70000)))
x_train = np.vstack((x_train, (x_train+7000)%70000))
x_val = np.vstack((x_val, (x_val+7000)%70000))

for x in range(8):
    x_train = np.vstack((x_train, (x_train[x+1]+7000)%70000))
    x_val = np.vstack((x_val, (x_val[x+1]+7000)%70000))

err_list2 = []
for x in range(7):
    clf = svm.SVC()
    x_svm_tr = A[x_train[x]]
    y_svm_tr = d[x_train[x]]
    clf.fit(x_svm_tr, y_svm_tr)
    ypred = clf.predict(A[x_val[x]])
    err_list2.append(np.mean(d[x_val[x]]!=y_pred))
```

[11]:
```
print(err_list2)
print(np.mean(err_list2))
```

```
[0.49685714285714283, 0.4948571428571429, 0.515, 0.509, 0.49328571428571427,
0.4948571428571429, 0.49814285714285716]
0.5002857142857143
```

[ ]:
```
```