Ex5

October 14, 2020

0.1 Exercise Set 5

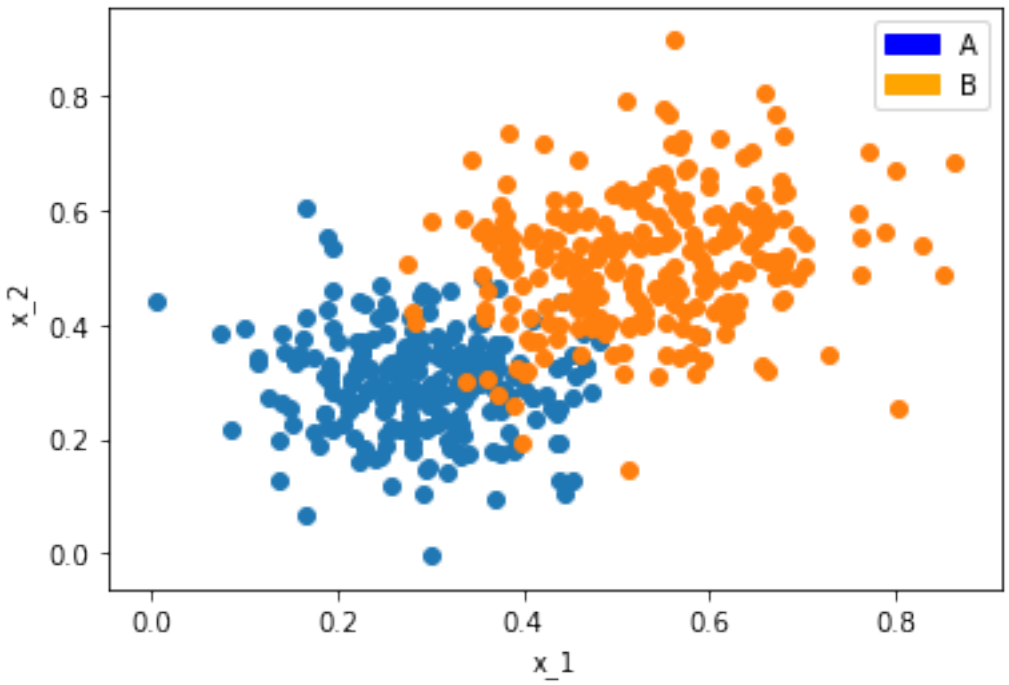
1 Jake Muff

2 Question 1: Fisher DIscriminant

Part 1. Plotting the data points from the two samples in the x_1 x_2 plane

[199]:
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches


A = np.loadtxt('DatasetA.txt')
B = np.loadtxt('DatasetB.txt')


A_1 = np.loadtxt("DatasetA.txt")[:,0 ] #first column of data set A
A_2 = np.loadtxt("DatasetA.txt")[:,1 ] #second column of data set A
B_1 = np.loadtxt("DatasetB.txt")[:,0 ] #first column of data set B
B_2 = np.loadtxt("DatasetB.txt")[:,1 ] #second column of data set B
plt.scatter(A_1, A_2)
plt.scatter(B_1, B_2)
blue_patch = mpatches.Patch(color='blue', label='A')
orange_patch = mpatches.Patch(color='orange', label='B')
plt.legend(handles=[blue_patch, orange_patch])
plt.xlabel('x_1')
plt.ylabel('x_2')
```

[199]:
```
Text(0, 0.5, 'x_2')
```

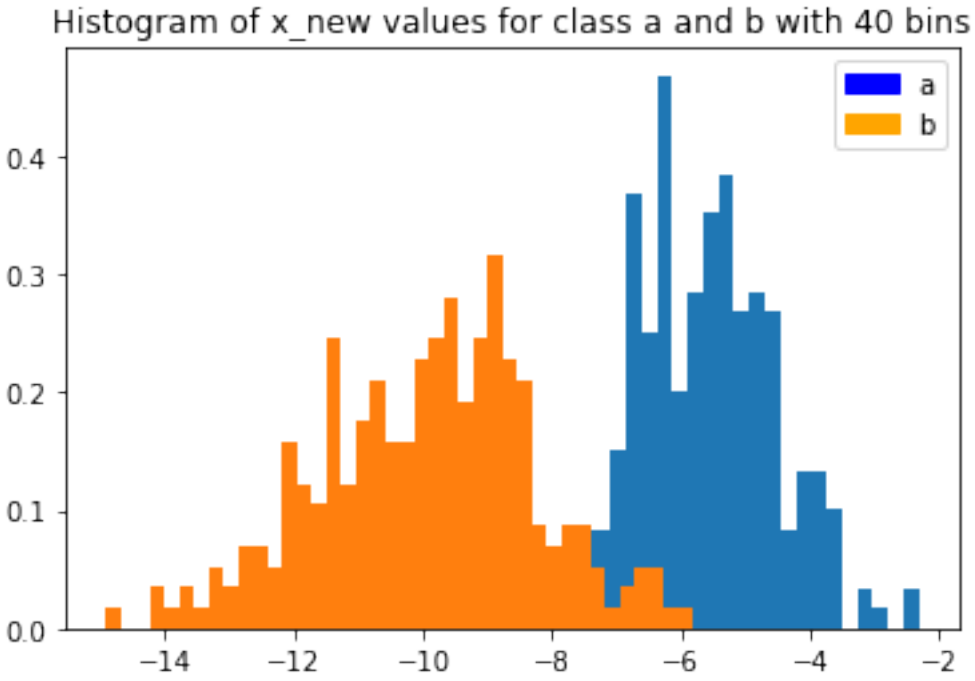


Part 2. Calculating vector c. For this is used a numpy algortihm to calculate the inverse. I also assumed by expectation values the question meant average or mean values. The question was then simply following the equation given.

[202]:
```
V_a = np.cov(A_1, A_2)
V_b = np.cov(B_1, B_2)
#print((V_a + V_b))
V = np.linalg.inv(V_a+V_b)


mu_a1 = np.mean(A_1)
mu_a2 = np.mean(A_2)
mu_b1 = np.mean(B_1)
mu_b2 = np.mean(B_2)
mu_a = np.array([[mu_a1], [mu_a2]])
mu_b = np.array([[mu_b1], [mu_b2]])
#print(mu_a- mu_b)


y= (mu_a - mu_b)
c=np.matmul(V,y)

x_newa = c[0] * A_1 + c[1] * A_2
x_newb = c[0] * B_1 + c[1] * B_2
counta, binsa, ignored = plt.hist(x_newa, 40, density=True)
```

```
countb, binsb, ignored = plt.hist(x_newb, 40, density=True)
blue_patch = mpatches.Patch(color='blue', label='a')
orange_patch = mpatches.Patch(color='orange', label='b')
plt.legend(handles=[blue_patch, orange_patch])
plt.title('Histogram of x_new values for class a and b with 40 bins')
print('Vector c:')
print(c)
```

```
Vector c:
[[-10.28518883]
 [ -8.83421184]]
```



Part 3. This was the hardest part. To calculate the 96% rejection I simply parsed the x_new for class b into a new array for which normalized values were greater than or equal to 0.96. I also found another way using interp1d which gave roughly the same value, however it did spit out an error.

I believe the result could be improved by changing the way my bin widths were calculated. The plots were shown below with both rejection lines calculated and the cdf's plotted with them. The cdfs plotted were also not connected as I thought they should be.

[214]:
```
from scipy.interpolate import interp1d
from scipy import optimize
```

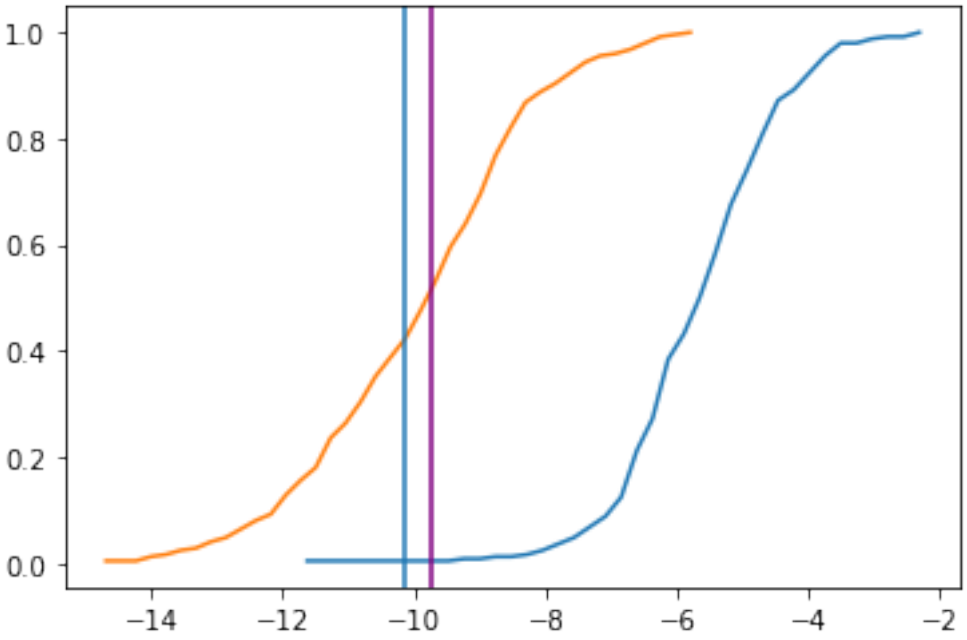
```
#print(len(x_newb))
dx_a = binsa[1]-binsa[0] #from 0 to 1
dx_b = binsb[1] - binsb[0]

F1 = np.cumsum(counta*dx_a)
F2 = np.cumsum(countb * dx_b)

plt.plot(binsa[1:], F1)
plt.plot(binsb[1:], F2)

print(binsa[9])
print(F1[9])
print(binsb[9])
print(F2[9])

for x in range(250):
    if((x/250)>= 0.96):
        x_new_g = x_newb[x]

print(x_new_g)
plt.axvline(x_new_g, color='purple')

bins_b = 40

bin_widths_b = np.histogram_bin_edges(countb, binsb[1:])
 #b y axis for hist above
print(len(x_newb), len(bin_widths_b))
cdf_b = scipy.interpolate.interp1d(countb, np.cumsum(bins_b * bin_widths_b),␣
 ↪bounds_error=False, fill_value=(0, 1))



sol = scipy.optimize.root_scalar(
    lambda x: cdf_b(x) - 0.96,
    x0=(x_newb.mean() + x_newb.min()) / 2,
    x1=(x_newb.mean() + x_newb.max()) / 2)
print(sol)
plt.axvline(sol.root)
```

```
-9.7129166747385
0.004
-12.86338950288135
0.06399999999999981
-9.731877714303526
250 40
      converged: True
```

```
           flag: 'converged'
 function_calls: 2
     iterations: 1
           root: -10.154080307521845
```

```
/usr/local/anaconda3/lib/python3.7/site-packages/scipy/optimize/zeros.py:343:
RuntimeWarning: Tolerance of 4.560077054074419 reached.
  warnings.warn(msg, RuntimeWarning)
```

[214]: `<matplotlib.lines.Line2D at 0x7fed53f10550>`



Q2

October 14, 2020

0.1 Question 2: Kolmogorov-Smirnov Test

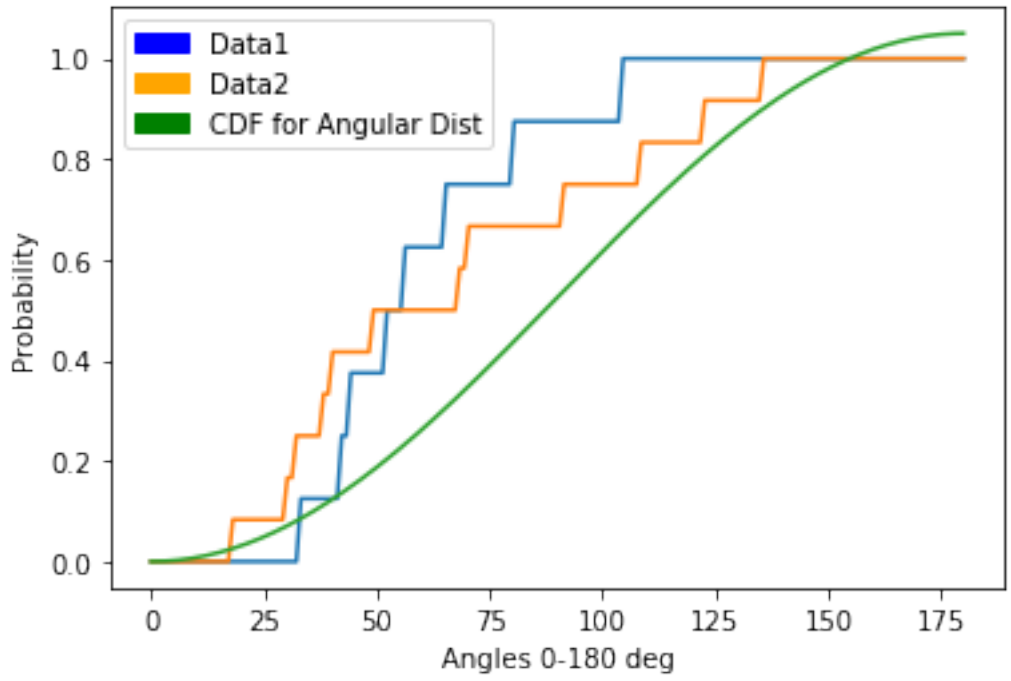
[204]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy import stats
from scipy.stats import norm
from scipy.interpolate import interp1d
from scipy import optimize
from scipy.special import kolmogorov
from scipy.stats import kstwobign
import matplotlib.patches as mpatches

Data1 = [80, 44, 56, 65, 33, 52, 42, 104]
Data2 = [18, 40, 108, 70, 135, 68, 32, 30, 38, 122, 49, 91]
Data3 = Data1+Data2

test = []
for x in range(181):
    test.append(x)


Data1_sorted = np.sort(Data1)
Data2_sorted = np.sort(Data2)
Data3_sorted = np.sort(Data3)


#stats.kstest(Data1, 'norm')
kst = stats.ks_2samp(Data1, Data2)
print(kst)

#print(np.sort(Data1))
#DataY1 = np.cumsum(np.sort(Data1))/ np.sum(Data1)
#DataY2 = np.cumsum(np.sort(Data2))/np.sum(Data2)

Datay1 = np.linspace(1/len(Data1), 1, len(Data1))
Datay2 = np.linspace(1/len(Data2), 1, len(Data2))
Datay3 = np.linspace(1/len(Data3), 1, len(Data3))
```

```
F1 = interp1d(np.sort(Data1), Datay1, kind="previous", bounds_error = False,␣
 ↪fill_value = (0,1)) #interpolation for Data set 1
F2 = interp1d(np.sort(Data2), Datay2, kind="previous", bounds_error = False,␣
 ↪fill_value = (0,1)) #interpolation for data set 2
F3 = interp1d(np.sort(Data3), Datay3, kind="previous", bounds_error = False,␣
 ↪fill_value = (0,1))



diffmax =0
for deg in Data1+Data2:
    diff = np.abs(F1(deg)-F2(deg))
    diffmax= diff if diff > diffmax else diffmax


print('KS Distance: %f' %diffmax) #i


S = np.sqrt((len(Data1)*len(Data2)) / (len(Data1)+len(Data2))) #critical values␣
 ↪(i.e big sqrt)


P_KS = kolmogorov(diffmax*S) #here we dont use 1- F_k because kolmogorov from␣
 ↪scipy gives the complementary cum dist
print('P_KS: %f' %P_KS) #ii

gamma = 0.1
thetaD = np.linspace(0,180,180)
thetaR = thetaD*(np.pi/180)
#E = (gamma*np.sin(theta)+theta)*(180/np.pi) #integral of dist over theta
def angular_cdf(theta, gamma):
    return -0.5*(1.+gamma/2.)*(np.cos(theta) - 1.)

def angular(theta, gamma):
    return 1 + gamma*np.cos(theta)
#calculating diff_max for expected vs data3=data1+data2
KSD_A = 0
for angle in Data3:
    diff_A = np.abs(angular_cdf(angle, gamma) - F3(angle))
    KSD_A = diff_A if diff_A > KSD_A else KSD_A


print('KS Distance for angular dist: %f' %KSD_A)

S_A = np.sqrt((len(Data1)+len(Data2)))
P_KS_A = kolmogorov(KSD_A*S_A) # 0.35456057524597623
```

```
print('P_KS for angular dist: %f' %(P_KS_A))

x = np.linspace(0,180,180)
plt.plot(x, F1(x))
plt.plot(x,F2(x))
plt.plot(x, angular_cdf(thetaR,gamma))
blue_patch = mpatches.Patch(color='blue', label='Data1')
orange_patch = mpatches.Patch(color='orange', label='Data2')
green_patch =  mpatches.Patch(color='green', label='CDF for Angular Dist')
plt.legend(handles=[blue_patch, orange_patch, green_patch])
plt.xlabel('Angles 0-180 deg')
plt.ylabel('Probability')
```

```
Ks_2sampResult(statistic=0.2916666666666667, pvalue=0.7432404540763674)
KS Distance: 0.291667
P_KS: 0.808819
KS Distance for angular dist: 0.575142
P_KS for angular dist: 0.000004
```

[204]: `Text(0, 0.5, 'Probability')`



As we can see the two angle distributions (Data1 & Data2) out calculations indicate that they do indded originate from the same source as the P value is fairly high. The null hypothesis (H_0) is that the sample follows the distribution and we accept it. Interestingly the in built stats.ks_2samp

gives a more accurate result but still similar. The test statistic is also less than the critical value up to a signifiance level of 0.001 which correlates our results.

For the combined data set vs the expected distribution we find that the two samples were from populations with different distributions i.e originate from different sources and they are not compatible.

Even though my results lead me to what I presume is the correct answer I believe my actual numbers for the P-Value and D-test statistic are off but I was unable to find out why.

[ ]: