STEP 1: wood.py

This can be ran by: **python wood.py 10 hw2_training.txt hw2_testing.txt**
where these stand for: python wood.py k trainData testData

STEP 2:

P(V=0|C=0)=0.214285714286
P(V=0|C=1)=0.0
P(V=0|C=2)=0.132530120482
P(V=0|C=3)=0.0
P(V=0|C=4)=0.521739130435
P(V=0|C=5)=0.0
P(V=0|C=6)=0.0
P(V=1|C=0)=0.571428571429
P(V=1|C=1)=0.0
P(V=1|C=2)=0.180722891566
P(V=1|C=3)=0.025974025974
P(V=1|C=4)=0.188405797101
P(V=1|C=5)=0.046875
P(V=1|C=6)=0.0
P(V=2|C=0)=0.0
P(V=2|C=1)=0.0
P(V=2|C=2)=0.0
P(V=2|C=3)=0.480519480519
P(V=2|C=4)=0.0
P(V=2|C=5)=0.015625
P(V=2|C=6)=0.0
P(V=3|C=0)=0.178571428571
P(V=3|C=1)=0.0
P(V=3|C=2)=0.686746987952
P(V=3|C=3)=0.0
P(V=3|C=4)=0.0
P(V=3|C=5)=0.0
P(V=3|C=6)=0.0
P(V=4|C=0)=0.0
P(V=4|C=1)=0.0
P(V=4|C=2)=0.0
P(V=4|C=3)=0.012987012987
P(V=4|C=4)=0.0
P(V=4|C=5)=0.515625
P(V=4|C=6)=0.0
P(V=5|C=0)=0.0
P(V=5|C=1)=0.0
P(V=5|C=2)=0.0
P(V=5|C=3)=0.233766233766
P(V=5|C=4)=0.0
P(V=5|C=5)=0.359375
P(V=5|C=6)=0.0
P(V=6|C=0)=0.0
P(V=6|C=1)=0.987179487179
P(V=6|C=2)=0.0
P(V=6|C=3)=0.0
P(V=6|C=4)=0.0
P(V=6|C=5)=0.0
P(V=6|C=6)=0.0
P(V=7|C=0)=0.0
P(V=7|C=1)=0.0
P(V=7|C=2)=0.0
P(V=7|C=3)=0.246753246753
P(V=7|C=4)=0.0
P(V=7|C=5)=0.015625
P(V=7|C=6)=0.0
P(V=8|C=0)=0.0
P(V=8|C=1)=0.0128205128205
P(V=8|C=2)=0.0
P(V=8|C=3)=0.0
P(V=8|C=4)=0.0
P(V=8|C=5)=0.0
P(V=8|C=6)=0.972602739726
P(V=9|C=0)=0.0357142857143
P(V=9|C=1)=0.0
P(V=9|C=2)=0.0
P(V=9|C=3)=0.0
P(V=9|C=4)=0.289855072464
P(V=9|C=5)=0.046875
P(V=9|C=6)=0.027397260274
P(C=5)=0.128
P(C=6)=0.146
P(C=1)=0.156
P(C=3)=0.154
P(C=0)=0.112
P(C=2)=0.166
P(C=4)=0.138

STEP 3: wood.py method test

Classification Error Rate Mean: 0.244166666667 STD: 0.0144577929774

STEP 4:

k=2:	Classification Error Rate Mean: 0.675	STD: 0.0
3:	Classification Error Rate Mean: 0.56	STD: 0.0133333333333
5:	Classification Error Rate Mean: 0.378333333333	STD: 0.0489046691704
6:	Classification Error Rate Mean: 0.258333333333	STD: 0.0562731433871
8:	Classification Error Rate Mean: 0.245833333333	STD: 0.0284434136098
12:	Classification Error Rate Mean: 0.260833333333	STD: 0.0182764268329
15:	Classification Error Rate Mean: 0.2575	STD: 0.0114564392374
20:	Classification Error Rate Mean: 0.259166666667	STD: 0.0146486631927
50 :	Classification Error Rate Mean: 0.2375	STD: 0.0164249335536
100:	Classification Error Rate Mean: 0.225	STD: 0.0139885237069

From sampling the average and standard deviation of the classification error rate for k-means using k=2,3,5,6,8,12,15,20,50, and 100 we notice the following. We notice that the standard deviation is relatively low (0.01 to 0.05), and as k grows, the STD goes from low to high then back to low. This intuitively makes sense because when k is low, <5, there are only so many clusters that can be made, thus the mean will be the same through the trials. Then when k is around 5 there are many ways that the clusters could be made and each resulting with different results depending on where they are initialized, causing the higher STD. When k grows large all of the clusters will begin to resemble each other from different trials because there are so many clusters that it is not important where they are initialized. As k grows the mean decreases with less amount for each k. This makes sense because as there are more groups, the data will be better classified into groups with those around it. Thus having groups of better related data and being able to better classify the data.

BONUS 1: wood2.py

This can be ran by: **python wood2.py 10 hw2_training.txt hw2_testing.txt 3**

Where these stand for: python wood2.py k trainData testData dimensions

This may take a minute, it will finish

dimensions can be any integer

generate data by editing generate_data.py dims to {dimensions}

dim=2:Classification Error Rate Mean: 0.254166666667 STD: 0.0264181124736

3: Classification Error Rate Mean: 0.2025 STD: 0.0123883906228

4: Classification Error Rate Mean: 0.0658333333333 STD: 0.0790964600978

5: Classification Error Rate Mean: 0.0183333333333 STD: 0.055

5: Classification Error Rate Mean :0.0225 STD: 0.0647698918394

5: Classification Error Rate Mean: 0.0191666666667 STD: 0.0383785964656

6: Classification Error Rate Mean: 0.0241666666667 STD: 0.0531572614461

6: Classification Error Rate Mean: 0.0416666666667 STD: 0.092870878105

8: Classification Error Rate Mean: 0.0933333333333 STD: 0.0781558272513

20: Classification Error Rate Mean: 0.065 STD: 0.080777472107

As the dimension increases the Classification tends to decrease until a certain point (dim>5) then it appears to increase slightly and have no correlation to the number of dimensions. This makes sense because as the dimensions increase, it creates more spaces in the model and allows the data that is closely related to have multiple dimensions to be close to each other and eliminate other data that is only close in a couple dimensions. Once the dimensions become very large then perhaps there is too much space, too much freedom, for the data and even data that is closely related will be too far apart for an accurate grouping.

BONUS 2: wood.py

This can be ran by: **python wood.py 10 hw2_training.txt hw2_testing.txt**
where these stand for: python wood.py k trainData testData
generate data by editing generate_data.py width

width = 0.1:	Classification Error Rate Mean: 0.0658333333333	STD: 0.0673351897434
1:	Classification Error Rate Mean: 0.0825	STD: 0.0475
3.2:	Classification Error Rate Mean: 0.173333333333	STD: 0.0268224615657
5:	Classification Error Rate Mean: 0.263333333333	STD: 0.0230337916018
10:	Classification Error Rate Mean: 0.576666666667	STD: 0.0328718048722
20:	Classification Error Rate Mean: 0.75	STD: 0.0266145323711
50:	Classification Error Rate Mean: 0.82	STD: 0.030776975521
100:	Classification Error Rate Mean: 0.868333333333	STD: 0.037969285833
1000:	Classification Error Rate Mean: 0.8225	STD: 0.0320698023415

As the Gaussian widths of the data generated for 2d data increased the data overlaps more, this results in the mean increasing. This is because there is data from different classes mixed in with each other so that it is impossible for k-means to group them separately. The mean will increase until it reaches the point of just randomly assigning the data to a group (~82%). The standard deviation is unchanged by the width increasing.

BONUS 3: wood.py

This can be ran by: **python wood.py 10 hw2_training.txt hw2_testing.txt**
where these stand for: python wood.py k trainData testData
Comment out line 51 and 65 and uncomment 52-55 and 66-69 in generate_data and edit the numbers there to change the weights

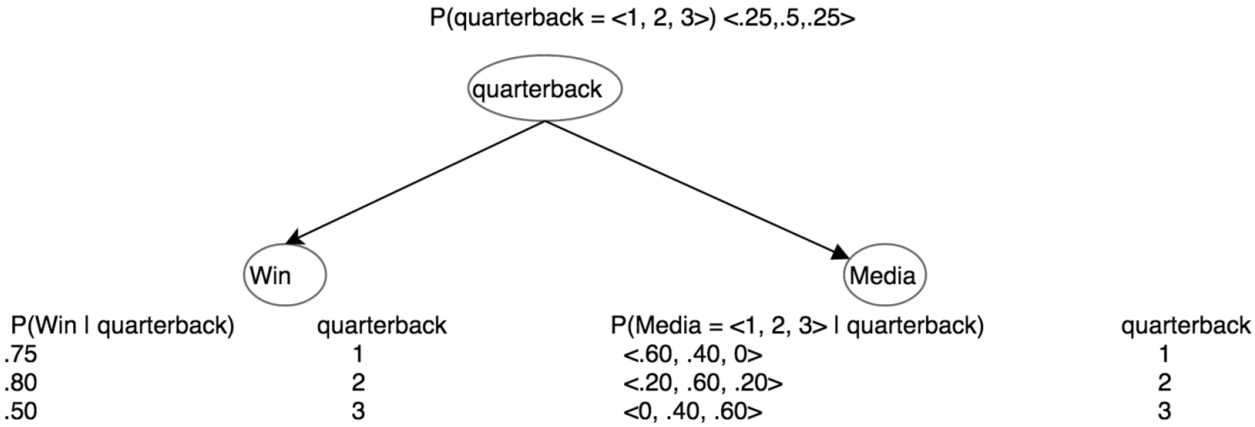
[x_weight, y_weight]

[3.2, 3.2]:	Classification Error Rate Mean: 0.231666666667	STD: 0.0412983723112
[5,5]:	Classification Error Rate Mean: 0.263333333333	STD: 0.0230337916018
[3.2, 5]:	Classification Error Rate Mean: 0.344166666667	STD: 0.0288795390853
[5, 3.2]:	Classification Error Rate Mean: 0.3675	STD: 0.0256715363346
[1, 1]:	Classification Error Rate Mean: 0.0825	STD: 0.0475
[1.5, 1.5]:	Classification Error Rate Mean: 0.0683333333333	STD: 0.0343187671366
[1, 1.5]	Classification Error Rate Mean: 0.160833333333	STD: 0.0974715058078

When the Gaussians are asymmetric, this results in a greater mean, i.e. more errors. This makes sense when we picture what is happening. When the data is symmetric then it can be visualized as a bunch a data points that, ultimately, are contained within an oval. When the data becomes asymmetric then that oval then becomes “squashed” and all distorted. Then when we

use the Euclidean distance metric one data point that is far away due to the distortion may be classified with a nearby group.

PROBLEM 2:



P(Outcome=win | MediaReport=3QB) = P(win, MediaReport = 3) / P(MediaReport=3QB)

P(MediaReport=3QB) = P(Media = 3 | quarterback)P(quarterback)

= P(Media = 3 | quarterback=2)P(quarterback=2) + (Media = 3 | quarterback=3)P(quarterback=3)

= (.20 .50) + (.60*.25)

= 0 .25

P(Win, MediaReport = 3) = P(Win, MediaReport = 3 | QB=1)P(QB=1) + P(Win, MediaReport = 3 | QB=2)P(QB=2) + P(Win, MediaReport = 3 | QB=3)P(QB=3)

=(0 * .25* .75) + (.20*.8*.5) + (.6*.5*.25) = .155

P(Outcome=win | MediaReport=3QB) =.155/.25 = **0 .62**