HW 1 REPORT

1)

1.1) At the start, the probability for choosing the door has car behind was 1/3 and we chose the first door. After we are said that there is a goat behind the door three, the probability having door behind for door one will be added on probability of door two because our preference was definite. So changing the door preference will increase our probability to find correct door from 1/3 to 2/3.

1.2) In the second example we have N doors and 1 of them includes goat behind. So probability of finding goat is 1/N. -1<k<n-1, as k becomes integer. We open 1 door so k = 1 in our case.
The probability to not getting in the first trial is N-1/N and getting in the second trial(after 1 of the doors opened) is 1/N-2. When we multiply them, we find (N-1)/N(N-2). That gives the probability that we find in the second trial so changing the door is better to find the goat because:

$$(N-1)/N(N-2) > 1/N$$

2)

2.1) In this part we don't have any prior information, MLE of the question becomes directly #heads/#trials which is 7/20. Let's prove this:

P(x="heads") = Q, Xi = k
$f(x) = Q^k * (1- Q)^{(20-k)}$

We have 20 different x_i. If i^{th} coin is head, x_i=1. If it is not, x_i=0.
$L_x (\theta) = \prod (Q^k * (1- Q)^{(20-k)})$
$K_x = \ln(L_x) = \ln(Q^{\Sigma k} * (1-Q)^{(20 - \Sigma k)})$

The summation of k's become number of heads which is 7.

$K_x = \ln(\theta^7 * (1- \theta)^{13}) = 7\ln \theta + 13 \ln(1-\theta)$

We should derivate the Kx to find the maximum of likelihood estimators:
Kx' = 7/ Q + 13 (-1/1- Q) = 0

7/ Q = 13/1- Q => 7 - 7 Q = 13 Q

Q =7/20

2.2)

The formula for MAP estimator is given as :

$$\Theta MAP = (k + \alpha - 1)/(n + \alpha + \beta - 2)$$

So our answer is:

$$P(\theta) = (N1 + 6)/(N1 + N2 + 18)$$

2.3) ΘMLE and ΘMAP has been calculated. In the question, if there was no experiment the MAP formula would be

$$\theta = (\alpha - 1)/(\alpha + \beta - 2)$$

Where alfa and beta are the parameters of Beta(alfa, beta) distribution. In order to find MLE, choose alfa as 8 and beta as 14 which makes our theta value 7/20.

3)

3.1) Eucledian distance metric will be used in this data because the distance between two nodes are needed to define if the drug is pathologically active. In the use of cosine distance metric, the angle between nodes is priority but we

don’t need the angle of nodes toward each other so euclidian is a good preference. Actually all Markov models work for this algorithm.

3.2) According to the table below, accuracy increases until a point and decreases after some value of k as expected. When number of k increases, number of neighbors included to the prediction increases too. Until a point that is an advantage for accuracy rate however after a point there are some irrelevant data(near but not from the true class of validated protein) in the neighbors which affects prediction in a bad way and decrease the accuracy rate. So the results are increases and decreases after some point as expected. Precision should also increase. It increases as my expectation without small decreases in the middle. Infinity represents there is no active protein.

	ACCURACY	**PRECISION**
k = 1	%78.5	%35.6
k = 3	%88.3525	%79.2
k = 5	%86.36	%80.1
k = 10	%83.2	%90
k = 20	%82	%96
k = 50	%79.9	∞
k = 100	%79.9	∞
k = 200	%79.9	∞

3.3) For time elapse, k shows the number of neighbors included to the classification calculation. Because the number of k is not increasing in a huge manner, the increase in the elapsed time is not sharp. There are just slight changes in the elapsed time as expected. Majority of the time spent for the euclidian distance calculations to know where our protein is in our space. The time required is a huge number like 15 minutes because kNN is not a feasible solution for this problem. There are lots of rows and columns to count which is a huge burden for computer memory. kNN is not a good solution for this. For theory calculation, we know the complexity of kNN algorithm is **O(N*D+k)**(N: Number of samples in training set, D: Dimension of the training set, k: Number of nearest neighbors). Because k is quite smaller than N our time complexity is **O(N)**.

	ELAPSED TIME IN SECONDS
k = 1	457.35078
k = 3	458.08
k = 5	458.70
k = 10	459.003
k = 20	459.8
k = 50	460.55
k = 100	465.122
k = 200	470.01

4)

4.1) In our example, we check for four different classes. For all these four classes, there will be P(Di) in the denominator which comes from Naive Bayes f formula. Because all the probabilities calculated for different classes to choose maximum of them for classifying new data includes P(Di), the denominators of them can be ignored. They are same.

4.2) We need number of unique words used in all emails because other remaining words will be no meaning to use. There are even some words which are used in no of the emails so they just make the job of coder harder. There are 4 classes so number of 4 x #unique words will be enough. We should also need to know the represented number of those words so 4 x #unique words + 4 parameters needed to implement this model.

4.3) Using MLE for this problem is not a good way to predict with huge percentage. There are more than 37.000 words which is a size of an English dictionary. It is impossible for our emails includes all those words inside. In MLE, we can not use Laplace so if a word did not used in an email, it will join probability as log0 which is minus infinity. We accept those as space text so our prediction will become a chance because majority of emails did not have at least one word from all dictionary. So accuracy rate is close to 1/4 because huge percentage of our predictions will be space. As seen in the below paragraph, accuracy is 23.7 which is 194 correct and 606 wrong predictions.

	Space (Prediction)	Medicine (Prediction)	Chriptology (Prediction)	Electronic (Prediction)
Space (Actual)	192	7	1	0

	Space (Prediction)	Medicine (Prediction)	Chriptology (Prediction)	Electronic (Prediction)
Medicine (Actual)	198	2	0	0
Chriptology (Actual)	191	9	0	0
Electronic (Actual)	195	3	2	0

4.4) In MAP case, we use Laplace(adding α (1 in our case) to nominator and a x number of words in dictionary) Method to get rid of zeros in the MLE example which causes minus infinity possibility(an empty email). Because we add all of the summation and we compare the possibilities nothing change in the prediction result if you use Laplace Method. As a result of this, accuracy increases. Accuracy is 92.25. 740 true and 60 wrong answers.

	Space (Prediction)	Medicine (Prediction)	Chriptology (Prediction)	Electronic (Prediction)
Space (Actual)	190	4	3	3
Medicine (Actual)	5	184	2	9
Chriptology (Actual)	1	3	186	10

	Space (Prediction)	Medicine (Prediction)	Chriptology (Prediction)	Electronic (Prediction)
Electronic (Actual)	2	5	13	180

4.5) The accuracy of MAP decreased to 75.75 if α value become 100. We observed this decrease because the increase in the α value cause worse prediction. If you think on the case it is infinity, because of the value log(infinity), all possibilities for new word will contribute same even if their repetition inside mail are different. So prediction would be same for all mails in this case which decrease accuracy rate quite sharp. We see the decrease of the accuracy as a start point of this way. We expect that accuracy will drop down with the increase in α value. Experiment rights our hypothesis because accuracy is 75.75.

4.6) When I print 20 most commonly used words, I saw that all these words are related with their email topics. Like “nasa" or “space” in space emails, normally. By the help of seeing those words the model and its success may be interpretable. After prediction, coder may print those words and check for whether the words are related with its mail topic. Printing those may help us to understand if the model successful in this way.

```
20 words common in Space——->  ['space', 'nasa', 'launch',
'earth', 'system', 'time', 'posting', 'orbit', 'moon',
'data', 'host', 'shuttle', 'people', 'net', 'spacecraft',
'satellite', 'solar', 'message', 'jpl', 'lunar']

20 words common in Medicine——->  ['medical', 'people',
'health', 'disease', 'patients', 'time', 'msg', 'food',
'cancer', 'hiv', 'well', 'years', 'treatment', 'posting',
'number', 'medicine', 'help', 'doctor', 'study', 'host']

20 words common in Chriptology——->  ['key', 'encryption',
'chip', 'government', 'clipper', 'people', 'system',
'keys', 'public', 'privacy', 'security', 'message', 'law',
'des', 'escrow', 'phone', 'time', 'nsa', 'data', 'secure']
```

```
20 words common in Electronic——->  ['power', 'ground',
'host', 'work', 'circuit', 'posting', 'time', 'wire',
'current', 'copy', 'message', 'radio', 'high', 'help',
'voltage', 'well', 'phone', 'problem', 'output', 'amp']
```

4.7) I used comparison logic to find highest and lowest probability. That means I upload the value of global lowest and highest values in each iteration of “for" as seen in the code. It could increase the time a little bit maybe but it was easier to find like this than finding and sorting all the probabilities. (The probabilities defined are the values whose denominators ignored.)

```
Lowest and highest probability of Space:
-18287.143166546903  and  -1.3862943611198906

Lowest and highest probability of Medicine:
-18369.886174019804  and  -1.3862943611198906

Lowest and highest probability of Chriptology:
-17555.82885415734  and  -1.3862943611198906

Lowest and highest probability of Electronic:
-18383.234407242093  and  -1.3862943611198906
```

4.8) In Bernoulli Naive Bayes Model, we have only two alternatives. We use those model for data including strict two options. In our mail data, we multiple alternatives because we need to know the repletion of a word in email. Knowing whether the word exist is not usable information for us to predict the class of email. If I used Bernoulli, probably I would check for if the word is exists and make prediction on this so accuracy would be very low. To do this I would use kNN as in question 3 to predict email classes.

CODES

Q3)

```
import os
import csv
import math
import random
import operator
```

```
Import time

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

trainFeatures = np.genfromtxt("question-3-train-features.csv", delimiter=",")
trainLabels = np.genfromtxt("question-3-train-labels.csv", delimiter=",")
validFeatures = np.genfromtxt("question-3-valid-features.csv", delimiter=",")
validLabels = np.genfromtxt("question-3-valid-labels.csv", delimiter=“,")

k = 3
trainF = np.array(trainFeatures)
validF = np.array(validFeatures)
trainL = np.array(trainLabels)
testLab = np.array(validLabels)
nn = 0
trueN = 0

def euclidean_distance(sample1, sample2):
    assert len(sample1) == len(sample2), "Different number of features exist for the given
samples"
    return np.power(np.subtract(sample1, sample2), 2).sum()

def get_neighbors(trainData, testArr, k):
    distances = []
    neighbors = []

    for index in range(len(trainData)):
        distances.append((euclidean_distance(trainData[index], testArr), index))
```

```
    distances.sort(key = operator.itemgetter(0))

    for i in range(k):
        neighbors.append(distances[i])
    return neighbors

def classify(distanceArray, k):
    class_Arr = []
    for index in range(k):
        class_Arr.append(trainL[distanceArray[index][1]])
    return max(set(class_Arr), key=class_Arr.count)

def calc_accuracy(gt_y, pred_y):
    correct = 0
    for g_y, p_y in zip(gt_y, pred_y):
        if g_y == p_y:
            correct += 1
            if(g_y == 1):
                global trueN
                trueN += 1

    return (correct/float(len(gt_y))*100)

seconds = time.time()
finArr = []
for arr in range(len(validF)):
    neighbors = get_neighbors(trainF, validF[arr], k)
    onesNum = classify(neighbors, k)
    if(onesNum == 1):
        global nn
```

```
        nn += 1
    finArr.append(onesNum)
print(calc_accuracy(finArr, testLab))
endSeconds = time.time()
print("Runtime: ", endSeconds - seconds)
print("Precision", (trueN/(nn))*100)
```

Q4)

```
import os
import csv
import math
import random
import operator
import numpy as np
import pandas as pd
```

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

testFeatures = pd.read_csv("question-4-test-features.csv", header = None)
testLabels = pd.read_csv("question-4-test-labels.csv", header = None)
trainFeatures = pd.read_csv("question-4-train-features.csv", header = None)
trainLabels = pd.read_csv("question-4-train-labels.csv", header = None)

trainWords = np.genfromtxt("question-4-vocab.txt", dtype = str)

noWords = 37358
lowestSpace = float('inf')
highestSpace = float('-inf')
lowestMedicine = float('inf')
highestMedicine = float('-inf')
lowestChrip = float('inf')
highestChrip = float('-inf')
lowestElec = float('inf')
highestElec = float('-inf')

def getVarNum():
    #number of words arrays
    spaceArr = np.zeros(noWords)
    medicineArr = np.zeros(noWords)
    chripArr = np.zeros(noWords)
    elecArr = np.zeros(noWords)
    testArr = testFeatures.to_numpy()
    testLab = testLabels.to_numpy()
    finArr = list()

    #number of essays in different classes
```

```
    countSpace = 0
    countElec = 0
    countMedicine = 0
    countChrip = 0

    for index, label in trainLabels.iterrows():
        if(label[0] == 1):
            countSpace += 1
            spaceArr = np.array([spaceArr, trainFeatures.loc[index]]).sum(axis=0)
        elif(label[0] == 0):
            countMedicine += 1
            medicineArr = np.array([medicineArr, trainFeatures.loc[index]]).sum(axis=0)
        elif(label[0] == 2):
            countChrip += 1
            chripArr = np.array([chripArr, trainFeatures.loc[index]]).sum(axis=0)
        elif(label[0] == 3):
            countElec += 1
            elecArr = np.array([elecArr, trainFeatures.loc[index]]).sum(axis=0)

    printMax20Words(spaceArr, medicineArr, chripArr, elecArr)

    for i in range(len(testArr)):
        finArr.append(calculateParametersMAP(spaceArr, countSpace, medicineArr,
countElec, chripArr, countMedicine, elecArr, countChrip, testArr[i]))

    print("Accuracy: " + str(calc_accuracy(finArr, testLab)))

    #for i in range(len(testArr)):
     #   finArr.append(calculateParametersMLE(spaceArr, countSpace, medicineArr,
countElec, chripArr, countMedicine, elecArr, countChrip, testArr[i]))
```

```
   # print("Accuracy: " + str(calc_accuracy(finArr, testLab)))

#part4.6 Calculation of max word
def printMax20Words(spaceArr, medicineArr, chripArr, elecArr):

    sortedSpace = np.argsort(spaceArr)
    sortedMedicine = np.argsort(medicineArr)
    sortedChrip = np.argsort(chripArr)
    sortedElec = np.argsort(elecArr)
    finSpace = list()
    finMedicine = list()
    finChrip = list()
    finElec = list()

    for index in range(noWords - 1, noWords - 21, -1):
        finSpace.append(trainWords[sortedSpace[index]])
        finMedicine.append(trainWords[sortedMedicine[index]])
        finChrip.append(trainWords[sortedChrip[index]])
        finElec.append(trainWords[sortedElec[index]])

    print("20 words common in Space----> ", finSpace, "\n")
    print("20 words common in Medicine----> ", finMedicine, "\n")
    print("20 words common in Chriptology----> ", finChrip, "\n")
    print("20 words common in Electronic----> ", finElec, "\n")



    def calculateParametersMLE(spaceArr, countSpace, medicineArr, countElec,
chripArr, countMedicine, elecArr, countChrip, testArr):
    #word counts in classes
    numSpace = np.array(spaceArr).sum()
```

```
numElec = np.array(elecArr).sum()
numMedicine = np.array(medicineArr).sum()
numChrip = np.array(chripArr).sum()

#infinity check
chSpace = False
chElec = False
chMedicine = False
chChrip = False

#essay numbers in classes
eSum = countSpace + countMedicine + countElec + countChrip
eSpace = countSpace/eSum
eElec = countElec/eSum
eMedicine = countMedicine/eSum
eChrip = countChrip/eSum

#non zero elements
nonSpace = np.count_nonzero(spaceArr)
nonMedicine = np.count_nonzero(medicineArr)
nonElec = np.count_nonzero(elecArr)
nonChrip = np.count_nonzero(chripArr)

#MLE estimators Space
probSpace = math.log(eSpace)
for i in range(noWords):
    if(spaceArr[i]/numSpace == 0 and testArr[i] == 0):
        continue
    else:
        if(spaceArr[i]/numSpace == 0):
            chSpace = True
```

```
            break
        logVal = np.log((spaceArr[i])/(numSpace))
        probSpace += (logVal*testArr[i])

#MLE estimators Medicine
probMedicine = math.log(eMedicine)
for i in range(noWords):
    if(medicineArr[i]/numMedicine == 0 and testArr[i] == 0):
        continue
    else:
        if(medicineArr[i]/numMedicine == 0):
            chMedicine = True
            break
        logVal = np.log((medicineArr[i])/(numMedicine))
        probMedicine += (logVal*testArr[i])



#MLE estimators Electronic
probElec = math.log(eElec)
for i in range(noWords):
    if(elecArr[i]/numElec == 0 and testArr[i] == 0):
        continue
    else:
        if(elecArr[i]/numElec == 0):
            chElec = True
            break
        logVal = np.log((elecArr[i])/(numElec))
        probElec += (logVal*testArr[i])

#MLE estimators Chriptology
probChrip = math.log(eChrip)
```

```
    for i in range(noWords):
        if(chripArr[i]/numChrip == 0 and testArr[i] == 0):
            continue
        else:
            if(chripArr[i]/numChrip == 0):
                chChrip = True
                break
            logVal = np.log((chripArr[i])/(numChrip))
            probChrip += (logVal*testArr[i])

    if(chSpace and chMedicine and chElec and chChrip):
        return 1

    if(probSpace>probMedicine and probSpace>probChrip and probSpace>probElec):
        return 0
    elif(probMedicine> probChrip and probMedicine>probElec and
probMedicine>probSpace):
        return 1
    elif(probChrip>probSpace and probChrip>probMedicine and probChrip>probElec):
        return 2
    elif(probElec>probSpace and probElec>probMedicine and probElec>probChrip):
        return 3



def calculateParametersMAP(spaceArr, countSpace, medicineArr, countElec, chripArr,
countMedicine, elecArr, countChrip, testArr):

    #lowest highest probability
    global lowestSpace
    global highestSpace
    global lowestMedicine
```

```
global highestMedicine
global lowestElec
global highestElec
global lowestChrip
global highestChrip

#word counts in classes
numSpace = np.array(spaceArr).sum()
numElec = np.array(elecArr).sum()
numMedicine = np.array(medicineArr).sum()
numChrip = np.array(chripArr).sum()

#essay numbers in classes
eSum = countSpace + countMedicine + countElec + countChrip
eSpace = countSpace/eSum
eElec = countElec/eSum
eMedicine = countMedicine/eSum
eChrip = countChrip/eSum

#non zero elements
nonSpace = np.count_nonzero(spaceArr)
nonMedicine = np.count_nonzero(medicineArr)
nonElec = np.count_nonzero(elecArr)
nonChrip = np.count_nonzero(chripArr)

#MAP estimators Space

probSpace = math.log(eSpace)
for i in range(noWords):
    logVal = math.log((spaceArr[i]+100)/(numSpace+(100*noWords)))
    probSpace += (logVal*testArr[i])
```

```
    if(lowestSpace > probSpace):
        lowestSpace = probSpace
    if(highestSpace < probSpace):
        highestSpace = probSpace

#MAP estimators Medicine
probMedicine = math.log(eMedicine)

for i in range(noWords):
    logVal = math.log((medicineArr[i]+100)/(numMedicine+(100*noWords)))
    probMedicine += (logVal*testArr[i])
    if(lowestMedicine > probMedicine):
        lowestMedicine = probMedicine
    if(highestMedicine < probMedicine):
        highestMedicine = probMedicine

#MAP estimators Electronic
probElec = math.log(eElec)

for i in range(noWords):
    logVal = math.log((elecArr[i]+100)/(numElec+(100*noWords)))
    probElec += (logVal*testArr[i])
    if(lowestElec > probElec):
        lowestElec = probElec
    if(highestElec < probElec):
        highestElec = probElec

#MAP estimators Chriptology
probChrip = math.log(eChrip)

for i in range(noWords):
```

```
        logVal = math.log((chripArr[i]+100)/(numChrip+(100*noWords)))
        probChrip += (logVal*testArr[i])
        if(lowestChrip > probChrip):
            lowestChrip = probChrip
        if(highestChrip < probChrip):
            highestChrip = probChrip

    if(probSpace>probMedicine and probSpace>probChrip and probSpace>probElec):
        return 1
    elif(probMedicine> probChrip and probMedicine>probElec and
probMedicine>probSpace):
        return 0
    elif(probChrip>probSpace and probChrip>probMedicine and probChrip>probElec):
        return 2
    elif(probElec>probSpace and probElec>probMedicine and probElec>probChrip):
        return 3

def calc_accuracy(gt_y, pred_y):
    correct = 0
    for g_y, p_y in zip(gt_y, pred_y):
        if g_y == p_y:
            correct += 1

    return (correct/float(len(gt_y))*100)



getVarNum()
print("Lowest and highest probability of Space: ", lowestSpace, " and ", highestSpace)
print("Lowest and highest probability of Medicine: ", lowestMedicine, " and ",
highestMedicine)
```

```
print("Lowest and highest probability of Chriptology: ", lowestChrip, " and ", highestChrip)
print("Lowest and highest probability of Electronic: ", lowestElec, " and ", highestElec)
```