CS 464 - Introduction to Machine Learning

Homework Assignment 1

Yusuf Dalva

21602867
Section 2

November 10, 2019

Yusuf Dalva
21602867

Homework 1

CS 464
November 10, 2019

Question 1 - The Monty Hall Problem

Question 1.1:

If we don't have any prior beliefs while selecting a door, the possibility that the car is behind a door is:

$$P(D_1) = \frac{1}{3} \tag{1}$$

Where D_1 denotes the event that the car is behind door 1.

After this step, given that we selected the first door the host can either open the door 2 or 3, depending on which door the car is. The host is going to open one of the doors if the prize is behind the selected door $P(O_2|D_1) = P(O_3|D_1) = \frac{1}{2}$. Where O_i denotes the host opens door i. Additionally $P(O_3|D_2) = 1$ and $P(O_3|D_3) = 0$
Applying the Bayes's Rule, the probability that the car is behind door 2 that host opens door 3 is:

$$P(D_2|O_3) = \frac{P(O_3|D_2)P(D_2)}{P(O_3|D_1)P(D_1) + P(O_3|D_2)P(D_2) + P(O_3|D_3)P(D_3)} \tag{2}$$

$$\frac{P(O_3|D_2)P(D_2)}{P(O_3|D_1)P(D_1) + P(O_3|D_2)P(D_2) + P(O_3|D_3)P(D_3)} = \frac{1 * \frac{1}{3}}{1/2 * 1/3 + 1 * 1/3 + 0} = \frac{2}{3} \tag{3}$$

The probability of winning by selecting door 2, given that door 3 opened is given above, with respect to that as $\frac{2}{3} > \frac{1}{3}$ we have higher winning possibility in the case we switch.

Question 1.2:

Assuming that there are N doors which the car is behind one of them, the probability of car being behind door 1 is $P(D_1) = \frac{1}{3}$

Applying the same logic here, the case where the contender switches to some other door (say door 2) and the host opens a door (say door 3), the probability of winning when we switch to door 2 is:

$$P(D_2|O_3) = \frac{P(O_3|D_2)P(D_2)}{\sum_{i=1}^{N} P(O_3|D_i)P(D_i)} \tag{4}$$

Evaluating this equation:

$$P(D_2|O_3) = \frac{\frac{1}{N-1} * \frac{1}{N}}{(N-2) * \frac{1}{N-2} * \frac{1}{N} + \frac{1}{N} * \frac{1}{N-1} + 0} = \frac{N-1}{N * (N-2)} \tag{5}$$

In this calculation it is assumed that we are selecting the goat for N-2 times at door 1, selecting the car at door 1 which gives the sum at the denominator as explained in equation 5.

As the number $\frac{N-1}{N*(N-2)} > \frac{1}{N}$, the competitor is better off switching.

Question 2 - MLE and MAP

Question 2.1:

As the ordering of the occurrence of the tosses is known after the trials, the event specified can only occur in one way. Allowing θ to be the probability of getting heads, the probability of the specified event (trials) can be illustrated in the way given below:

$$P(D|\theta) = \theta^h(1-\theta)^t \tag{6}$$

Where h is the number of heads and t is the number of tails obtained.
In order to get the MLE estimator for this expression, the derivative of the expression is taken and made equal to zero, in order to gain advantage in terms of computation the operation is going to be done on the logarithm values of the functions. The MLE estimator for θ is going to be the value of θ that makes the expression above 0:

$$\frac{d}{d\theta}ln(\theta^h(1-\theta)^t) = 0 \tag{7}$$

The steps of calculation is shown below

$$\frac{d}{d\theta}ln(\theta^h(1-\theta)^t) = h\frac{d}{d\theta}ln\theta + t\frac{d}{d\theta}ln(1-\theta) \tag{8}$$

$$= \frac{h}{\theta} - \frac{t}{1-\theta} = \frac{h-\theta h - t\theta}{\theta(1-\theta)} = 0 \tag{9}$$

Which implies that,

$$\theta_{MLE} = \frac{h}{h+t} \tag{10}$$

Where h is the number of heads obtained and t is the number of tails obtained. These values gives the result of 0.35 as a MLE estimate of θ.

Question 2.2:

To find the MAP estimate this time we need to maximize the value of $P(\theta|D)$. In order to do that the following equation is going to be solved and a θ value will be obtained:

$$\frac{d}{d\theta}P(\theta|D) = \frac{d}{d\theta}\frac{P(D|\theta)P(\theta)}{P(D)} \tag{11}$$

Where $P(D|\theta) = \theta^7(1-\theta)^{13}$, $P(\theta) = \frac{(N_1+N_2+1)!}{N_1!N_2!}\theta^{N_1}(1-\theta)^{N_2}$. Taking the derivative of the logarithm and assigning it to 0 just like in the previous question:

$$\frac{d}{d\theta}ln(P(\theta|D)) = \frac{d}{d\theta}\frac{(N_1+N_2+1)!}{N_1!N_2!} + (N_1+7)\frac{d}{d\theta}ln(\theta) + (N_2+13)\frac{d}{d\theta}ln(1-\theta) = 0 \tag{12}$$

Going one step further:

$$\frac{N_1+7}{\theta} - \frac{N_2+13}{1-\theta} = 0 \tag{13}$$

The solution of the equation given is:

$$\theta_{MAP} = \frac{N_1+7}{N_1+N_2+20} \tag{14}$$

Where 7 is the number of heads from the first experiment and 20 is the amount of trials done.

Question 2.3:

As it can be seen from the equation above the trials that create our prior belief is included in the estimator found. If no experiment was conducted before, the MAP estimate will be equal with the MLE estimate as shown in the equation below:

$$\theta_{MAP} = \frac{N_1+7}{N_1+N_2+20} = \frac{7}{20} = \theta_{MLE} \tag{15}$$

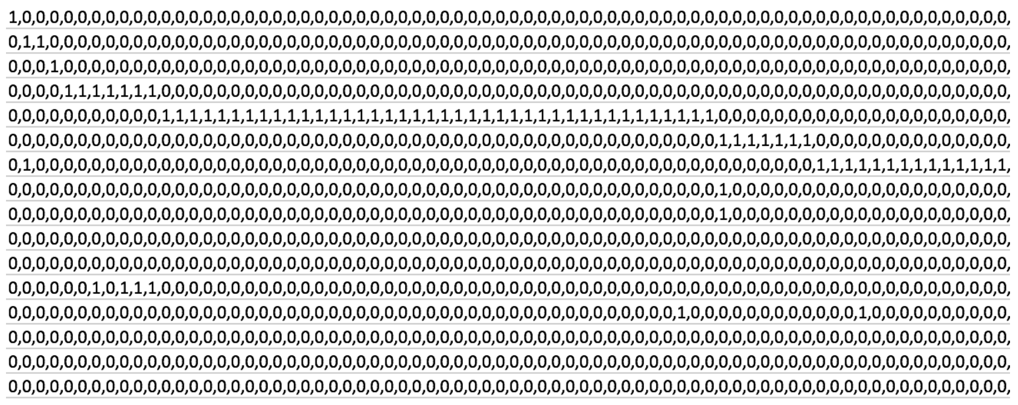
Since no experiment conducted a-priori, $N_1 = 0$ and $N_2 = 0$. This shows that $\theta_{MAP} = \theta_{MLE}$

Question 3 - DrugBank Drug target Identifiers Data Set

Question 3.1:

After the first look to the data set provided, the first thing done was identifying the values of the features in order to select a good distance measure. Just after a first look the it is identified that the data contains binary values which can be seen from the screenshot from the file "question-3-train-features.csv". The screenshot is provided below:

After identifying this fact, the second thing done was searching for different distance measures. In this regard, the Manhattan Distance measure and Euclidean Distance measures examined. By definition Euclidean distance looks for geometric distance and Manhattan Distance gives the distance by summing the differences between the data values for the same feature. They are expressed mathematically below:



```
1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,1,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
```

Figure 1: Sample values from "question-3-train-features.csv"

Euclidean Distance:

$$ED(x,y) = \sqrt{\sum_{i=1}^{V}(x_i - y_i)^2} \tag{16}$$

Manhattan Distance:

$$MD(x,y) = \sum_{i=1}^{V}|x_i - y_i| \tag{17}$$

V represents the number of features here.
As it may be seen from the equations, the Manhattan distance omits the square root operation and deals with absolute value rather than power function. In addition to that the data provided is binary, which means that taking the second power of the distance and taking the square root of that does not make any sense (which Manhattan Distance handles by summing the absolute values). Due to these factors, the Manhattan Distance is taken as a distance measure.
To be sure that the decision made is proper, both of the distance measures implemented and 10 experimental measures performed in order to be sure that the decision is optimal in terms of time elapsed in an operation.
The code used for this computation and the plot of the result is given below:

```python
# Euclidean Distance between two data points
def euclidean_distance(data_1, data_2):
    assert data_1.size == data_2.size
    distance = np.subtract(data_1, data_2)
    distance = np.sqrt(np.power(distance, 2))
    return np.sum(distance)
```

```
# Manhattan Distance between two data points
def manhattan_distance(row_1, row_2):
    assert row_1.size == row_2.size
    difference = np.subtract(row_1, row_2)
    difference = np.absolute(difference)
    return np.sum(difference)
```

After plotting the difference between the two distance measures, the resulting bar chart is given as below:

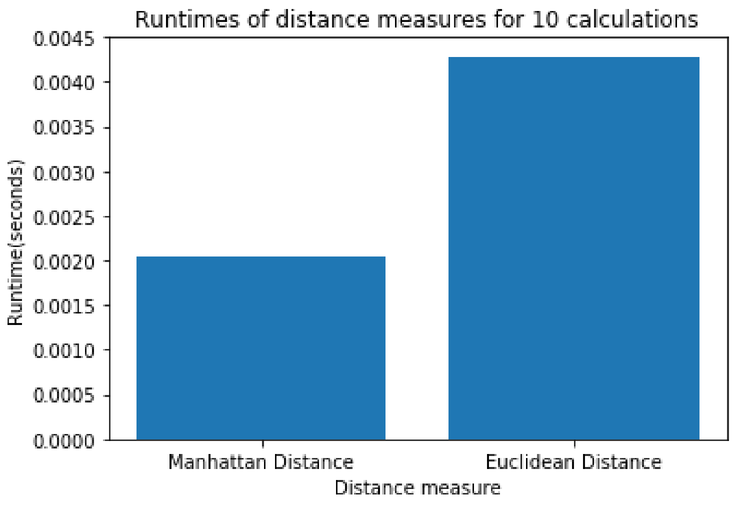


Figure 2: Difference of run times for 10 computing 10 differences between data points from training set

As a result, for the implementation of the algorithm the Manhattan distance is selected as a distance metric.

Since the kNN-classifier is going to calculate many different distances between different instances, using a slower algorithm will decrease the performance significantly. To prove this assumption the kNN algorithm was tested with Euclidean Distance as a distance measure but the run time for computing all distances necessary for classifier to be able to operate was approximately 7 hours, the output is given below, in terms of seconds:

```
# Compute all of the euclidean distances
print('Start!')
start = time.time()
compute_all_distances(valid_features, training_features)
dist_matrix
```

```
    end = time.time()
    print(end - start)
```

The output:

```
    Start!
    25322.214943170547
```

This version of the classifier will not be in the uploaded zip file both for clarity and since it does not include a final analysis.

Question 3.2:

As there is no parameters that are going to be calculated when training the data, the total runtime of the kNN classifier (labeling the new data included) is supplied in this section. The necessary data imports for running kNN algorithm, for easy data management the libraries pandas and numpy are being used in the code:

```
# Paths for the data files - This work assumes that the data
files are in the local directory with the file
# Can change the directories to access to the files
training_data_paths = ["question-3-train-protein-index.csv",
"question-3-train-features.csv", "question-3-train-labels.csv"]
validate_data_paths = ["question-3-valid-protein-index.csv",
"question-3-valid-features.csv", "question-3-valid-labels.csv"]
index_path = 'question-3-drug-index.csv'
training_labels = pd.read_csv(training_data_paths[2], header =None).to_numpy()
training_features = pd.read_csv(training_data_paths[1], header =None).to_numpy()
valid_labels = pd.read_csv(validate_data_paths[2], header = None).to_numpy()
valid_features = pd.read_csv(validate_data_paths[1], header = None).to_numpy()
```

In the code these instances are being used to get results from the methods provided. The methods used to computing the kNN-classified labels is provided below:

```
# Compute distances for one validation data
def compute_distances(valid_row, training_data, training_labels):
    distances = []
    for sample_idx in range(training_data.shape[0]):
        distances.append((manhattan_distance(valid_row,
        training_data[sample_idx]),
        training_labels[sample_idx][0]))
    return distances
```

By the method above the distance of a given validation point to the training data is being computed.

```
# Get the kth nearest neighbors of the sample data
def get_k_neighbors(training_data, training_labels, test_row, k):
    distances = compute_distances(test_row, training_data, training_labels)
    neighbors = distances = sorted(distances, key=operator.itemgetter(0))
    return neighbors[0:k]
```

Then by sorting the distance, label pairs on distance metric (located in index [0]), the k nearest neighbors returned. At this step, since there are many data points which has the same distance value with the validation data point, sorting algorithm used has importance. After several trials, the method *sorted*() is determined to be the algorithm giving the overall accuracy, which looks at the other keys of the tuples in the sorted list, in case of a tie. This operation is treated as the training step for kNN algorithm, which is going to be computed for every k value.

```
def evaluate_KNN_value(training_data, training_labels,
test_data_sample, k):
    k_neighbors = get_k_neighbors(training_features,
    training_labels, test_data_sample, k)
    positive_labels = 0
    negative_labels = 0
    for n in range(k):
        if k_neighbors[n][1] == 0:
            negative_labels += 1
        else:
            positive_labels += 1
    if positive_labels > negative_labels:
        return 1
    return 0
```

Then by looking at the evaluated distances for a validation data point, the the number of positive (1) and negative (0) values of the neighbors are being computed and the value having the maximum count in the neighbor set is returned as the predicted value.

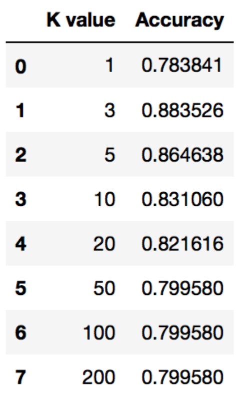
```
def evaluate_classifier(training_data, training_labels,
    test_data, test_labels, k):
    true_count = 0
    predicted_positive = 0
    predicted_negative = 0
    positives = [0,0] # first index is the number or false
    # positives, second is true posivites
    negatives = [0,0]
    # first index is the number of false negatives
    # second is true negatives
    for idx in range(test_data.shape[0]):
        val = evaluate_KNN_value(training_data,
        raining_labels,
        test_data[idx], k)
        if val == test_labels[idx]:
            true_count += 1
            if val == 1:
                positives[1] += 1
                predicted_positive += 1
            else:
                negatives[1] += 1
                predicted_negative += 1
           else:
            if val == 1:
                positives[0] += 1
                predicted_positive += 1
            else:
                negatives[0] += 1
                predicted_negative += 1
    result = ((true_count / (test_labels.size)),
    (predicted_positive, predicted_negative), (positives,
    negatives))
    return result
```

With the method above the kNN classifier for a particular k value, is evaluated by returning the accuracy, the numbers of positives, true positives, negatives and true negatives predicted. By using these values the precision and accuracy values can be calculated.

```
def evaluate_for_all_k(training_data, training_labels,
test_data, test_labels):
    k_set = (1,3,5,10,20,50,100,200)
    accuracy = []
    times = []
    precision = []
    results = []
    for i in range(len(k_set)):
        start = time.time()
        val = (evaluate_classifier(training_data,
        training_labels, test_data, test_labels, k_set[i]))
        end = time.time()
        times.append(end - start)
        if val[1][0] == 0:
            precision.append(0)
        else:
            precision.append(val[2][0][1] / val[1][0])
        accuracy.append(val[0])
    results.append(accuracy)
    results.append(precision)
    results.append(times)
    return results
```

The method given above computes and reports the runtimes and the evaluation values computed above and reports them for all of the k values desired. Even though there is a pattern observed after k = 50, the other values are also being computed just to be sure about the argument.

The accuracy values for the K-set (1, 3, 5, 10, 20, 50, 100, 200) is given below:



	K value	Accuracy
0	1	0.783841
1	3	0.883526
2	5	0.864638
3	10	0.831060
4	20	0.821616
5	50	0.799580
6	100	0.799580
7	200	0.799580

Figure 3: Accuracy values with changing K values for kNN classifier

For clarity, the values illustrated in Figure 3 is expressed in Figure 4 in order to observe the pattern.

As it can be seen from these figures, up until a point the accuracy of the kNN classifier increases (K increases from 1 to 3). But then the decrease in accuracy starts. With a data centered approach, it can be concluded that as K is more than a certain point, the label predicted becomes nearer to the average value of all data. Since in the dataset the amount of proteins having 0 value (inactive) are more, as the value of K increases the classifier is likely to predict 0 as a label. This can be observed in the precision values which is the ratio:

$$Precision = \frac{TP}{TP + FP} \tag{18}$$

Where TP denotes true positives and FP denotes false positives.
The precision values obtained from the kNN classifier is given below:

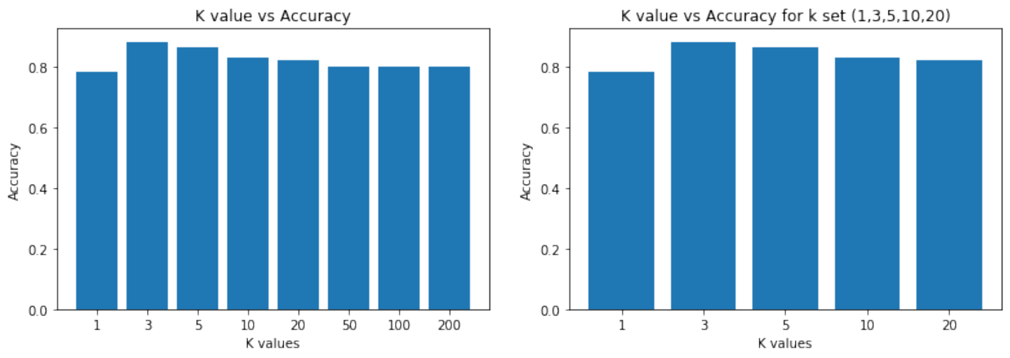
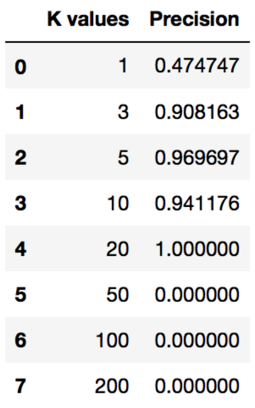


Figure 4: Bar charts showing Accuracy values with changing K values



	K values	Precision
0	1	0.474747
1	3	0.908163
2	5	0.969697
3	10	0.941176
4	20	1.000000
5	50	0.000000
6	100	0.000000
7	200	0.000000

Figure 5: Precision values with changing K values

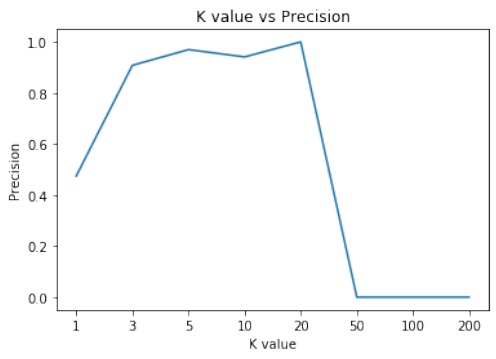
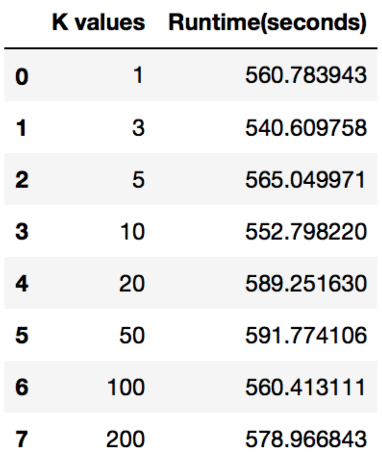


Figure 6: Plot showing Precision values with changing K values

As it can be observed here, the amount of predicted true values [1] reaches 0 as K increases, which also gives the conclusion that if the value of K increases a lot the value predicted would be closer to the label that occurs most. This is an error that causes drop in accuracy as seen in Figure 4. These results are expected, since only one neighbor can mislead if it an exceptional data point and the precision will be low again if the K value is way too large to be efficient. Because of the reasoning explained in above the results obtained was not surprising at all.

Question 3.3:

While running the kNN-classifier for different k values the run times are also recorded. The values obtained given below:



	K values	Runtime(seconds)
0	1	560.783943
1	3	540.609758
2	5	565.049971
3	10	552.798220
4	20	589.251630
5	50	591.774106
6	100	560.413111
7	200	578.966843

Figure 7: Plot showing Precision values with changing K values

The observed run times are really close to each other considering the changes in k values. It makes sense since the distances between the validation entries and training entries will be computed for every k value, since the data points that has the minimum distance will be needed. In order to retrieve these data points all of the distances must be computed and after sorting the (distance, label) pairs, k values that has minimum distance to the data point which is going to be predicted will be taken. So, the value of k does not make any difference in terms of overall computational complexity.
To show that there is no relation between k values and the runtime the plot of the k value versus runtime is given below:

Considering the runtime data obtained from the experiment done for $Question 3.2$, the running time of this algorithm is too much compared to the size of data. And because of the fact that it needs to traverse every feature in every instance multiple times, it cannot be considered as efficient in terms of asymptotic running times. The brute-force kNN algorithm traces all of the features of all of the training data for each validation set entry, the algorithm

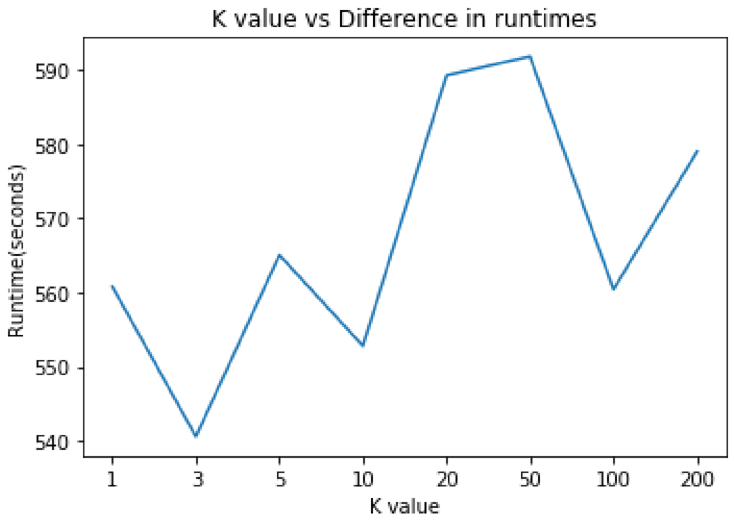


Figure 8: Plot showing Precision values with changing K values

traverses all of the data with three nested loop structures. If we assume all the amount of validation data set entries, training set entries and the features to be n (an arbitrarily large number) its asymptotic running time will be cubic due to its nested loop structure including 3 loops.
Shortly, the asymptotic runtime of kNN algorithm is:

$$\theta(n^3) \tag{19}$$

As the data set size grows (or dimensions of feature vectors), the running time of the algorithm will also grow, but in a cubic manner. So, it can be concluded that the kNN algorithm is not a feasible solution if the data set is large. It is going to take a lot of time to get a result for a given validation set.

Question 4 - Sentiment Analysis on Emails

Question 4.1:

In order to explain the answer for this question more clearly, the equation 4.3 is given below, which is the subject of this question:

$$P(Y = y_k|D_i) = \frac{P(Y = y_k)\prod_{j=1}^{V} P(X_j|Y = y)^t}{\sum_k P(Y = y_k)\prod_{j=1}^{V} P(X_j|Y = y_k)^t} \tag{20}$$

Where t is equal to the occurrence of word j in the email i. Considering equation (20), the denominator is simply the sum of the occurrence possibility of each document. In a certain classifier, the value given in eqn. (20) will be attempted to be maximized where the sum of the probabilities of the document being classified to one document type will be same for all of the values of y_k. To be more simple the nominator simply expresses the probability that the given document D_i is a member of the document class y_k whereas the denominator implies the sum of the probabilities that the document is a member of email class y_k, which has no effect on maximizing the value given in equation 5.

Question 4.2:

As it can be seen from the derivation made in the assignment sheet, the equations 4.4 and 4.5 implies that there are 4n + 3 parameters that are needed to be estimated. In order to explain the reasoning behind this expression, equation 4.4 from the assignment is given below:

$$P(Y = y_k|D_i) \propto P(Y = y_k) \prod_{j=1}^{V} P(X_j|Y = y_k)^t \tag{21}$$

Where t denotes the number of occurrences of word j-th word from the vocabulary in the document class D_i. Here, assuming that there are n words in the vocabulary, all of the possibilities of these words occurring in the document type k must be predicted. There are 4 document types, so the number of predictions for parameters in the form of $P(X_j|Y = y_k)$ is 4n. Also the values of $P(Y = y_k)$ should also be predicted. However, since the sum of the probability of the document being a member of a document type for all types is 1. Predicting $k - 1$ of this parameter should be enough (k is the number of document types). This fact is illustrated in the equation below:

$$\sum_{i=1}^{k} P(Y = y_i) = 1 \tag{22}$$

By doing the necessary calculations the amount of the parameters that are needed to be predicted is: $4 * 37358 + 3 = 149435$, where 37358 is the number of words in the vocabulary and 3 is Number of document classes - 1.

Question 4.3:

The necessary data imports, with information messages for Training a Naive Bayes classifier is given below:

```python
# Data imports
print('INFO: Starting data scan')
```

```
test_data_tables = ['question-4-test-features.csv', 'question-4-test-labels.csv']
train_data_tables = ['question-4-train-features.csv', 'question-4-train-labels.csv']
train_features = pd.read_csv(train_data_tables[0], header=None).to_numpy()
print('INFO: Training data features scanned')
train_labels = pd.read_csv(train_data_tables[1], header=None).to_numpy()
print('INFO: Training data labels scanned')
test_features = pd.read_csv(test_data_tables[0], header=None).to_numpy()
print('INFO: Test data features scanned')
test_labels = pd.read_csv(test_data_tables[1], header=None).to_numpy()
print('INFO: Test data labels scanned')
vocabulary = pd.read_csv('question-4-vocab.txt', sep = '\n', header = None)
print('INFO: Vocabulary scanned')
print('Data scan completed')
```

As an initial step, the training data is classified according to its labels, using the method provided below:

```
# Dividing documents into 4 subclasses
medicine_class = []
space_class = []
cryp_class = []
elec_class = []
def classify_documents(data_features, data_labels):
    for idx in range(data_labels.size):
        data = np.asarray(data_features[idx])
        if data_labels[idx][0] == 0:
            medicine_class.append(data)
        elif data_labels[idx][0] == 1:
            space_class.append(data)
        elif data_labels[idx][0] == 2:
            cryp_class.append(data)
        elif data_labels[idx][0] == 3:
            elec_class.append(data)
classify_documents(train_features, train_labels)
medicine_train = np.asarray(medicine_class)
space_train = np.asarray(space_class)
cryp_train = np.asarray(cryp_class)
elec_train = np.asarray(elec_class)
```

Which is just a simple classification that uses the labels attached to the training data provided.

For the next step the parameters involving the occurrence probabilities of the vocabulary words in a specific document type is being computed. These parameters are simply the $P(X_j|y = y_k)$ values in equation (23). These parameters are calculated according to the MLE estimator in the form given at the assignment:

$$\theta_j|y = y_k = \frac{T_(i, y = y_k)}{\sum_{j=1}^{V} T_(i, y = y_k)} \tag{23}$$

Initially the cumulative word counts are being computed with the given methods:

```
# Form series including the cumulative word count in a document
# (Documents) x (No of occurrences of each word)
def compute_word_count(train_set):
    word_count = np.zeros(train_set.shape[1])
    for idx in range(train_set.shape[0]):
        word_count = np.add(word_count, train_set[idx])
    return word_count
def compute_total_word_counts(medicine_set, space_set, cryp_set, elec_set):
    word_counts = []
    word_counts.append(compute_word_count(medicine_set))
    word_counts.append(compute_word_count(space_set))
    word_counts.append(compute_word_count(cryp_set))
    word_counts.append(compute_word_count(elec_set))
    return np.array(word_counts)
total_word_counts = compute_total_word_counts(medicine_train,
space_train, cryp_train, elec_train)
```

After that, the parameters related to the vocabulary words and the document types are being computed in the following way:

```
# Form matrix including learned parameters
def form_parameter_matrix(sets):
    dist_word_count = sets[0].size
    param_matrix = np.zeros([4, dist_word_count])
    sums = np.zeros(4)
    for i in range(4):
        word_count = np.sum(sets[i])
        for word in range(sets[i].size):
            param_matrix[i][word] = (sets[i][word]) /
            (word_count + sets[i].size)
    return param_matrix
MLE_params = form_parameter_matrix(total_word_counts)
```

After computing this step, the final step is being done, by calculating the parameters $P(Y = y_k)$ in the given in the assignment:

$$\pi_(y = y_k) = \frac{N_(y = y_k)}{N} \tag{24}$$

After computing these parameters, the prediction for the document type is being done for test set in the following method:

```
p_medicine = math.log(medicine_train.shape[0] / train_features.shape[0])
p_space = math.log(space_train.shape[0] / train_features.shape[0])
p_cryp = math.log(cryp_train.shape[0] / train_features.shape[0])
p_elec = math.log(elec_train.shape[0] / train_features.shape[0])
# MLE Predictor for a document
def predict_document(data_to_predict, variables):
    type_0 = p_medicine # Assuming it has type medicine
    type_1 = p_space # Assuming it has type space
    type_2 = p_cryp # Assuming it has type cryptology
    type_3 = p_elec # Assuming it has type space
    flags = [0,0,0,0]
    for i in range(data_to_predict.size):
        if flags[0] == 0:
            if (data_to_predict[i] == 0 and variables[0][i] == 0):
                type_0 += 0
            elif variables[0][i] == 0:
                flags[0] = 1
                type_0 = -1 * math.inf # smallest number in python as (-infinity)
            else:
                type_0 += (data_to_predict[i]) * math.log(variables[0][i])
        if flags[1] == 0:
            if (data_to_predict[i] == 0 and variables[1][i] == 0):
                type_1 += 0
            elif variables[1][i] == 0:
                flags[1] = 1
                type_1 = -1 * math.inf # smallest number in python as (-infinity)
            else:
                type_1 += (data_to_predict[i]) * math.log(variables[1][i])
        if flags[2] == 0:
            if (data_to_predict[i] == 0 and variables[2][i] == 0):
                type_2 += 0
            elif variables[2][i] == 0:
                flags[2] = 1
                type_2 = -1 * math.inf # smallest number in python as (-infinity)
            else:
                type_2 += (data_to_predict[i]) * math.log(variables[2][i])
        if flags[3] == 0:
            if (data_to_predict[i] == 0 and variables[3][i] == 0):
                type_3 += 0
            elif variables[3][i] == 0:
                flags[3] = 1
                type_3 = -1 * math.inf
            else:
                type_3 += (data_to_predict[i]) * math.log(variables[3][i])
    results = [] # Will store the results as tuples
    # (maxiimized_parameter, document code)
    results.append((type_0, 0))
    results.append((type_1, 1))
    results.append((type_2, 2))
    results.append((type_3, 3))
    results.sort(key=operator.itemgetter(0), reverse=True)
    max_value = results[0][0]
    if results[1][0] == max_value:
        return 1
    return results[0][1]
```

In order not to deal with floating point numbers which are above the computers limits, the logarithm of the value given in equation (23) is maximized. In the occurrences of 0*log0, the value is assumed as 0, but in the case of A*log0 the parameter value is assigned to the minimum number which is -∞. At this case it is intended that the document cannot be classified to the document type which the log0 value belongs to by assigning the value as the smallest number possible and stopping the computation. However, this case occurs frequently with the validation data and hence there is a tie occurring due to the $-\infty$ values as the decision values. In this case the tie situation occurs and the space class is being selected. This fact creates a significant decrease in the accuracy. With the data obtained, this fact can be proven easily.

With this approach for classification, the accuracy is found as 0.28, the confusion matrix for the classifier is given below:

The distribution among the confusion matrix also proves that the log0 values creates a situation that the tie occurs frequently, which can be considered as a bias. The use of infrequent words can be a choice of a writer, which can occur for any test set. So considering such a sharp approach can be falsifying, as seen in the accuracy value. Due to this bias observed by words resulting with log0 values in decision parameters, the use of the MLE estimate is a bad idea.

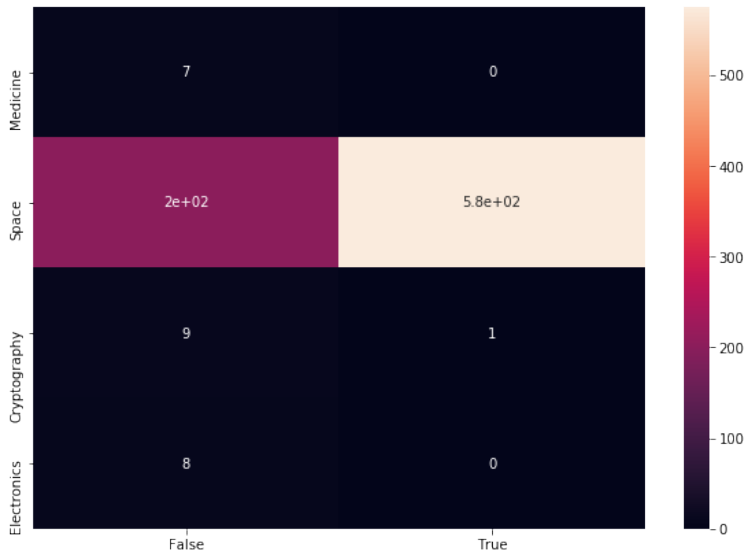


Figure 9: Confusion matrix for the MLE classifier.

Question 4.4:

In order to extend the classifier, a parameter named a is added to the method computes the estimations. The code is given below:

```python
# Form matrix including learned parameters
def form_parameter_matrix(sets, a):
    dist_word_count = sets[0].size
    param_matrix = np.zeros([4, dist_word_count])
    sums = np.zeros(4)
    for i in range(4):
        word_count = np.sum(sets[i])
        for word in range(sets[i].size):
            param_matrix[i][word] = (sets[i][word] + a) /
            (word_count + sets[i].size * a)
    return param_matrix
```

As a result of this additive smoothing the log0 values that result with the bias subjected in the discussion at Question 4.3 does not appear any more. The accuracy obtained by applying additive smoothing with $\alpha = 1$ (which is represented by a $= 1$), is found as 0.9235, which is significantly different than the value found with MLE estimator. The confusion matrix for MAP estimator is given below:
As the confusion matrix shows, now all data is being predicted without a bias, which can be shown with the distribution of the predictions for each class. Thus it can be concluded that the Dirichlet prior eliminates the bias in a significant way, as the distribution of the

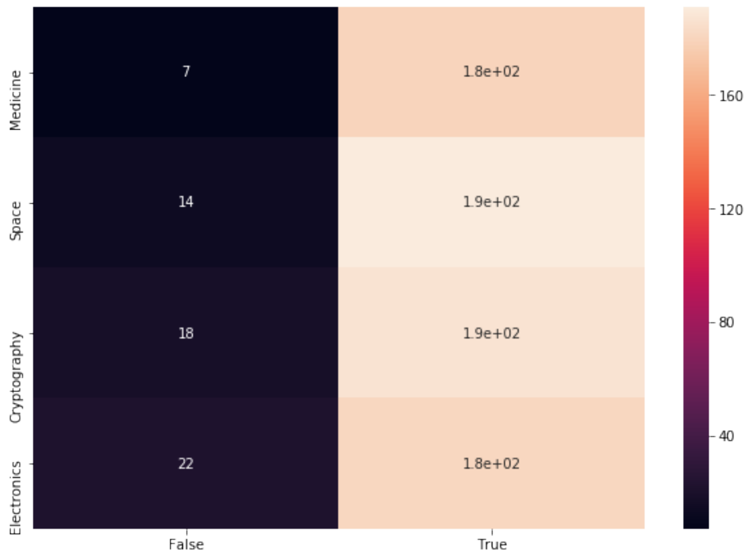


Figure 10: Confusion matrix for MAP estimator with Dirichlet prior $\alpha = 1$

predictions for the test data does not gather on one instance (such as space) because of the ties. The difference between the accuracy values obtained with different estimators can be seen in the following bar chart:

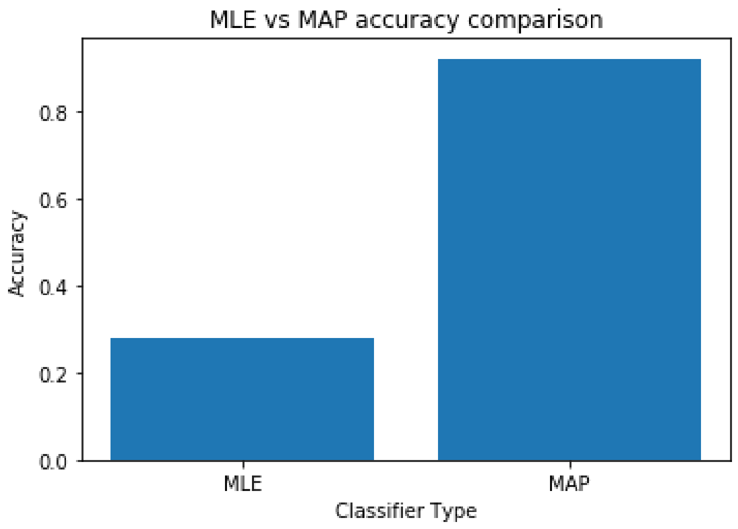


Figure 11: Accuracy values of MLE estimator and MAP estimator with Dirichlet Prior $\alpha = 1$

Question 4.5:

The value of the Dirichlet prior is crucial for the accuracy of the results. Even though with the value $\alpha = 1$ the results tends to increase significantly, it cannot be said as α approaches $\alpha = \infty$. The reason behind that is, as the value of the prior increases it is going to be more dominant against the value representing the occurrence of a particular vocabulary word and the estimated value will be just the ratio of two arbitrarily big numbers. In this case the estimator is expected to have a very low accuracy value. To illustrate the discussion the parameter value with a Dirichlet prior is given below:

$$\theta_(j|y = y_k) = \frac{T_(i, y = y_k) + \alpha}{\sum_{j=1}^{V} T_(i, y = y_k) + \alpha * V} \tag{25}$$

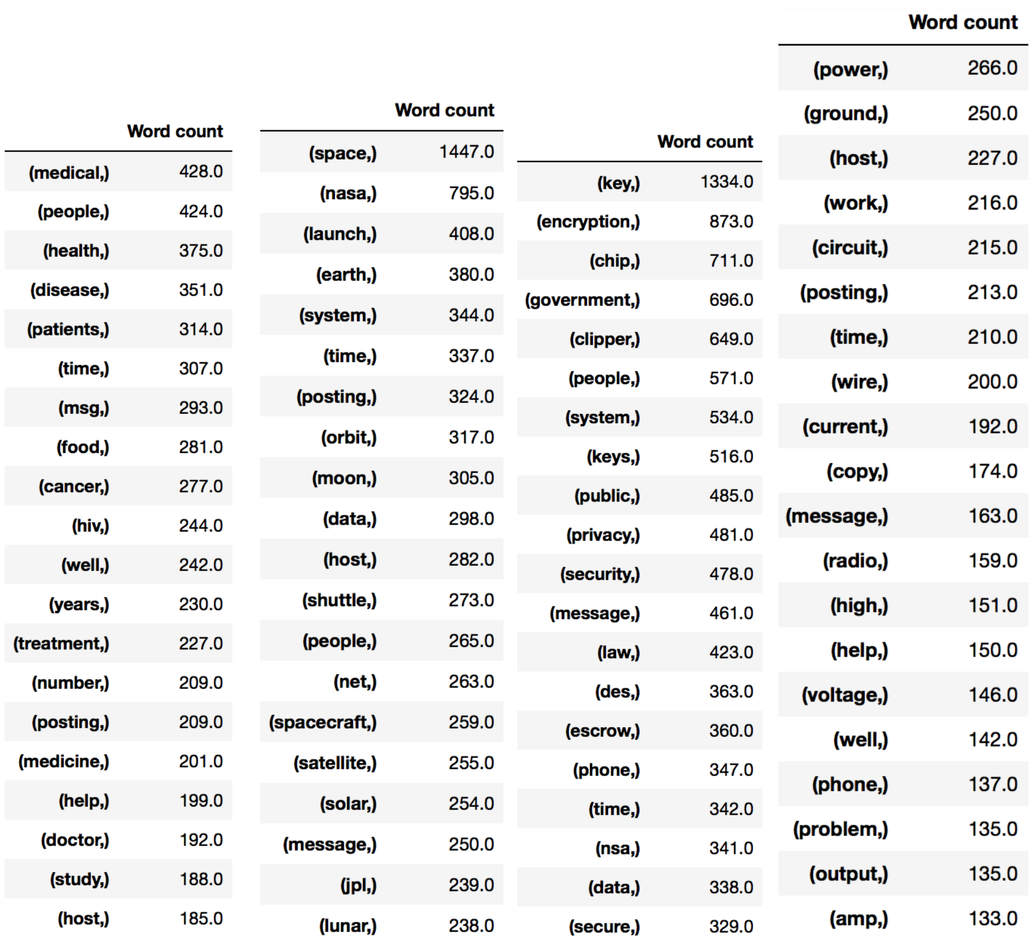
As it may be seen from the equation, with a large α value, the value of alpha will outlast the value of $T_(i, y = y_k)$ and the accuracy will drop drastically.

Question 4.6:

To find the most commonly used 20 words from the document set provided as the training set, the following methods are being used:

```python
# Form words versus occurances and give the max occured 20 words
def get_word_count_mapping(occurrance_matrix):
    words = []
    for doc_class in occurrance_matrix:
        words.append(pd.DataFrame(doc_class, index = vocabulary,
        columns = ['Word count']))
    return words
def get_most_frequent_20_words():
    d_type = [('word', str), ('count', int)]
    doc_word_count_matrix = get_word_count_mapping(total_word_counts)
    types = ('medicine', 'space', 'cryptography', 'electronics')
    tables = []
    for idx in range(len(doc_word_count_matrix)):
        tables.append(doc_word_count_matrix[idx].sort_values(by=
        'Word count'], ascending=False)[0:20])
    return tables
most_likely_word_tables = get_most_frequent_20_words()
```

Using the results obtained, the most common words found are given below:



	Word count
(medical,)	428.0
(people,)	424.0
(health,)	375.0
(disease,)	351.0
(patients,)	314.0
(time,)	307.0
(msg,)	293.0
(food,)	281.0
(cancer,)	277.0
(hiv,)	244.0
(well,)	242.0
(years,)	230.0
(treatment,)	227.0
(number,)	209.0
(posting,)	209.0
(medicine,)	201.0
(help,)	199.0
(doctor,)	192.0
(study,)	188.0
(host,)	185.0

	Word count
(space,)	1447.0
(nasa,)	795.0
(launch,)	408.0
(earth,)	380.0
(system,)	344.0
(time,)	337.0
(posting,)	324.0
(orbit,)	317.0
(moon,)	305.0
(data,)	298.0
(host,)	282.0
(shuttle,)	273.0
(people,)	265.0
(net,)	263.0
(spacecraft,)	259.0
(satellite,)	255.0
(solar,)	254.0
(message,)	250.0
(jpl,)	239.0
(lunar,)	238.0

	Word count
(key,)	1334.0
(encryption,)	873.0
(chip,)	711.0
(government,)	696.0
(clipper,)	649.0
(people,)	571.0
(system,)	534.0
(keys,)	516.0
(public,)	485.0
(privacy,)	481.0
(security,)	478.0
(message,)	461.0
(law,)	423.0
(des,)	363.0
(escrow,)	360.0
(phone,)	347.0
(time,)	342.0
(nsa,)	341.0
(data,)	338.0
(secure,)	329.0

	Word count
(power,)	266.0
(ground,)	250.0
(host,)	227.0
(work,)	216.0
(circuit,)	215.0
(posting,)	213.0
(time,)	210.0
(wire,)	200.0
(current,)	192.0
(copy,)	174.0
(message,)	163.0
(radio,)	159.0
(high,)	151.0
(help,)	150.0
(voltage,)	146.0
(well,)	142.0
(phone,)	137.0
(problem,)	135.0
(output,)	135.0
(amp,)	133.0

Figure 12: The 20 words mostly occurred in medicine, space, cryptography, electricity documents

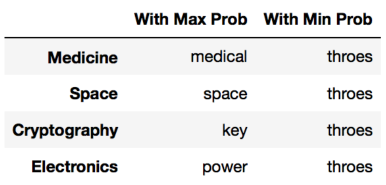
According to data obtained, the most common words occurred are medical for medicine documents, space for space documents, key for cryptography, power for electricity documents. These words can be easily linked with the document types but the word 'key' can be commented on its several different meanings, the same is valid for the word 'power'. When the document types are already known the interpretation easy, but for a different set of document types it may be harder. However, considering this example the interpretation of the links between the documents is easy and straightforward considering the domains medicine, space, cryptography and electricity.

Question 4.7:

As in the question definition the quantity which is subjected to the minimum and maximum probability is not specified clearly, this question is going to be answered in two ways. The first way concerns with the vocabulary words that has the minimum and the maximum probability of occurrence. The answer for this question is given below: Extending the code written for Question 4.6, the words that occur in the minimum and maximum amount is computed with the following method:

```python
# Word with highest probability and lowest probability
probability_matrix = form_parameter_matrix(total_word_counts, 0)
def get_words_occuring_min_and_max(prob_matrix):
    doc_types = ['Medicine', 'Space', 'Cryptography', 'Electronics']
    results = []
    for idx in range(prob_matrix.shape[0]):
        prob_words = pd.DataFrame(prob_matrix[idx], columns =
        ['Ocurrance Probability'], index = vocabulary)
        prob_words = prob_words.sort_values(by=['Ocurrance
        Probability'], ascending=False)
        results.append((prob_words.index.values[0][0],
        prob_words.index.values[prob_matrix.shape[1] - 1][0]))
    return pd.DataFrame(results, index = doc_types, columns =
    ['With Max Prob', 'With Min Prob'])
get_words_occuring_min_and_max(probability_matrix)
```

Using the result of this method, the following table constructed:



	With Max Prob	With Min Prob
Medicine	medical	throes
Space	space	throes
Cryptography	key	throes
Electronics	power	throes

Figure 13: Words with maximum and minimum occurrence in medicine, space, cryptography and electronics documents

When we consider the words that are used most frequently, it does make sense for the words that are used most frequently. However, the least used word which is ”throes”, it is hard to predict since it is not related with any of the document types.
For the second answer, the email instances are taken into consideration, the answer is given below:
To compute the parameters required(index of the mostly occurred and the least occurred mails for each email set), the following code snippet used:

```
probability_matrix = form_parameter_matrix(total_word_counts, 1)
def get_emails_with_highest_lowest_prob(prob_matrix, mail_set, mail_prob_set):
    doc_types = ['Medicine', 'Space', 'Cryptography', 'Electronics']
    results = []
    for idx in range(prob_matrix.shape[0]):
        prob_list = []
        for i in range(mail_set.shape[0]):
            email_prob_const =
            np.sum(np.multiply(np.log(prob_matrix[idx]),
            mail_set[i])) + mail_prob_set[idx]
            prob_list.append((email_prob_const, i, idx))
            prob_list = sorted(prob_list, key =
            operator.itemgetter(0))
        results.append(prob_list)
    return results
def get_max_min_mail_indices(result_matrix):
    min_max = []
    for idx in range(len(result_matrix)):
        min_max.append((result_matrix[idx][0][1],
        result_matrix[idx][len(result_matrix[idx]) - 1][1]))
    return min_max
mail_probs = [p_medicine, p_space, p_cryp, p_elec]
result = get_emails_with_highest_lowest_prob(probability_matrix,
test_features, mail_probs)
get_max_min_mail_indices(result)
```

When this code is evaluated, the logarithm of the probabilities of an email in the test set to be in a document class is obtained. Result is as follows (index of mail with minimum probability, index of mail with maximum probability):

```
[(622, 776), (622, 776), (622, 776), (622, 776)]
```

According to these results it is impossible to predict the email having the maximum and minimum probability of being classified in a certain document class. For the maximum case, the logarithm value shows the emails that does have 0 value in the sum (only including the foreign words to the training set). This is the maximum value that the sum can get since it is a particular sum of logarithms of probabilities. For the minimum case, for the mails contain only the words that are common in all of the email types even though they are not much in count, the minimum value from the sum is obtained, which is a sum that is arbitrarily small compared to all of the other mail instances. Supporting these arguments, the indexes obtained for all of the document types, having maximum and minimum classification probability, are same. Since these mails only include either the common words in the documents or the words which are not seen before they are extremely hard to predict in terms of classification.

Question 4.8:

According to the Bernoulli Document model, only the existence or absence of a word is going to be considered rather than the number of occurrences of the word in a document. So, in order to apply the Bernoulli Document Model rather than Multinomial Document Model the classifier will only look for the existence or absence of the word and calculate a probability accordingly. So the parameter that we are going to maximize will be:

$$P(X|Y = y_k) = \prod_{t=1}^{V}[X_t P(w_t|Y = y_k) + (1 - X_t)(1 - P(w_t|Y = y_k))] \tag{26}$$

According to this equation the code is just going to compute the consider the probability showing the ratio of the documents of a specific type that has the word to the documents having that type. The cumulative approach is going to change in this manner in the code.