Homework 4

Ayman Tawaalai

February 2024

1 Introduction

The purpose of the given assignment is to study and implement Naive Bayes through practical means. The assignment states that the student must one hot encode and ngram the patient sequence data from homework 1. Once that is completed, it will serve as the input features of x. Next, K means clustering must be done and it will serve as the truth labels, y. Once the x and y values are attained, multinomial and gaussian naive bayes will be implemented separately.

2 Naive Bayes

Naive Bayes is a supervised machine learning classification model that uses probability. It works by taking in probabilities for predicted outcomes and uses those probabilities to determine if the given inputs are what is the expected output. An example of this can be seen through spam email classification. In this scenario, emails are scanned and each word in the email will weigh the probability of it being real or spam. Whichever has the higher probability will be sorted into the respective category. There are two types of Naive Bayes. The email example most similarly fits with multinomial naive bayes because of the use of discrete variables. Gaussian Naive Bayes focus on continuous variables. It works by getting the probability dense function for each class and then uses it for classification purposes.

$$P(A|B) = \frac{P(B|A) * P(A)}{P(B)}$$

Figure 1: Naive Bayes Formula

```
In [103]:  datahot = datahot.astype(int)
           print(datahot.values)

           [[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]]
```

Figure 2: One Hot Conversion

```
In [137]:  n = 30
           def generatengrams(data, n):
               ngrams_list = [tuple(data[i:i+n]) for i in range(len(data)-n+1)]
               return ngrams_list

           ngrams = generatengrams(patientdata, n)

           ngramsdf = pd.DataFrame(ngrams, columns=[f'Ngram_{i+1}' for i in range(n)])

           print(ngramsdf)

                 Ngram_1  Ngram_2  Ngram_3  Ngram_4  Ngram_5  Ngram_6  Ngram_7  Ngram_8  \
           0         58.0     33.0     34.0     62.0     33.0     48.0     58.0     33.0
           1         33.0     34.0     62.0     33.0     48.0     58.0     33.0     34.0
           2         34.0     62.0     33.0     48.0     58.0     33.0     34.0     33.0
           3         62.0     33.0     48.0     58.0     33.0     34.0     33.0     62.0
           4         33.0     48.0     58.0     33.0     34.0     33.0     62.0     33.0
           ...        ...      ...      ...      ...      ...      ...      ...      ...
           29296     58.0     33.0     34.0     33.0     34.0     48.0     58.0     33.0
           29297     33.0     34.0     33.0     34.0     48.0     58.0     33.0     34.0
           29298     34.0     33.0     34.0     48.0     58.0     33.0     34.0     33.0
           29299     33.0     34.0     48.0     58.0     33.0     34.0     33.0     34.0
           29300     34.0     48.0     58.0     33.0     34.0     33.0     34.0     33.0

                 Ngram_9  Ngram_10  ...  Ngram_21  Ngram_22  Ngram_23  Ngram_24  \
           0         34.0      33.0  ...      33.0      62.0      48.0      33.0
           1         33.0      62.0  ...      62.0      48.0      33.0      58.0
           2         62.0      33.0  ...      48.0      33.0      58.0      33.0
           3         33.0      58.0  ...      33.0      58.0      33.0      34.0
           4         58.0      33.0  ...      58.0      33.0      34.0      33.0
           ...        ...       ...  ...       ...       ...       ...       ...
           29296     34.0      33.0  ...      34.0      48.0      58.0      33.0
           29297     33.0      34.0  ...      48.0      58.0      33.0      34.0
           29298     34.0      33.0  ...      58.0      33.0      34.0      33.0
           29299     33.0      34.0  ...      33.0      34.0      33.0      34.0
           29300     34.0      62.0  ...      34.0      33.0      34.0      48.0

                 Ngram_25  Ngram_26  Ngram_27  Ngram_28  Ngram_29  Ngram_30
           0         58.0      33.0      34.0      33.0      62.0      33.0
           1         33.0      34.0      33.0      62.0      33.0      48.0
           2         34.0      33.0      62.0      33.0      48.0      33.0
           3         33.0      62.0      33.0      48.0      33.0      58.0
```

Figure 3: NGram conversion of orginal patient data

3 Implementation

Since the working data is housed in the third column, we will load the files using pandas. Once all the strings are parsed to floats, dummies is used for one hot conversion. The next step will be to conduct ngram as shown in figure three. Since the x values have now been generated it is now time to get the y values. The one hot was sent into the k means function and the user would have to input how many clusters they would like and how many iterations the model should run for. After K means is completed the y values have been generated. With both x and y values, it is time to begin Naive Bayes.

The first implementation will be for multinomial. The main work horse is the fit function. The function will take in both x and y values, split into training sets and testing sets, and uses Laplace smoothing to calculate the class probabilities.

```
import numpy as np
labels = []
def kmeans(x,k,iters):
    centroids = x[np.random.choice(len(x), k, replace=False)]
    for z in range(iters):
        labels = np.argmin(np.linalg.norm(x[:, np.newaxis] - centroids, axis=2), axis=1)
        new_centroids = np.array([x[labels == i].mean(axis=0) for i in range(k)])
        if np.all(centroids == new_centroids):
            break

        centroids = new_centroids
    return labels
y = kmeans(datahot.values, 3, 1000)
print(y)
```

Figure 4: KMeans implementation

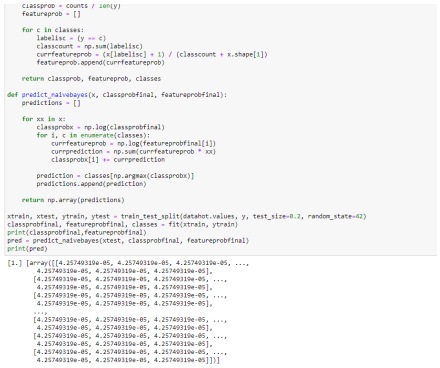


Figure 5: Multinomial Naive Bayes implementation

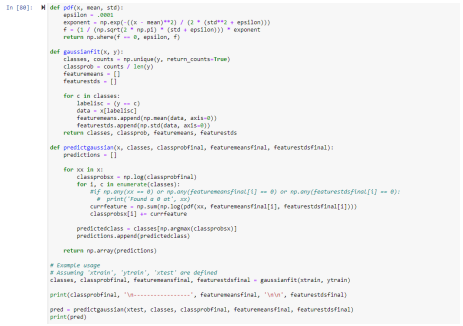


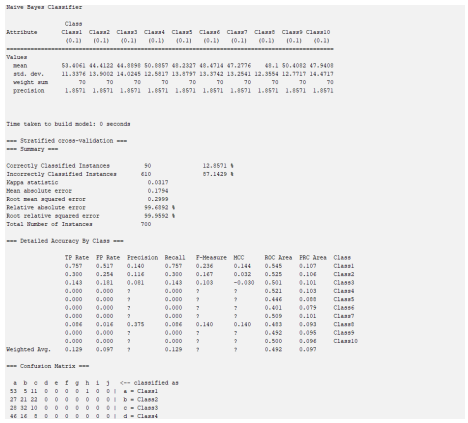
Figure 6: Gaussia Naive Bayes implementation

It places all the feature probabilities into an array and will return that array along with the class probability and classes itself. The predict function will take the final probabilities, along with the test data, and then predict the test labels for each test. Accuracy is then computed using SK learn.

The second implementation will be for gaussian. The same x and y values would be used, but a new fit function would have to be made. The gaussianfit function will return the feature means and standard deviation but is different from the fit function where it will return the feature probabilities. The predict function will take in the finals returned by the fit function and use the pdf function for calculation. The pdf function includes an epsilon to avoid divide by zero errors which were an issue. It will use that calculation to predict the class labels and append that to the predictions array, which is what is returned.

4 WEKA

To load the data into weka, some changes were made to the pd loading code. The new changes will now have the third column, along with the key value of



```
Naive Bayes Classifier

                Class
Attribute       Class1  Class2  Class3  Class4  Class5  Class6  Class7  Class8  Class9 Class10
                  (0.1)   (0.1)   (0.1)   (0.1)   (0.1)   (0.1)   (0.1)   (0.1)   (0.1)   (0.1)
================================================================================================
Values
  mean          53.4061 44.4122 44.8095 50.0857 48.2327 45.4714 47.2776    48.1 50.4082 47.9408
  std. dev.     11.3576 15.9002 14.0245 12.5817 13.9797 13.3742 13.2541 12.3554 12.7717 14.4717
  weight sum          70      70      70      70      70      70      70      70      70      70
  precision      1.8571  1.8571  1.8571  1.8571  1.8571  1.8571  1.8571  1.8571  1.8571  1.8571


Time taken to build model: 0 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances          90               12.8571 %
Incorrectly Classified Instances       610               87.1429 %
Kappa statistic                          0.0317
Mean absolute error                      0.1794
Root mean squared error                  0.2599
Relative absolute error                 99.6352 %
Root relative squared error             99.9592 %
Total Number of Instances              700

=== Detailed Accuracy By Class ===

               TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
               0.757    0.917    0.140      0.757    0.236      0.144    0.545     0.107     Class1
               0.300    0.254    0.116      0.300    0.167      0.032    0.525     0.106     Class2
               0.143    0.181    0.081      0.143    0.103      -0.030   0.501     0.101     Class3
               0.000    0.000    ?          0.000    ?          ?        0.521     0.103     Class4
               0.000    0.000    ?          0.000    ?          ?        0.446     0.088     Class5
               0.000    0.000    ?          0.000    ?          ?        0.401     0.079     Class6
               0.000    0.000    ?          0.000    ?          ?        0.509     0.101     Class7
               0.086    0.016    0.375      0.086    0.140      0.140    0.483     0.093     Class8
               0.000    0.000    ?          0.000    ?          ?        0.492     0.096     Class9
               0.000    0.000    ?          0.000    ?          ?        0.500     0.096     Class10
Weighted Avg.  0.129    0.097    ?          0.129    ?          ?        0.492     0.097

=== Confusion Matrix ===

  a  b  c  d  e  f  g  h  i  j   <-- classified as
 53  5 11  0  0  0  0  1  0  0 |  a = Class1
 27 21 22  0  0  0  0  0  0  0 |  b = Class2
 28 32 10  0  0  0  0  0  0  0 |  c = Class3
 46 16  6  0  0  0  0  0  0  0 |  d = Class4
```

Figure 7: Weka Summary

the classes, be written into a csv file. That csv file will be loaded for naive bayes in weka. The results show that only 13% of instances were classified correctly. Furthermore, the summary provided by weka suggests that Naive Bayes does not work best for the data set.

5 Conclusion

The purpose of this assignment was to implement both version of Naive Bayes in python to better understand how the model works from a white box perspective. The python implementation had both multinomial and Gaussian to accomplish this. The low accuracy was further reflected in Weka where it showed a low accuracy there as well in the summary. In conclusion, the given data set does not bode well with Naive Bayes tries to accomplish.