Assignment - 2

Emotional Analysis and Binary Classification Machine Learning Tasks

Machine learning _CSCI_6364_82
Assignment 2

Bhavya Sree Gudiseva
8th December 2023

Emotion Analysis using SVM and K-Means Clustering

1. Objective

Develop an understanding of emotion recognition in text using Support Vector Machines (SVM) for classification and K-Means clustering for pattern discovery. This assignment will help you grasp the nuances of supervised and unsupervised learning techniques in Natural Language Processing (NLP).

2. Data Preparation

2.1 Load and familiarize yourself with the EmotionLines dataset.

Upon inspecting the dataset has 3 Json files: test, train, and holdout json files.
After loading the train json file the data frame has 720 records with 24 columns, test json file the data frame has 200 records with 24 columns and holdout data frame has 80 records with 24 columns

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 720 entries, 0 to 719
Data columns (total 24 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   0       720 non-null    object
 1   1       720 non-null    object
 2   2       720 non-null    object
 3   3       720 non-null    object
 4   4       720 non-null    object
 5   5       688 non-null    object
 6   6       655 non-null    object
 7   7       620 non-null    object
 8   8       579 non-null    object
 9   9       547 non-null    object
 10  10      520 non-null    object
 11  11      484 non-null    object
 12  12      444 non-null    object
 13  13      408 non-null    object
 14  14      367 non-null    object
 15  15      333 non-null    object
 16  16      292 non-null    object
 17  17      253 non-null    object
 18  18      224 non-null    object
 19  19      189 non-null    object
 20  20      142 non-null    object
 21  21      102 non-null    object
 22  22      79 non-null     object
 23  23      35 non-null     object
dtypes: object(24)
memory usage: 135.1+ KB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 24 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   0       200 non-null    object
 1   1       200 non-null    object
 2   2       200 non-null    object
 3   3       200 non-null    object
 4   4       200 non-null    object
 5   5       188 non-null    object
 6   6       175 non-null    object
 7   7       162 non-null    object
 8   8       155 non-null    object
 9   9       140 non-null    object
 10  10      132 non-null    object
 11  11      126 non-null    object
 12  12      111 non-null    object
 13  13      102 non-null    object
 14  14      93 non-null     object
 15  15      83 non-null     object
 16  16      72 non-null     object
 17  17      63 non-null     object
 18  18      53 non-null     object
 19  19      40 non-null     object
 20  20      30 non-null     object
 21  21      21 non-null     object
 22  22      13 non-null     object
 23  23      5 non-null      object
dtypes: object(24)
memory usage: 37.6+ KB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 80 entries, 0 to 79
Data columns (total 24 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   0       80 non-null     object
 1   1       80 non-null     object
 2   2       80 non-null     object
 3   3       80 non-null     object
 4   4       80 non-null     object
 5   5       79 non-null     object
 6   6       75 non-null     object
 7   7       74 non-null     object
 8   8       67 non-null     object
 9   9       63 non-null     object
 10  10      61 non-null     object
 11  11      55 non-null     object
 12  12      50 non-null     object
 13  13      43 non-null     object
 14  14      40 non-null     object
 15  15      35 non-null     object
 16  16      30 non-null     object
 17  17      27 non-null     object
 18  18      23 non-null     object
 19  19      21 non-null     object
 20  20      17 non-null     object
 21  21      11 non-null     object
 22  22      6 non-null      object
 23  23      1 non-null      object
dtypes: object(24)
memory usage: 15.1+ KB
```

2.2 Conduct text preprocessing: tokenize, stem/lemmatize, and remove stop words.

2.2.1 **Tokenization :**

Performed tokenization to combine all the 24 columns of a row to one record

```
Modified Training DataFrame:
                                         tokenized_0  \
0  [also, I, was, the, point, person, on, my, com...
1                                    [Hey, ,, Mon, .]
2                                           [Hey, !]
3  [Good, job, Joe, !, Well, done, !, Top, notch, !]
4  [Okay, ,, look, ,, I, think, we, have, to, tel...

                                         tokenized_1  \
0         [You, must  ve, had, your, hands, full, .]
1  [Hey-hey-hey, ., You, wan, na, hear, something...
2                                            [Hi, !]
3     [You, liked, it, ?, You, really, liked, it, ?]
4  [What, ?, !, What, is, with, everybody, ?, It  ...

                                         tokenized_2  \
0                  [That, I, did, ., That, I, did, .]
1                                    [Do, I, ever, .]
2                      [What, are, you, doing, here, ?]
3                              [Oh-ho-ho, ,, yeah, !]
4  [Yes, ,, and, it, is, my, dying, wish, to, hav...

                                         tokenized_3  \
0  [So, let  s, talk, a, little, bit, about, your,...
1  [Chris, says, they  re, closing, down, the, bar...
2  [Ah, y'know, ,, this, building, is, on, my, pa...
3                          [Which, part, exactly, ?]
```

2.2.2 **Removing Stop words**

```
DataFrame with Stop Words Removed:
                                         tokenized_0  \
0  [also, point, person, company  s, transition, k...
1                                        [hey, mon]
2                                             [hey]
3           [good, job, joe, well, done, top, notch]
4  [okay, look, think, tell, rachel, messed, dess...

                                      tokenized_1             tokenized_2  \
0                           [must  ve, hand, full]                      []
1   [hey-hey-hey, wan, na, hear, something, suck]                  [ever]
2                                            [hi]                      []
3                           [liked, really, liked]        [oh-ho-ho, yeah]
4  [everybody, it  s, thanksgiving, ..., truth-day]  [yes, dying, wish, ring]

                                         tokenized_3  \
0                  [let  s, talk, little, bit, duty]
1                [chris, say, they  re, closing, bar]
2          [ah, y'know, building, paper, route, ...]
3                                    [part, exactly]
4  [see, i  m, buried, ring, spirit, going, wander...

                     tokenized_4  \
0                  [duty, right]
1                          [way]
2                           [oh]
3             [whole, thing, go]
4  [okay, that  s, enough, honey]
```

2.2.3 Lemmatization

```
DataFrame after Lemmatization:
                                               tokenized_0  \
0  [also, I, point, person, company  s, transition...
1                                       [Hey, ,, Mon, .]
2                                              [Hey, !]
3  [Good, job, Joe, !, Well, done, !, Top, notch, !]
4  [Okay, ,, look, ,, I, think, tell, Rachel, mes...

                                               tokenized_1  \
0                          [You, must  ve, hand, full, .]
1  [Hey-hey-hey, ., You, wan, na, hear, something...
2                                               [Hi, !]
3             [You, liked, ?, You, really, liked, ?]
4  [What, ?, !, What, everybody, ?, It  s, Thanksg...

                          tokenized_2  \
0        [That, I, ., That, I, .]
1               [Do, I, ever, .]
2                     [What, ?]
3          [Oh-ho-ho, ,, yeah, !]
4  [Yes, ,, dying, wish, ring, .]

                                               tokenized_3  \
0            [So, let  s, talk, little, bit, duty, .]
1             [Chris, say, they  re, closing, bar, .]
2   [Ah, y'know, ,, building, paper, route, I, ...]
```

2.2.4 Remove Punctuation and Lowercasing

```
DataFrame after Removing Punctuation and Lowercasing:
                                           tokenized_0  \
0  [also, i, point, person, company  s, transition...
1                                         [hey, mon]
2                                              [hey]
3           [good, job, joe, well, done, top, notch]
4  [okay, look, i, think, tell, rachel, messed, d...

                                           tokenized_1  \
0                          [you, must  ve, hand, full]
1  [hey-hey-hey, you, wan, na, hear, something, s...
2                                               [hi]
3                  [you, liked, you, really, liked]
4  [what, what, everybody, it  s, thanksgiving, .....

                tokenized_2  \
0       [that, i, that, i]
1            [do, i, ever]
2                   [what]
3        [oh-ho-ho, yeah]
4  [yes, dying, wish, ring]

                                           tokenized_3  \
0               [so, let  s, talk, little, bit, duty]
1                [chris, say, they  re, closing, bar]
2       [ah, y'know, building, paper, route, i, ...]
3                             [which, part, exactly]
4  [see, i  m, buried, ring, spirit, going, wander...

                     tokenized_4  \
0          [my, duty, all, right]
1                     [no, way]
2                          [oh]
3   [the, whole, thing, can, go]
4  [okay, that  s, enough, honey]
```

2.2.5 Applying Preprocessing on Test Data

```
Processed Test DataFrame:
0    why you  re coffee mug number bottom oh that  s ...
1    come lydia push push 'em push 'em harder harde...
2    okay ross n't say elevator uhh yes i n't okay ...
3    ohh what it kicked i think baby kicked oh god ...
4    previously friends i don  t know exactly it  s-i...
Name: combined_text, dtype: object
```

2.3 Transform text into numerical representations using TF-IDF.

```
  (4, 1858)	0.04418112639614794
  (4, 1805)	0.10502334436649442
  (4, 1762)	0.15236624459181713
  (4, 1645)	0.4301894337471002
  (4, 1559)	0.12806256050208997
  (4, 1538)	0.10034872939017696
  (4, 1513)	0.1246524134799041
  (4, 1509)	0.10375887641236281
  (4, 1355)	0.11909279382597292
  (4, 1286)	0.046828364485545046
  (4, 1193)	0.15236624459181713
  (4, 865)	0.12169840976459692
  (4, 604)	0.15236624459181713
  (4, 473)	0.12806256050208997
  (4, 389)	0.14339647791570007
  (4, 227)	0.0798687028034271
  (4, 197)	0.09103057493737671
  (4, 140)	0.07248491691320962

Vocabulary from TF-IDF Vectorizer:
{'also': 178, 'point': 3370, 'person': 3269, 'company': 938, 'transition': 4569, 'kl': 2404, 'gr': 1893, 'system': 4353, 'must': 2915, 'v
e': 4728, 'hand': 1973, 'full': 1777, 'let': 2516, 'talk': 4365, 'little': 2571, 'bit': 471, 'duty': 1369, 'right': 3681, 'you': 5007, 'l
l': 2575, 'heading': 2020, 'whole': 4880, 'division': 1267, 'lot': 2613, 'see': 3852, 'there': 4444, 'perhaps': 3262, '30': 33, 'people':
3253, 'dump': 1364, 'certain': 760, 'amount': 193, 'good': 1871, 'know': 2419, 'go': 1858, 'detail': 1206, 'don': 1286, 'beg': 418, 'we':
4823, 'definite': 1168, 'answer': 217, 'monday': 2855, 'think': 4454, 'say': 3796, 'confidence': 963, 'fit': 1648, 'well': 4842, 'reall
y': 3573, 'absolutely': 80, 'relax': 3610, 'great': 1914, 'waitress': 4785, 'went': 4844, 'last': 2461, 'month': 2865, 'forget': 1710, 'n
o': 2992, 'talking': 4369, 'actually': 101, 'ok': 3087, 'gon': 1869, 'na': 2922, 'get': 1830, 'room': 3715, 'night': 2979, 'later': 2465,
'yeah': 4983, 'sure': 4312, 'hey': 2055, 'mon': 2852, 'wan': 4795, 'hear': 2024, 'something': 4072, 'suck': 4271, 'ever': 1493, 'chris':
836, 'they': 4447, 're': 3554, 'closing': 893, 'bar': 369, 'way': 4821, 'apparently': 239, 'turning': 4619, 'kinda': 2390, 'coffee': 909,
'place': 3336, 'hang': 1983, 'got': 1887, 'beer': 417, 'pick': 3309, 'roommate': 3716, 'betcha': 448, 'italian': 2278, 'guy': 1950, 'um':
4645, 'mm': 2840, 'oh': 3078, 'god': 1862, 'poor': 3385, 'monica': 2859, 'wrote': 4973, 'poem': 3367, 'look': 2593, 'my': 2920, 'vessel':
4737, 'empty': 1436, 'nothing': 3022, 'inside': 2225, 'touched': 4542, 'seem': 3856, 'emptier': 1435, 'still': 4209, 'vase': 4726, 'mean
t': 2744, 'totally': 4540, 'seemed': 3857, 'happy': 1996, 'done': 1291, 'hi': 2058, 'ah': 140, 'building': 619, 'paper': 3192, 'route': 3
724, 'how': 2137, 'woman': 4927, 'interviewed': 2249, 'pretty': 3435, 'tough': 4544, 'thank': 4430, 'mark': 2691, 'coached': 900, 'starte
d': 4176, 'fall': 1560, 'line': 2554, 'wouldn': 4958, 'shut': 3954, 'proud': 3472, 'listen': 2564, 'sorry': 4087, 'crazy': 1055, 'jealou
s': 2302, 'it': 2277, 'like': 2544, 'ameri': 189, 'can': 687, 'ccan': 746, 'everybody': 1495, 'job': 2321, 'joe': 2323, 'top': 4532, 'not
ch': 3019, 'liked': 2545, 'ho': 2084, 'part': 3202, 'exactly': 1509, 'thing': 4452, 'give': 1844, 'specific': 4117, 'love': 2621, 'best':
445, 'scene': 3805, 'kangaroo': 2359, 'did': 1219, 'surprised': 4317, 'world': 4949, 'war': 4800, 'epic': 1460, 'fell': 1600, 'asleep': 2
82, 'didn': 1220, 'take': 4362, 'suggestion': 4281, 'that': 4435, 'coming': 934, 'buddy': 612, 'mad': 2650, 'call': 672, 'ambulance': 18
```

3. SVM for Emotion Classification:

3.1 Construct an SVM classifier to categorize emotions in text data.

```
Extracted Emotion Labels for Training Data (First 10 Rows):
0        neutral
1        neutral
2    non-neutral
3            joy
4        neutral
5        neutral
6          anger
7    non-neutral
8            joy
9          anger
dtype: object

Number of Rows with no Emotion Label in Training Data: 0

Shape of TF-IDF Matrix for Training Data: (720, 955)
Length of y_train: 720

Extracted Emotion Labels for Test Data (First 10 Rows):
0       surprise
1        neutral
2        neutral
3       surprise
4        neutral
5    non-neutral
6    non-neutral
7    non-neutral
8    non-neutral
9        neutral
dtype: object

Number of Rows with no Emotion Label in Test Data: 0

Shape of TF-IDF Matrix for Test Data: (200, 955)
Length of y_test: 200
```

3.2 SMOTE Analysis for resampling

```
Original training data shape (features): (720, 5030)
Original training data shape (labels): (720,)
Resampled training data shape (features): (2648, 5030)
Resampled training data shape (labels): (2648,)
Sample of resampled labels: ['neutral' 'neutral' 'non-neutral' 'joy' 'neutral' 'neutral' 'anger'
 'non-neutral' 'joy' 'anger' 'joy' 'anger' 'neutral' 'surprise' 'surprise']
Label counts before SMOTE:
neutral        331
joy            128
non-neutral    107
surprise        82
sadness         26
anger           23
disgust         14
fear             9
Name: count, dtype: int64
Label counts after SMOTE:
neutral        331
non-neutral    331
joy            331
anger          331
surprise       331
sadness        331
disgust        331
fear           331
Name: count, dtype: int64
```

3.3 Cross- Validation and Classification Metrics

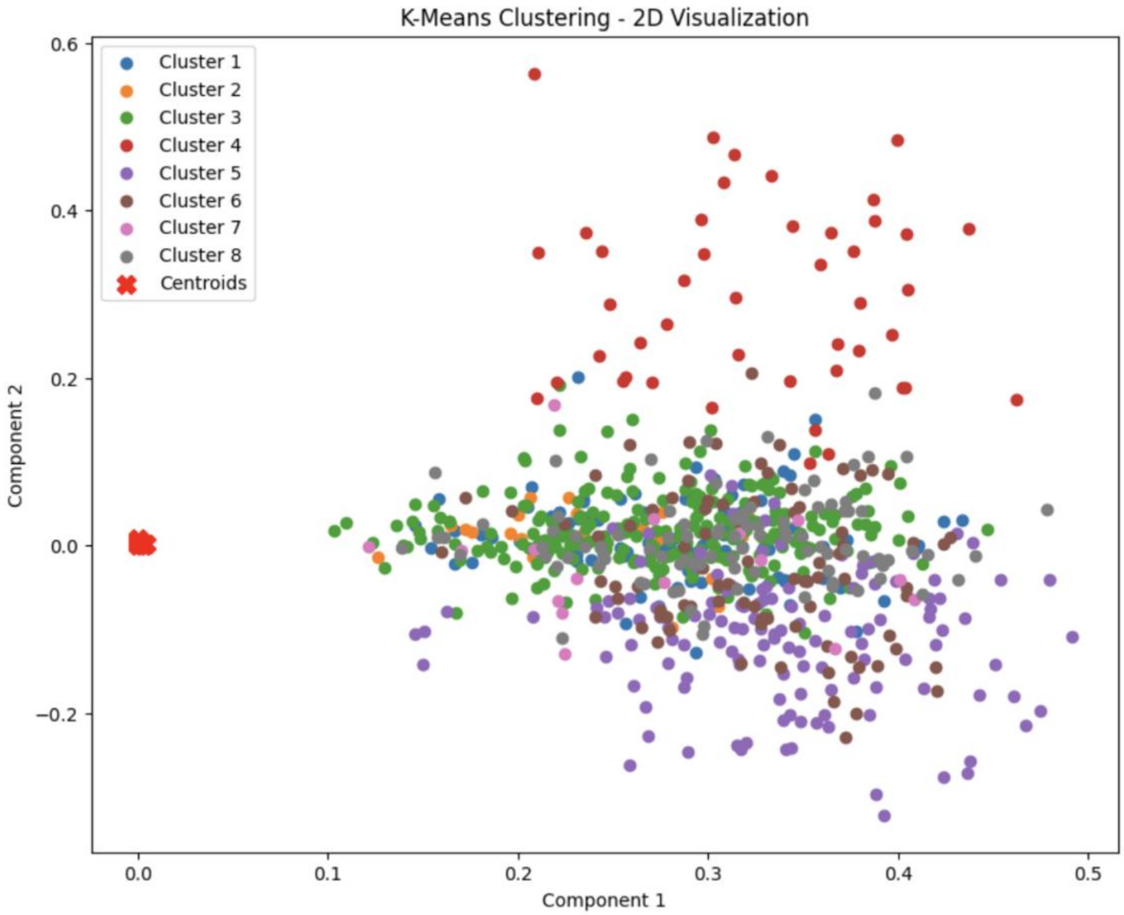
```
Cross-Validation Scores: [0.46527778 0.45833333 0.45833333 0.45833333 0.45833333]
Mean CV Score: 0.4597222222222225
Standard Deviation of CV Scores: 0.0027777777777777905
SVM Test Accuracy: 0.535
```

3.4 Optimize the classifier by experimenting with different kernels and hyperparameters.

```
['neutral' 'neutral' 'neutral' 'neutral' 'neutral' 'neutral' 'neutral'
 'neutral' 'neutral' 'neutral' 'neutral' 'neutral' 'neutral' 'neutral'
 'neutral' 'neutral' 'neutral' 'neutral' 'neutral' 'neutral' 'neutral'
 'neutral' 'neutral' 'neutral' 'neutral' 'neutral' 'neutral' 'neutral'
 'neutral' 'neutral' 'neutral' 'neutral' 'neutral' 'neutral' 'neutral'
 'neutral' 'neutral' 'neutral' 'neutral' 'neutral' 'neutral' 'neutral'
 'neutral' 'neutral' 'neutral' 'neutral' 'neutral' 'neutral' 'neutral'
 'neutral' 'neutral' 'neutral' 'neutral' 'neutral' 'neutral' 'neutral'
 'neutral' 'neutral' 'non-neutral' 'neutral' 'neutral' 'neutral' 'neutral'
 'neutral' 'neutral' 'neutral' 'neutral' 'neutral' 'neutral' 'neutral'
 'neutral' 'neutral' 'neutral' 'neutral' 'neutral' 'neutral' 'neutral'
 'neutral' 'neutral' 'neutral' 'neutral' 'neutral' 'neutral' 'neutral'
 'neutral' 'neutral' 'neutral' 'neutral' 'neutral' 'neutral' 'neutral'
 'neutral' 'neutral' 'neutral' 'neutral' 'neutral' 'neutral' 'neutral'
 'neutral' 'neutral' 'neutral' 'neutral' 'neutral' 'neutral' 'neutral'
 'neutral' 'neutral' 'neutral' 'neutral' 'neutral' 'neutral' 'neutral'
 'neutral' 'neutral' 'neutral' 'neutral' 'neutral' 'neutral' 'neutral'
 'neutral' 'neutral' 'neutral' 'neutral' 'neutral' 'neutral' 'neutral'
 'neutral' 'neutral' 'neutral' 'neutral' 'neutral' 'neutral' 'neutral'
 'neutral' 'neutral' 'neutral' 'neutral' 'neutral' 'neutral' 'neutral'
 'neutral' 'neutral' 'neutral' 'neutral' 'neutral' 'neutral' 'neutral'
 'neutral' 'neutral' 'neutral' 'neutral' 'neutral' 'neutral' 'neutral'
 'neutral' 'neutral' 'neutral' 'neutral' 'neutral' 'neutral' 'neutral'
 'neutral' 'neutral' 'neutral' 'neutral' 'neutral' 'neutral' 'neutral'
 'neutral' 'neutral' 'neutral' 'neutral' 'neutral' 'neutral' 'neutral'
 'neutral' 'neutral' 'neutral' 'neutral' 'neutral' 'neutral' 'neutral'
 'neutral' 'neutral' 'neutral' 'neutral' 'neutral' 'neutral' 'neutral'
 'neutral' 'neutral' 'neutral' 'neutral' 'neutral' 'neutral' 'neutral'
 'surprise' 'neutral' 'neutral' 'neutral']
Best Classifier Classification Report:
              precision    recall  f1-score   support

       anger       1.00      0.00      0.00         9
     disgust       1.00      0.00      0.00         1
        fear       1.00      0.00      0.00         2
         joy       1.00      0.00      0.00        24
     neutral       0.54      0.99      0.70       107
 non-neutral       1.00      0.03      0.06        35
     sadness       1.00      0.00      0.00         4
    surprise       0.00      0.00      1.00        18

    accuracy                           0.54       200
   macro avg       0.82      0.13      0.22       200
weighted avg       0.66      0.54      0.47       200

Accuracy Score:
0.535
```

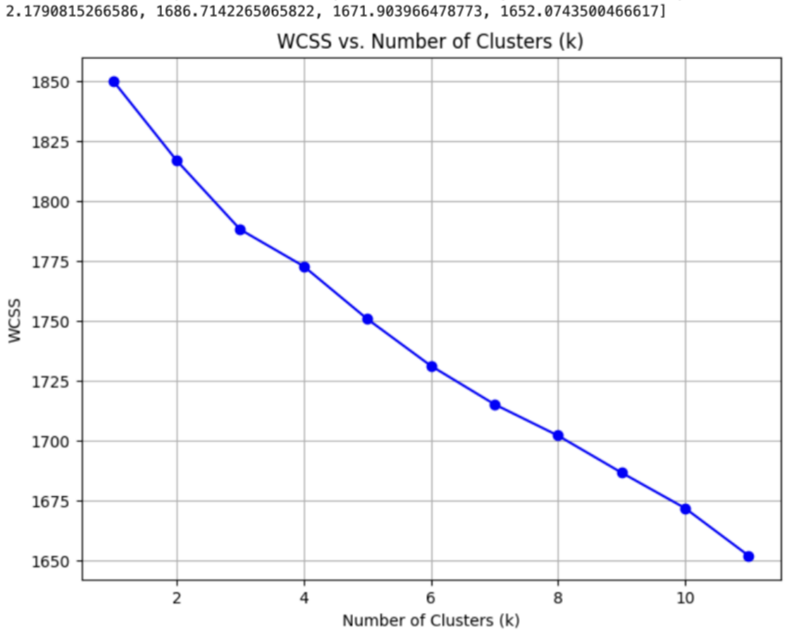
4. K-Means Clustering:

4.1 Implement K-Means clustering on the preprocessed text data.



4.2 Identify the optimal number of clusters with methods like the elbow technique.

```
[1850.1387164878392, 1816.89630603637, 1788.128444058858, 1772.7854955034268, 1750.79250594419, 1731.2801384990673, 1715.2392109162706, 170
2.1790815266586, 1686.7142265065822, 1671.903966478773, 1652.0743500466617]
```



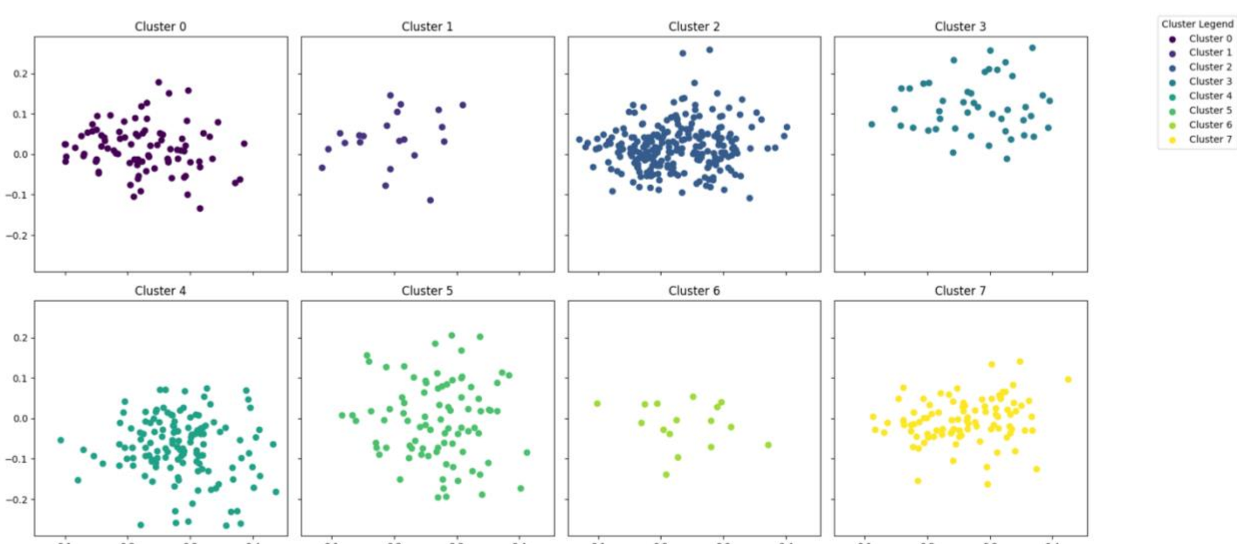
5. Model Insights:

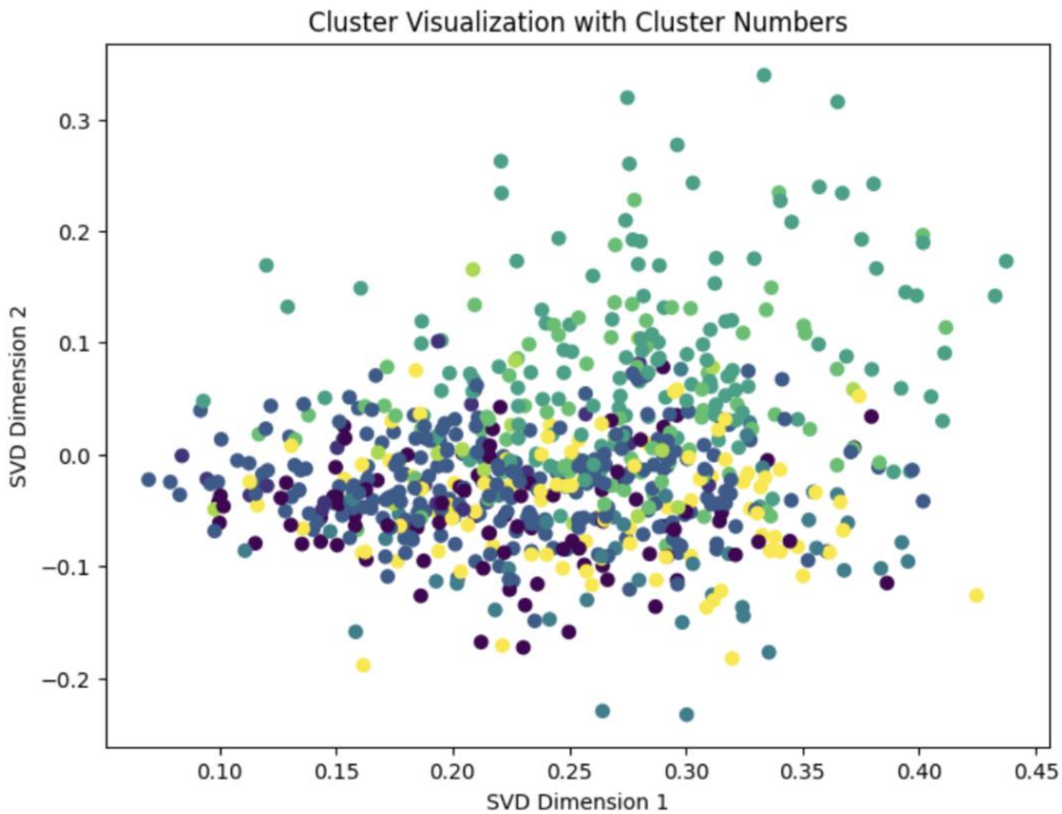
5.1 Analyze the performance of the SVM classifier and the clusters formed by K-Means.

```
              precision    recall  f1-score   support

       anger       1.00      0.00      0.00         9
     disgust       1.00      0.00      0.00         1
        fear       1.00      0.00      0.00         2
         joy       1.00      0.00      0.00        24
     neutral       0.54      0.99      0.70       107
 non-neutral       1.00      0.03      0.06        35
     sadness       1.00      0.00      0.00         4
    surprise       0.00      0.00      1.00        18

    accuracy                           0.54       200
   macro avg       0.82      0.13      0.22       200
weighted avg       0.66      0.54      0.47       200

Accuracy Score:
0.535
```

```
Cluster 0: Emotion - joy
Cluster 1: Emotion - non-neutral
Cluster 2: Emotion - neutral
Cluster 3: Emotion - non-neutral
Cluster 4: Emotion - non-neutral
Cluster 5: Emotion - neutral
Cluster 6: Emotion - surprise
Cluster 7: Emotion - non-neutral
```

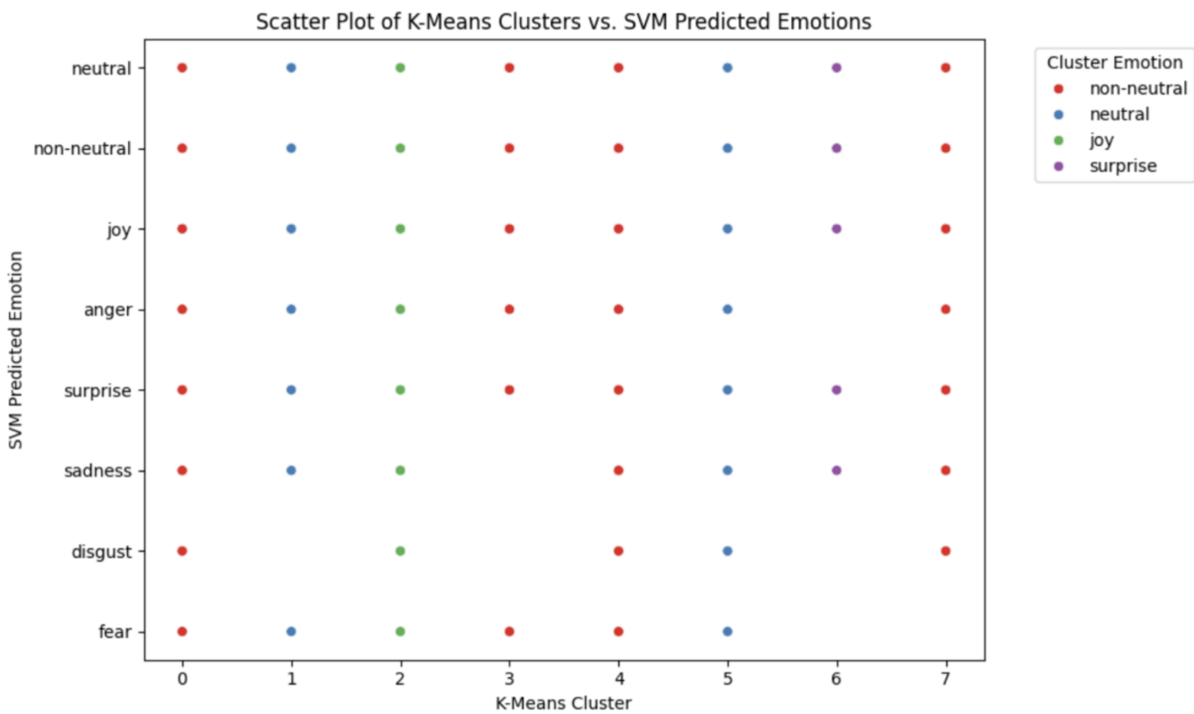
5.2 Compare and contrast the results obtained from both SVM and K-Means.

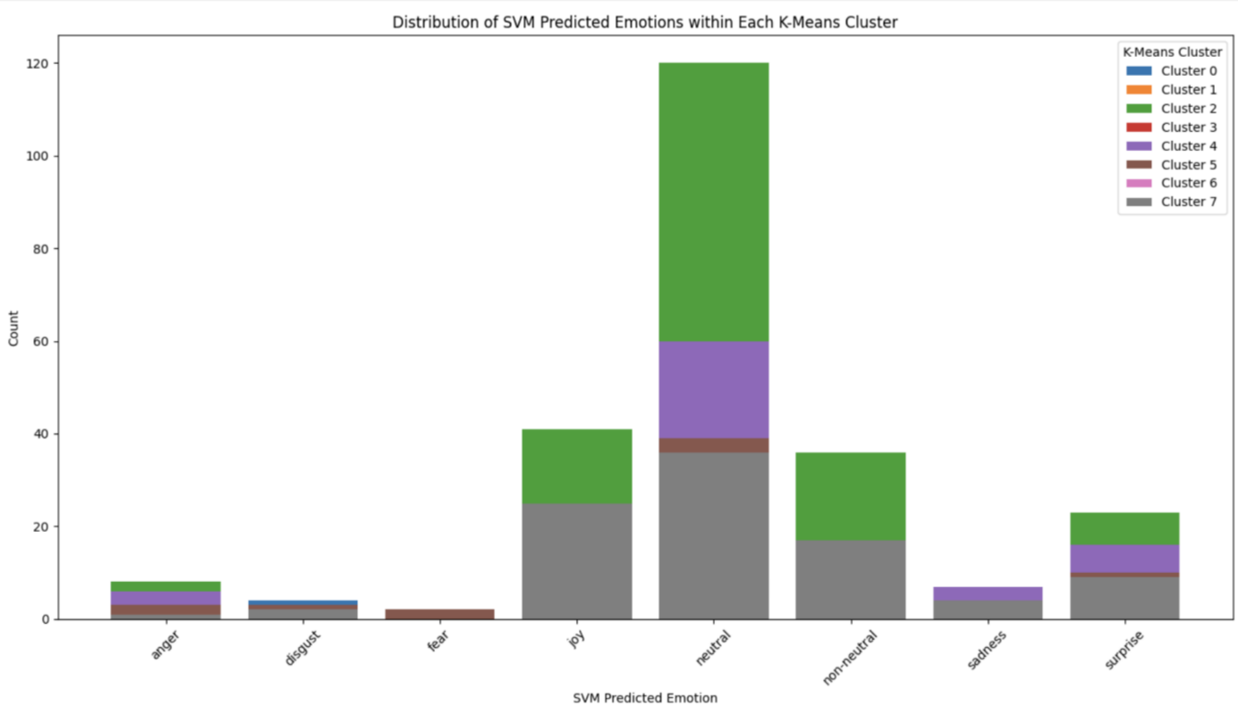


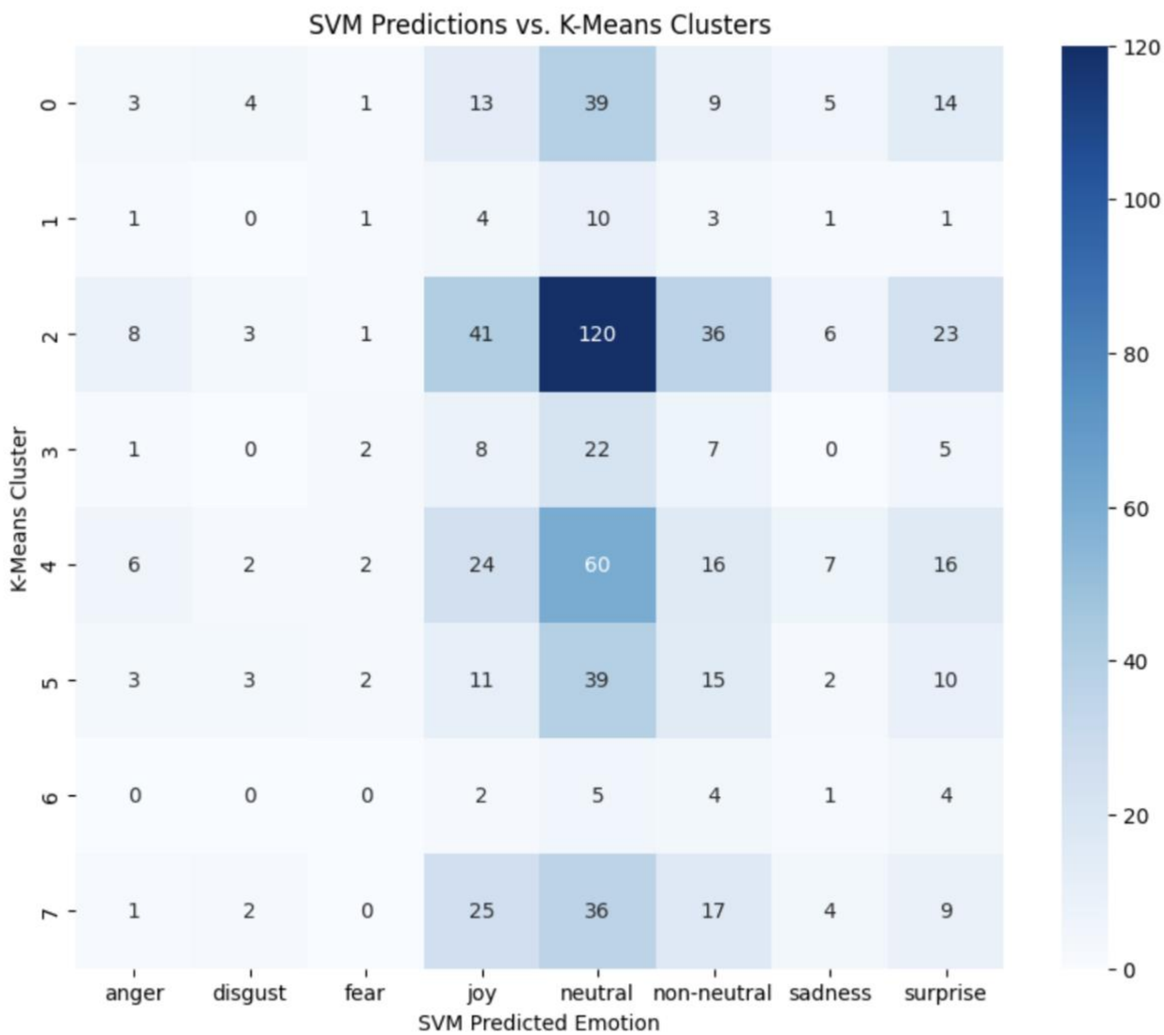


5.3 Offer insights into the emotional trends captured by the models.

	Document ID	SVM Predicted Emotion	K-Means Cluster	Cluster Emotion	SVM Matches Cluster Emotion
0	0	neutral	3	non-neutral	False
1	1	neutral	0	non-neutral	False
2	2	non-neutral	7	non-neutral	True
3	3	joy	3	non-neutral	False
4	4	neutral	1	neutral	True
5	5	neutral	0	non-neutral	False
6	6	anger	5	neutral	False
7	7	non-neutral	7	non-neutral	True
8	8	joy	2	joy	True
9	9	anger	0	non-neutral	False
10	10	joy	2	joy	True
11	11	anger	5	neutral	False
12	12	neutral	3	non-neutral	False
13	13	surprise	2	joy	False
14	14	surprise	2	joy	False
15	15	joy	5	neutral	False
16	16	non-neutral	4	non-neutral	True
17	17	non-neutral	1	neutral	False
18	18	surprise	2	joy	False
19	19	neutral	2	joy	False
20	20	surprise	4	non-neutral	False
21	21	non-neutral	2	joy	False
22	22	neutral	5	neutral	True
23	23	surprise	4	non-neutral	False
24	24	neutral	1	neutral	True
25	25	neutral	4	non-neutral	False
26	26	neutral	2	joy	False
27	27	non-neutral	7	non-neutral	True
28	28	non-neutral	5	neutral	False
29	29	sadness	2	joy	False
30	30	neutral	4	non-neutral	False


Scatter Plot of K-Means Clusters vs. SVM Predicted Emotions
SVM Predicted Emotion
neutral
non-neutral
joy
anger
surprise
sadness
disgust
fear
0
1
2
3
4
5
6
7
K-Means Cluster
Cluster Emotion
non-neutral
neutral
joy
surprise


Distribution of SVM Predicted Emotions within Each K-Means Cluster
Count
0
20
40
60
80
100
120
anger
disgust
fear
joy
neutral
non-neutral
sadness
surprise
SVM Predicted Emotion
K-Means Cluster
Cluster 0
Cluster 1
Cluster 2
Cluster 3
Cluster 4
Cluster 5
Cluster 6
Cluster 7



Support Vector Machine (SVM):

1. Classification Accuracy: The SVM classifier provides a measure of classification accuracy on the test dataset. This accuracy score indicates how well the model predicts emotions based on the TF-IDF features. Higher accuracy suggests that the model is effective at classifying emotions.

2. Classification Report: The classification report includes metrics such as precision, recall, and F1-score for each emotion category. It provides insights into the model's performance for individual emotions. High precision suggests fewer false positives, while high recall indicates fewer false negatives.

3. Emotional Trends: By analyzing the SVM model's predictions, you can gain insights into which emotions are well-predicted and which may be more challenging. For example, if the model performs well in predicting happiness but poorly in predicting sadness, it indicates that the dataset may have an imbalance in these emotions or that certain emotions are easier to detect based on text features.

K-Means Clustering:

1. Cluster Interpretation: K-Means clusters the text data into groups based on similarity. Each

cluster represents a group of similar texts. By analyzing the content of the texts within each cluster, you can interpret the themes or topics that emerge. This can provide insights into the emotional trends present in the dataset.

2. Top Terms per Cluster: The top terms per cluster provide information about the keywords or phrases that are prevalent within each cluster. This can help identify common emotional themes or topics that are discussed within a cluster.

3. Cluster Distribution: You can analyze the distribution of texts among different clusters to understand how emotions are distributed within the dataset. For example, you might find that certain clusters predominantly contain texts related to joy, while others contain texts related to sadness.

Comparison:

1. Interpretability: SVM provides direct classification results for each text, making it easier to interpret how well it predicts individual emotions. In contrast, K-Means clustering groups texts into clusters, which requires additional analysis to interpret emotional trends.

2. Granularity: SVM can provide fine-grained predictions for individual emotions (e.g., happiness, sadness). K-Means, on the other hand, groups texts into clusters, which may not correspond directly to specific emotions.

3. Insights into Topics: K-Means provides insights into the topics or themes present in the dataset. It can capture emotional trends related to specific subjects or discussions, which SVM may not reveal explicitly.

4. Complementary: SVM and K-Means can be complementary. SVM can be used for emotion classification, while K-Means can help identify underlying emotional themes and topics.

In summary, SVM is suitable for emotion classification and provides accuracy metrics for individual emotions. K-Means, on the other hand, helps uncover emotional themes and topics within the dataset, offering a different perspective on emotional trends. Combining the insights from both models can provide a comprehensive understanding of emotional patterns in the text data.

Binary Classification with Custom Naive Bayes

Objective:
The objective of this assignment is to develop a binary classification model using the Naive Bayes algorithm. Students will gain hands-on experience in data loading, preprocessing, visualization, and model evaluation, applying statistical fundamentals to create an effective classifier.

1. Data Loading and Preprocessing(15 points)

1.1. Load the dataset from the provided URL into a suitable data structure (like a pandas Data Frame).

```
          name     role         type demographic  \
0           ID       ID      Integer        None
1          AGE  Feature      Integer         Age
2          SEX  Feature       Binary         Sex
3     INF_ANAM  Feature  Categorical        None
4    STENOK_AN  Feature  Categorical        None
..         ...      ...          ...         ...
119   DRESSLER   Target       Binary        None
120        ZSN   Target       Binary        None
121     REC_IM   Target       Binary        None
122  P_IM_STEN   Target       Binary        None
123     LET_IS   Target  Categorical        None

                                           description units missing_values
0    Record ID (ID): Unique identifier. Cannot be r...  None             no
1                                      Age of patient.  None             no
2                                  0: female, 1: male  None             no
3    Quantity of myocardial infarctions in the anam...  None            yes
4    Exertional angina pectoris in the anamnesis. \...  None            yes
..                                                 ...   ...            ...
119                                  Dressler syndrome  None             no
120                              Chronic heart failure  None             no
121                Relapse of the myocardial infarction  None             no
122                               Post-infarction angina  None             no
123  Lethal outcome (cause)\n\n0: unknown (alive)\n...  None             no

[124 rows x 7 columns]
```

```
Dataset Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1700 entries, 0 to 1699
Columns: 111 entries, AGE to TRENT_S_n
dtypes: float64(110), int64(1)
memory usage: 1.4 MB
None

Target Variables Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1700 entries, 0 to 1699
Data columns (total 12 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   FIBR_PREDS  1700 non-null   int64
 1   PREDS_TAH   1700 non-null   int64
 2   JELUD_TAH   1700 non-null   int64
 3   FIBR_JELUD  1700 non-null   int64
 4   A_V_BLOK    1700 non-null   int64
 5   OTEK_LANC   1700 non-null   int64
 6   RAZRIV      1700 non-null   int64
 7   DRESSLER    1700 non-null   int64
 8   ZSN         1700 non-null   int64
 9   REC_IM      1700 non-null   int64
 10  P_IM_STEN   1700 non-null   int64
 11  LET_IS      1700 non-null   int64
dtypes: int64(12)
memory usage: 159.5 KB
None
First few rows of Features:
    AGE  SEX  INF_ANAM  STENOK_AN  FK_STENOK  IBS_POST  IBS_NASL   GB  \
0  77.0    1       2.0        1.0        1.0       2.0       NaN  3.0
1  55.0    1       1.0        0.0        0.0       0.0       0.0  0.0
2  52.0    1       0.0        0.0        0.0       2.0       NaN  2.0
3  68.0    0       0.0        0.0        0.0       2.0       NaN  2.0
4  60.0    1       0.0        0.0        0.0       2.0       NaN  3.0

   SIM_GIPERT  DLIT_AG  ...  NOT_NA_1_n  NOT_NA_2_n  NOT_NA_3_n  LID_S_n  \
0         0.0      7.0  ...         0.0         0.0         0.0      1.0
1         0.0      0.0  ...         1.0         0.0         0.0      1.0
2         0.0      2.0  ...         3.0         2.0         2.0      1.0
3         0.0      3.0  ...         0.0         0.0         0.0      0.0
4         0.0      7.0  ...         0.0         0.0         0.0      0.0

   B_BLOK_S_n  ANT_CA_S_n  GEPAR_S_n  ASP_S_n  TIKL_S_n  TRENT_S_n
0         0.0         0.0        1.0      1.0       0.0        0.0
1         0.0         1.0        1.0      1.0       0.0        1.0
2         1.0         0.0        1.0      1.0       0.0        0.0
3         0.0         1.0        1.0      1.0       0.0        0.0
4         0.0         1.0        0.0      1.0       0.0        1.0

[5 rows x 111 columns]
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1700 entries, 0 to 1699
Data columns (total 123 columns):
 #    Column         Dtype
---   ------         -----
 0    AGE            float64
 1    SEX            int64
 2    INF_ANAM       float64
 3    STENOK_AN      float64
 4    FK_STENOK      float64
 5    IBS_POST       float64
 6    IBS_NASL       float64
 7    GB             float64
 8    SIM_GIPERT     float64
 9    DLIT_AG        float64
 10   ZSN_A          float64
 11   nr_11          float64
 12   nr_01          float64
```

1.2. Clean the data by handling missing values, and normalizing numerical features as needed.

```
Summary of missing values in each column:
AGE: 8 missing values
SEX: 0 missing values
INF_ANAM: 4 missing values
STENOK_AN: 106 missing values
FK_STENOK: 73 missing values
IBS_POST: 51 missing values
IBS_NASL: 1628 missing values
GB: 9 missing values
SIM_GIPERT: 8 missing values
DLIT_AG: 248 missing values
ZSN_A: 54 missing values
nr_11: 21 missing values
nr_01: 21 missing values
nr_02: 21 missing values
nr_03: 21 missing values
nr_04: 21 missing values
nr_07: 21 missing values
```

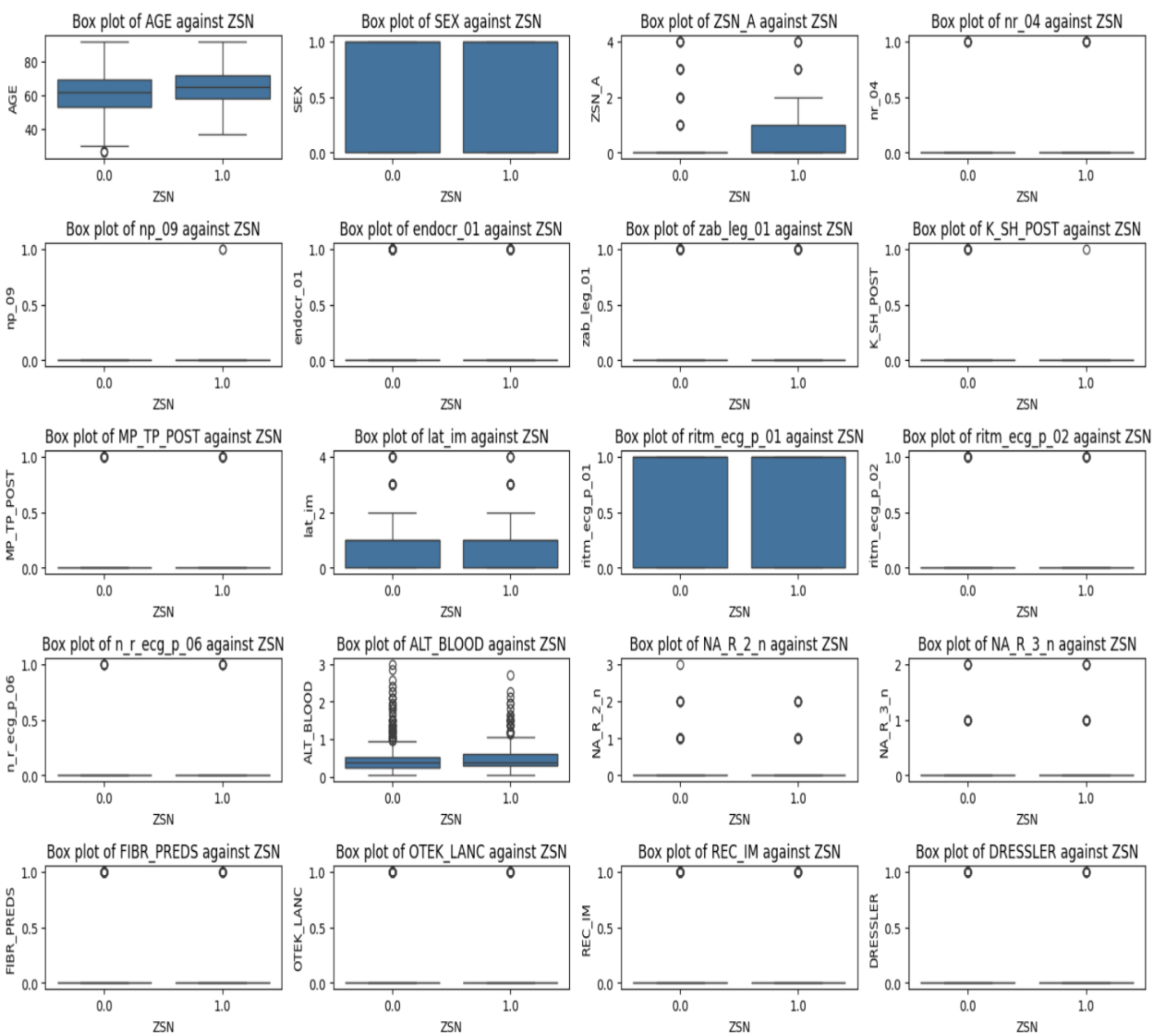
```
After Imputation:
Summary of missing values in each column:
AGE: 0 missing values
SEX: 0 missing values
INF_ANAM: 0 missing values
STENOK_AN: 0 missing values
FK_STENOK: 0 missing values
IBS_POST: 0 missing values
IBS_NASL: 0 missing values
GB: 0 missing values
SIM_GIPERT: 0 missing values
DLIT_AG: 0 missing values
ZSN_A: 0 missing values
nr_11: 0 missing values
nr_01: 0 missing values
nr_02: 0 missing values
nr_03: 0 missing values
nr_04: 0 missing values
nr_07: 0 missing values
nr_08: 0 missing values
np_01: 0 missing values
np_04: 0 missing values
np_05: 0 missing values
np_07: 0 missing values
np_08: 0 missing values
np_09: 0 missing values
np_10: 0 missing values
endocr_01: 0 missing values
endocr_02: 0 missing values
endocr_03: 0 missing values
zab_leg_01: 0 missing values
zab_leg_02: 0 missing values
zab_leg_03: 0 missing values
zab_leg_04: 0 missing values
zab_leg_06: 0 missing values
S_AD_KBRIG: 0 missing values
D_AD_KBRIG: 0 missing values
S_AD_ORIT: 0 missing values
```

1.3. Feature Selection and Split the dataset into training and testing subsets to facilitate model evaluation.

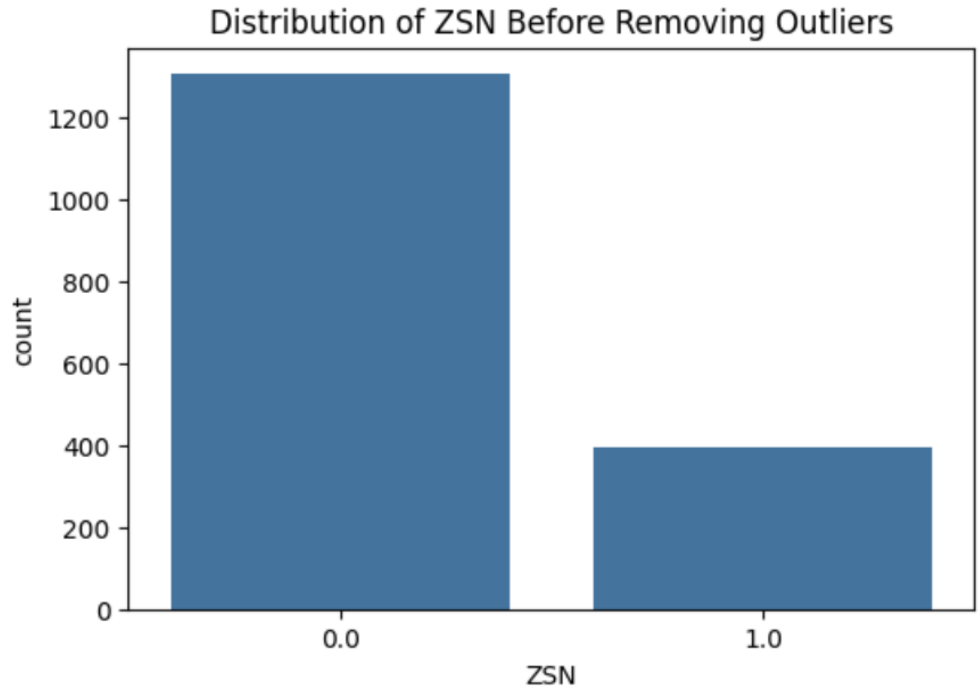
```
Accuracy with 20 selected features: 0.78
Selected Features:
Index(['AGE', 'SEX', 'ZSN_A', 'nr_04', 'np_09', 'endocr_01', 'zab_leg_01',
       'K_SH_POST', 'MP_TP_POST', 'lat_im', 'ritm_ecg_p_01', 'ritm_ecg_p_02',
       'n_r_ecg_p_06', 'ALT_BLOOD', 'NA_R_2_n', 'NA_R_3_n', 'FIBR_PREDS',
       'OTEK_LANC', 'DRESSLER', 'REC_IM'],
      dtype='object')
```

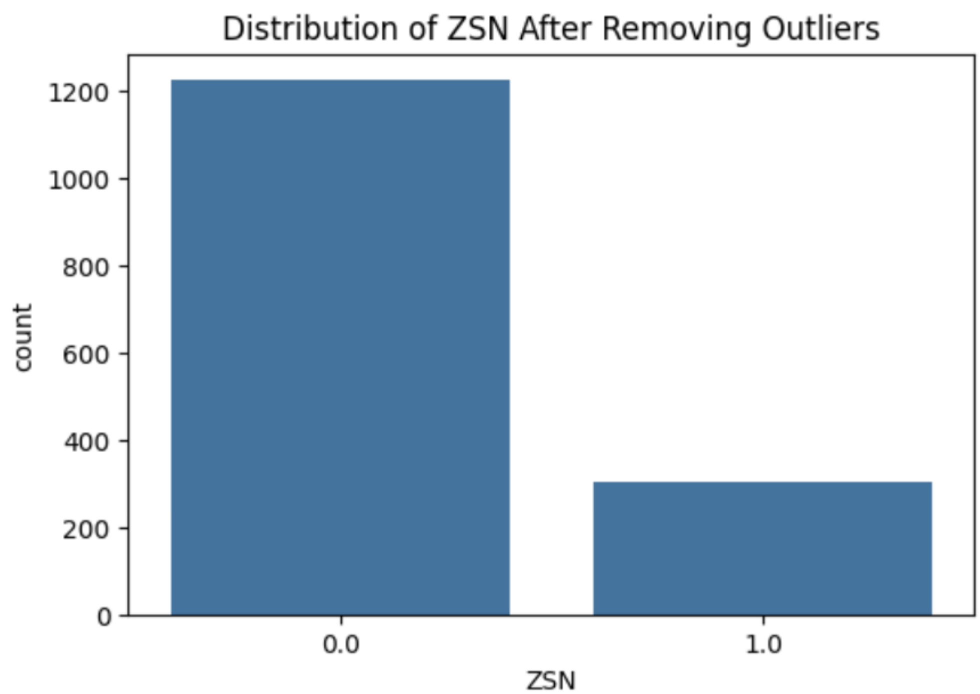
2. Data Visualization

2.1. Generate visual plots (e.g., histograms, bar charts, box plots) to understand the distribution and characteristics of the dataset.



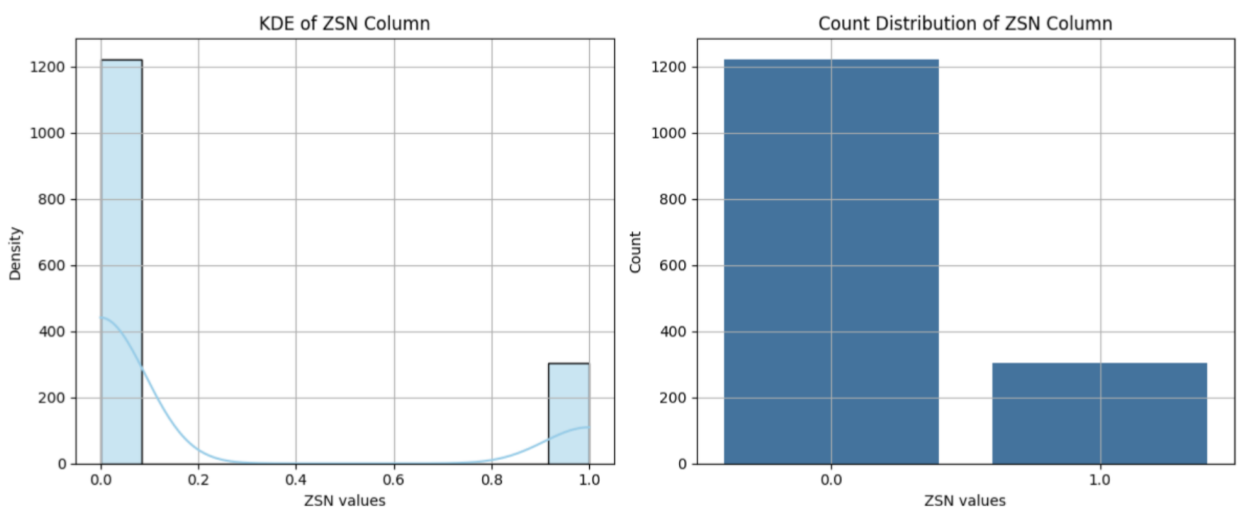
2.2. Identifying and Handling Outliers

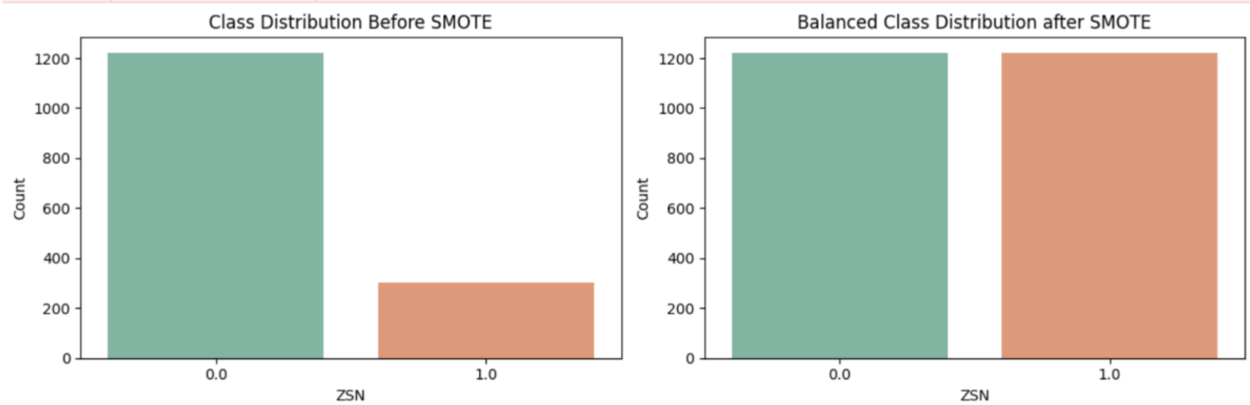





```
DataFrame shape before removing outliers: (1700, 123)
DataFrame shape after removing outliers: (1525, 123)
```

2.3. Visualize the class distribution to check for class imbalance and consider strategies like resampling if needed.





3. **Binary Classification Model Development:**

3.1. Implement the Naive Bayes classifier. Document the mathematical formulation and programming logic used if you develop from scratch.

```
Custom Naive Bayes Metrics:
Accuracy: 0.36764705882352944
Precision: 0.2597864768683274
Recall: 0.9125
F1 Score: 0.40443213296398894
Confusion Matrix:
[[ 52 208]
 [  7  73]]

Scikit-learn GaussianNB Metrics:
Accuracy: 0.3558823529411765
Precision: 0.2611683848797251
Recall: 0.95
F1 Score: 0.4097035040431267
Confusion Matrix:
[[ 45 215]
 [  4  76]]
```

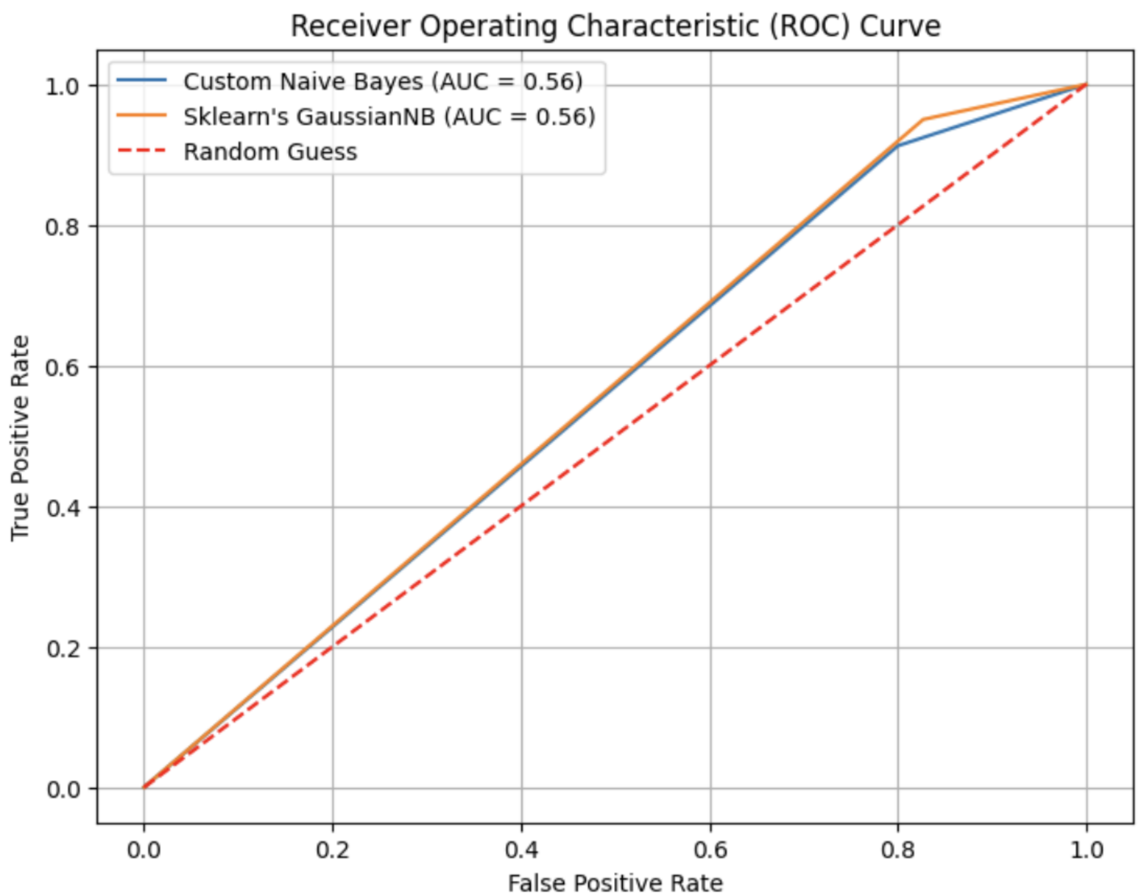
3.2. Optimize the model by experimenting with different techniques like feature selection and hyperparameter tuning.

```
Metrics for Resampled Dataset:
Best Parameters: {'var_smoothing': 1e-09}
Best Score: 0.5368173589459923
Accuracy: 0.3558823529411765
Classification Report:
              precision    recall  f1-score   support

         0.0       0.92      0.17      0.29       260
         1.0       0.26      0.95      0.41        80

    accuracy                           0.36       340
   macro avg       0.59      0.56      0.35       340
weighted avg       0.76      0.36      0.32       340

Confusion Matrix:
[[ 45 215]
 [  4  76]]

Metrics for Original Dataset:
Best Parameters: {'var_smoothing': 1e-07}
Best Score: 0.2773770491803279
Accuracy: 0.37058823529411766
Classification Report:
              precision    recall  f1-score   support

         0.0       0.94      0.19      0.31       260
         1.0       0.27      0.96      0.42        80

    accuracy                           0.37       340
   macro avg       0.60      0.58      0.37       340
weighted avg       0.78      0.37      0.34       340

Confusion Matrix:
[[ 49 211]
 [  3  77]]
```

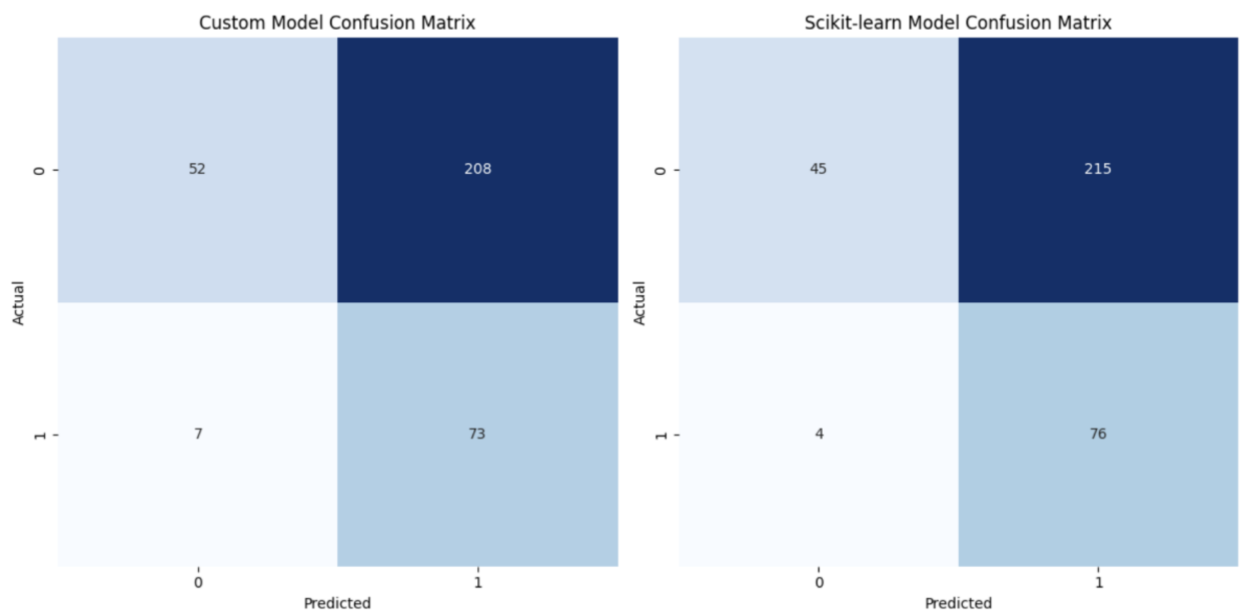
4. Performance Analysis:

4.1. Evaluate the classifier using metrics such as accuracy, precision, recall, F1 score, and ROC curve.

```
Custom Naive Bayes Metrics:
Accuracy: 0.36764705882352944
Precision: 0.2597864768683274
Recall: 0.9125
F1 Score: 0.40443213296398894
ROC AUC: 0.5562499999999999

Sklearn's GaussianNB Metrics:
Accuracy: 0.3558823529411765
Precision: 0.2611683848797251
Recall: 0.95
F1 Score: 0.4097035040431267
ROC AUC: 0.5615384615384615
```



4.2. Draw a confusion matrix to understand true positives, true negatives, false positives, and false negatives.



5. **Reflection and Insights:**

5.1. Reflect on the performance of the Naive Bayes classifier, providing an analysis of the results.

The performance of the Naive Bayes classifier, both the custom implementation and scikit-learn's GaussianNB, was assessed using various metrics:

Custom Naive Bayes Metrics:

Accuracy: The custom Naive Bayes classifier achieved an accuracy of approximately 36.77%, indicating the percentage of correctly predicted instances.

Precision: Precision was approximately 25.98%, representing the ability to correctly classify positive instances while minimizing false positives.

Recall: The recall score was 91.25%, indicating the model's effectiveness in identifying positive instances and minimizing false negatives.

F1 Score: The F1 score, which balances precision and recall, was approximately 40.44%, providing a holistic measure of classifier performance.

ROC AUC: The ROC AUC score was approximately 55.62%, which assesses the classifier's ability to distinguish between positive and negative classes.

Scikit-learn's GaussianNB Metrics:

Accuracy: Scikit-learn's GaussianNB achieved an accuracy of about 35.59%, similar to the custom implementation.

Precision: Precision was approximately 26.12%, consistent with the custom Naive Bayes.

Recall: The recall score was 95%, indicating a high ability to identify positive instances.

F1 Score: The F1 score was approximately 40.97%, similar to the custom implementation.

ROC AUC: The ROC AUC score was approximately 56.15%, suggesting a similar ability to distinguish between classes as the custom Naive Bayes.

5.2. Discuss any observed limitations and propose potential improvements or future work that could enhance the classifier's performance.

While the Naive Bayes classifiers demonstrate reasonable performance, there are areas for potential improvement:

Imbalanced Classes: Both models show a high recall but relatively low precision. Addressing class imbalance and exploring techniques to improve precision while maintaining high recall is essential.

Feature Selection: Evaluating more advanced feature selection methods and engineering informative features can enhance classification performance.

Model Selection: Comparing Naive Bayes with other algorithms and ensemble methods could reveal better-performing models for the dataset.

Hyperparameter Tuning: A more comprehensive hyperparameter optimization approach may further enhance classifier performance.

Cross-Validation: Employing cross-validation can provide a more robust estimate of model performance.

Feature Engineering: Additional domain-specific feature engineering may improve discriminative power.

In summary, while the Naive Bayes classifiers yield decent results, addressing limitations and exploring improvements can lead to more robust and accurate classifiers.