Sri Sankeerth Koduru - G42612138

Assignment 2:

Question 1:

Data Cleaning:

The following steps were taken to clean the data:

Loading the Dataframe:

This code snippet loads a dataset from the UCI Machine Learning Repository (specifically, ID 579). It then extracts features (X) and targets (y) into a pandas DataFrame. Finally, it focuses on the 'ZSN' target and drops the unnecessary 'ZSN_A' column.

Removing Columns:

After loading and examining the dataset, I focused on the discrete variables. I observed that the 'KFK_BLOOD’ column contained mostly unique values. This suggests that the 'KFK_BLOOD’ column offers limited predictive power for understanding our target variable. Consequently, I removed these columns from the dataset.

Feature Engineering:

After string encoding, I generated a correlation matrix and selected variables with correlations exceeding an absolute value of 0.075. The 'AGE', 'endocr_01', 'NA_R_3_n', 'zab_leg_01', 'n_r_ecg_p_06', 'ritm_ecg_p_02', 'NA_R_2_n', 'nr_04', 'MP_TP_POST', 'R_AB_3_n', 'NOT_NA_KB', 'lat_im', 'K_SH_POST', 'SEX', and 'IBS_NASL'were used to build a new dataframe named df_feature_selection.

Dealing with Null Values:

Following the feature engineering phase, I performed a thorough data quality assessment, pinpointing a solitary null value within the target variable. Notably, the columns 'IBS_NASL' and 'NOT_NA_KB' exhibited a significant number of null entries, prompting the decision to eliminate corresponding rows from the dataset. Considering the sparse occurrence of missing data and its impact on key columns, I chose to exclude the row containing the null value.

Dealing with Outliers:

After dealing with null values, I calculated Z-scores for each column to identify outliers exceeding 3 standard deviations from the mean. Using a threshold of 3 standard deviations, I identified and removed the outliers. Finally, I reviewed the cleaned dataset, confirming its size and inspecting the first few rows.

Normalized Data:

After dealing with outliers, I used the StandardScalar function to normalize all the features to prevent feature domination and ensure feature Interpretability.

SMOTE Data:

We used the SMOTE (Synthetic Minority Oversampling Technique) algorithm to address the class imbalance in the dataset. SMOTE creates synthetic samples for the minority class, improving the balance between classes and potentially enhancing model performance.

Data Visualization:

To begin the visualization process, I generated a correlation matrix to identify features with the strongest impact on the model. Histograms were created for these important features to understand their distributions, which is essential for selecting appropriate modeling techniques. Finally, I used pairplots to gain a comprehensive understanding of the dataset as a whole

Machine Learning Algorithm:

Data Split:

After normalizing the data, I slip the data into a training and testing dataset. The training set consists of 80% of the data (1300 rows) and the testing set consists of 20% of the data (326 rows).

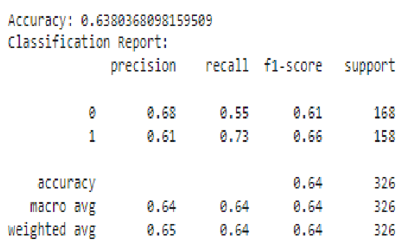
Machine Learning Algorithm:

BernoulliNB Model:

The BernoulliNB model, a member of the Naive Bayes family of algorithms, is designed specifically for binary features. This makes it a strong choice for text classification, where the presence or absence of words is a key factor. In an initial test, this model achieved an accuracy of 63.8%, with precision of 68% (class 0) and 61% (class 1). Recall rates were 55% (class 0) and 73% (class 1).

To enhance performance, parameter optimization via grid search was employed. This process identified ideal settings, including an alpha value of 0.5 and a binarization threshold of 0.5. The tuned BernoulliNB model demonstrated a marked improvement, achieving an accuracy of 69.9%. Precision also increased to 71% (class 0) and 69% (class 1). Improved recall rates further underscore the performance gains. This case study clearly demonstrates the power of parameter tuning in maximizing the effectiveness of machine learning models.

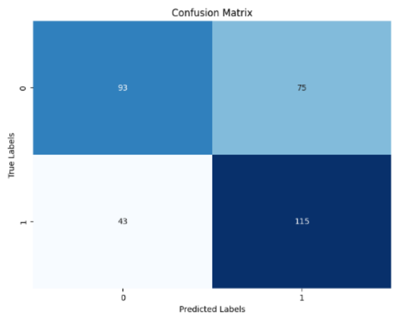
Image 1: BernoulliNB Baseline Model Metrics



```
Accuracy: 0.6380368098159509
Classification Report:
              precision    recall  f1-score   support

           0       0.68      0.55      0.61       168
           1       0.61      0.73      0.66       158

    accuracy                           0.64       326
   macro avg       0.64      0.64      0.64       326
weighted avg       0.65      0.64      0.64       326
```



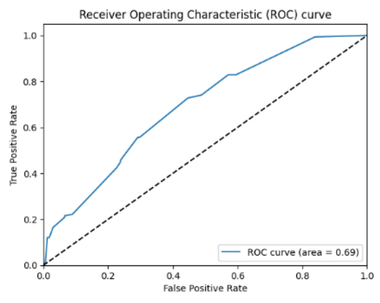
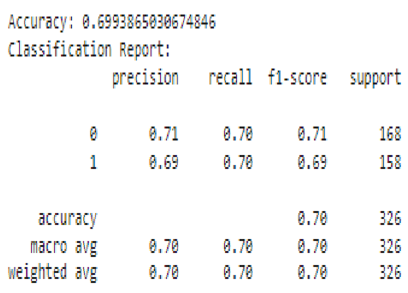


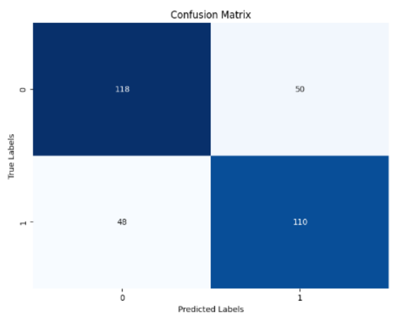
Image 2: BernoulliNB Enhanced Model Metrics

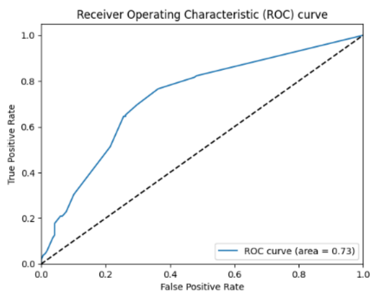


```
Accuracy: 0.6993865030674846
Classification Report:
              precision    recall  f1-score   support

           0       0.71      0.70      0.71       168
           1       0.69      0.70      0.69       158

    accuracy                           0.70       326
   macro avg       0.70      0.70      0.70       326
weighted avg       0.70      0.70      0.70       326
```





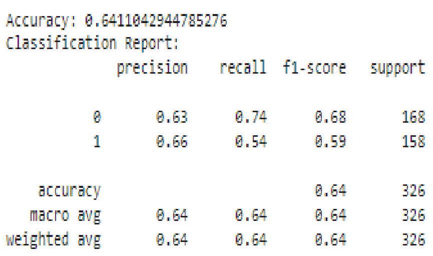
GaussianNB Model:

The GaussianNB model is a versatile machine learning algorithm based on the assumption that features within each class follow a Gaussian (normal) distribution. This makes it well-suited for handling continuous data. In contrast, the BernoulliNB model is designed for binary features.

In a baseline experiment, a GaussianNB model initially achieved an accuracy of 64%, with class 0 precision at 63% and class 1 precision at 66%. Through careful parameter tuning, specifically optimizing the var_smoothing parameter to 1.0 via grid search, the model's performance was enhanced. The tuned model boasts increased accuracy (65%), a higher precision for class 1 (69%), and an improved recall for class 0 (79%). This demonstrates the crucial role of parameter tuning in maximizing model performance.

While these results show improvement, it's important to note that the best performing BernoulliNB model on this dataset achieved an even higher accuracy of 70%. This underlines the significance of selecting the most suitable model for a given dataset and task.

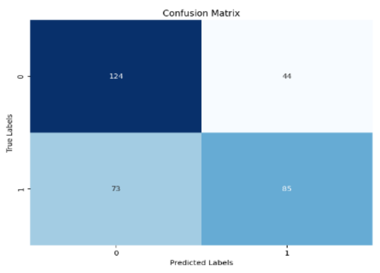
Image 3: GaussianNB Baseline Model Metrics



```
Accuracy: 0.6411042944785276
Classification Report:
              precision    recall  f1-score   support

           0       0.63      0.74      0.68       168
           1       0.66      0.54      0.59       158

    accuracy                           0.64       326
   macro avg       0.64      0.64      0.64       326
weighted avg       0.64      0.64      0.64       326
```



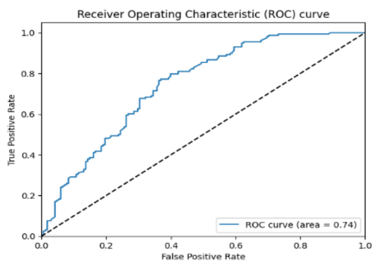
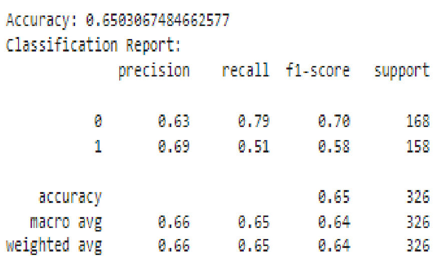


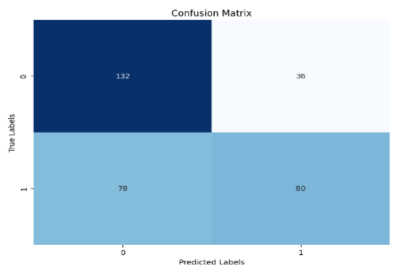
Image 4: GaussianNB Enhanced Model Metrics

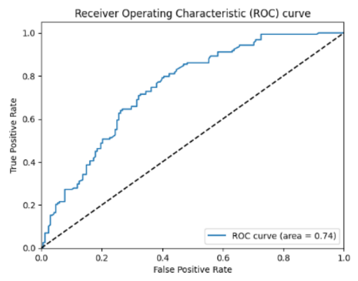


```
Accuracy: 0.6503067484662577
Classification Report:
              precision    recall  f1-score   support

           0       0.63      0.79      0.70       168
           1       0.69      0.51      0.58       158

    accuracy                           0.65       326
   macro avg       0.66      0.65      0.64       326
weighted avg       0.66      0.65      0.64       326
```





Conclusion:

This project demonstrated the importance of thorough data preparation for machine learning success. Data cleaning, feature engineering, and addressing class imbalance were key steps. The BernoulliNB model, after parameter tuning, outperformed the GaussianNB model on this dataset. This highlights the need to experiment with different algorithms and optimize their settings for the specific task at hand.

Challenges Faced:

- Significant class imbalance in the medical dataset.
- Difficulty analysing a large volume of complex medical data columns due to lack of domain-specific knowledge.

Future Work:

- Experiment with alternative data sampling techniques beyond SMOTE (e.g., random under sampling, ADASYN, combinations) to assess potential model performance gains.

- Refine feature engineering processes, potentially incorporating domain expertise, to enhance model input quality.
- Evaluate a range of machine learning models (e.g., decision trees, ensemble methods, or suitable deep learning architectures) to identify the best fit for the problem.

Question 2:

Data Preparation:

Loading the Model:

We begin by loading our JSON datasets: "friends_train.json" for training, "friends_dev.json" for development, and "friends_test.json" for our final holdout evaluation. These are structured into DataFrames (df_train, df_dev, df_test) to organize the data for the subsequent phases.

Processing Text:

The preprocess_text function is responsible for cleaning and normalizing our text. It lowercases the text, tokenizes it into words, reduces words to their root form using lemmatization, and filters out common stop words. This refined text is added as a 'preprocessed_utterance' column to each DataFrame.

Vectorizer:

We employ the TfidfVectorizer to create numerical TF-IDF representations of the text. TF-IDF highlights the importance of words within a document and across the corpus. We fit the vectorizer to the training data (tfidf_train) and transform the development (tfidf_dev) and holdout test (tfidf_test) sets using the learned vocabulary. These numerical vectors will serve as input for our machine learning model.

Data Visualization:

Histogram analysis confirms a class imbalance problem in all datasets, with the 'neutral' emotion dominating the distribution. This skewed distribution could hinder the models' ability to learn the nuances of other emotional categories.

SVM Model:

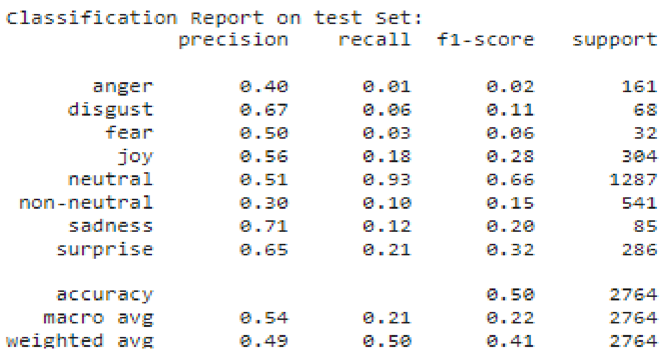
Model:

This sentiment analysis task employed a Support Vector Machine (SVM) for classification. To optimize performance, a grid search strategy was used with cross-validation (GridSearchCV) to tune hyperparameters including 'C' (regularization penalty), 'gamma' (influence of training data), and 'kernel' (data transformation). The optimal configuration found was {'C': 1, 'gamma': 1, 'kernel': 'rbf'}. This combination helps the SVM strike a balance between accurately fitting the training data and avoiding overfitting, leading to improved generalization on new, unseen sentiment examples.

Test Metrics:

The class imbalance significantly influences the model's performance on the holdout set, leading to an over-prediction of the "neutral" category. This likely stems from the dataset's skew toward neutral emotions, causing the model to prioritize this majority class. As a result, the model struggles to accurately classify other emotions and exhibits high false positives for the "neutral" class. This impact is evident in the low F1-scores for minority classes and further highlighted by the confusion matrix.

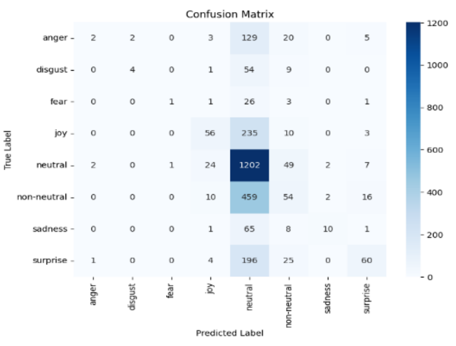
Image 5: SVM Model Metrics



```
Classification Report on test Set:
              precision    recall  f1-score   support

       anger       0.40      0.01      0.02       161
     disgust       0.67      0.06      0.11        68
        fear       0.50      0.03      0.06        32
         joy       0.56      0.18      0.28       304
     neutral       0.51      0.93      0.66      1287
 non-neutral       0.30      0.10      0.15       541
     sadness       0.71      0.12      0.20        85
    surprise       0.65      0.21      0.32       286

    accuracy                           0.50      2764
   macro avg       0.54      0.21      0.22      2764
weighted avg       0.49      0.50      0.41      2764
```



	anger	disgust	fear	joy	neutral	non-neutral	sadness	surprise
anger	2	2	0	3	129	20	0	5
disgust	0	4	0	1	54	9	0	0
fear	0	0	1	1	26	3	0	1
joy	0	0	0	56	235	10	0	3
neutral	2	0	1	24	1202	49	2	7
non-neutral	0	0	0	10	459	54	2	16
sadness	0	0	0	1	65	8	10	1
surprise	1	0	0	4	196	25	0	60

<u>**KNN Means Model:**</u>

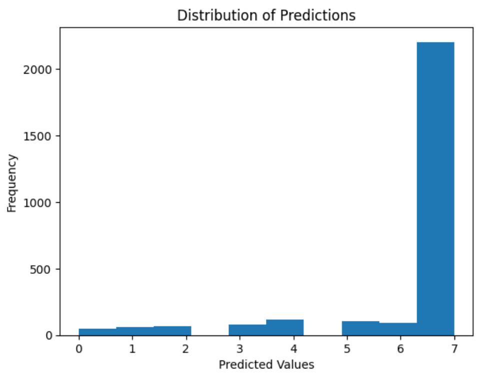
Model:

This sentiment analysis task utilizes the K-means clustering algorithm to group text data based on similarities in their TF-IDF representations. A grid search strategy with cross-validation (GridSearchCV) is employed to optimize the performance of the K-means model. Key hyperparameters, including the number of clusters ('n_clusters'), the initialization method ('init') for cluster centers, and the maximum number of iterations ('max_iter'), are systematically explored. This process aims to identify the configuration that leads to the most meaningful and well-defined clusters within the sentiment data. The optimal set of hyperparameters found was {'init': 'k-means++', 'max_iter': 300, 'n_clusters': 8}.

Test Metrics:

While the model correctly identified the presence of 8 clusters, the histogram of predictions reveals a significant imbalance in their distribution. A disproportionate number of data points are assigned to a single cluster, likely representing the neutral category. The remaining clusters have significantly fewer, or in some cases, no predictions associated with them. This pattern strongly suggests a class imbalance within the dataset, with neutral emotions dominating the data. Consequently, the model might be biased towards this majority class, hindering its ability to accurately discern and group other, less-represented emotions.

Image 6: K-Means Model Histogram



Comparison and Conclusion:

SVM and K-Means have fundamentally different approaches to emotion classification. SVM is a supervised algorithm, requiring labelled data to categorize emotions. In contrast, K-Means is unsupervised, grouping data based on similarities without predetermined labels. SVM's success hinges on metrics like precision, recall, and f1-scores, while K-Means relies on cluster quality indicators. However, both models currently struggle with the dataset, incorrectly identifying most emotions as neutral. This suggests class imbalance is a significant hurdle, and potentially that SVM and K-Means may not be the best fit for text classification.

Challenges Faced:

- Significant class imbalance in the medical dataset.
- Models may not be ideal to do text classification.

Future Work:

- Experiment with data sampling to assess potential model performance gains.
- Refine feature engineering by considering stemming (instead of lemmatization) and experimenting with alternative vectorizers.
- Try to utilize different models (i.e. neural networks, encoder-decoder models, and transformer models)