techdome
solutions

Title: Building a Sentiment Analysis Model using Natural Language Processing Techniques

Objective:

The objective of this project is to construct a robust sentiment analysis model utilizing Natural Language Processing (NLP) techniques. Sentiment analysis, also known as opinion mining, is a fundamental aspect of artificial intelligence with widespread applications in customer feedback analysis, social media monitoring, and market sentiment analysis. The primary goal is to develop a model capable of accurately categorizing text data into positive, negative, or neutral sentiments.

Data Preprocessing Steps:

1. Data Loading and Exploration:

- Loaded the dataset from the CSV file using the `pd.read_csv()` function.
- Specified the encoding type as 'latin1' to handle potential encoding issues.
- Explored the shape of the dataset using `data.shape` to determine the number of rows and columns.
- Examined the column names using `data.columns` to identify the features available.
- Inspected unique values in the 'label' column using `data['label'].unique()` to understand the distribution of labels.
- Displayed the first few rows of the dataset using `data.head()` to get an overview of the data.
- Adjusted the display settings to show full comments using `pd.set_option('display.max_colwidth', None)`.

2. Handling Missing Values:

Checked for null values in the dataset using `data.isnull().sum()` to ensure data integrity.

3. Data Visualization:

- Visualized the distribution of labels using a bar plot to identify any data imbalances.
- Plotted the counts of different labels to observe the frequency of each class.

4. Text Preprocessing:

Certainly! Let's break down each step of text preprocessing as implemented in your code:

- Converting Text to Lowercase:
  - In this step, all text data in the 'comment' column is converted to lowercase. This is done to ensure uniformity in text data, as uppercase and lowercase versions of the same word may be treated differently during subsequent processing steps.

Converted text data to lowercase using data['comment'].str.lower() to maintain consistency

- Removing Special Characters, Punctuation, and Symbols:

- The `remove_special_characters()` function is defined to remove any special characters, punctuation marks, and symbols from the text data. Regular expressions (regex) are used to define a pattern that matches any characters outside the range of alphanumeric characters (letters and numbers). The `re.sub()` function is then used to replace these characters with an empty string, effectively removing them from the text.

Defined a function remove_special_characters() to remove special characters, punctuation, and symbols from the text data

- Removing Stopwords:

- Stopwords are common words such as 'and', 'the', 'is', etc., that do not carry significant meaning in text analysis and are often removed during preprocessing. The NLTK library's stopwords corpus is used to obtain a set of English stopwords. The `remove_stopwords()` function is defined to remove stopwords from the text data by splitting the text into individual words and filtering out any words that are found in the stopwords set.

Utilized the NLTK library to remove stopwords from the text using the remove_stopwords() function

- Tokenization:

- Tokenization is the process of splitting text into individual words or tokens. The NLTK library's `word_tokenize()` function is used to tokenize the cleaned text data into a list of tokens (words). This step is essential for further analysis and processing of text data at the word level.

Tokenized the cleaned text data into individual words using the tokenize_text() function from NLTK

- Stemming:

- Stemming is the process of reducing words to their root or base form by removing suffixes and prefixes. The Porter Stemmer algorithm, implemented in the NLTK library's `PorterStemmer`, is used to stem the tokens. Stemming helps in reducing the dimensionality of the feature space by collapsing similar words to their common root form.

Employed the Porter Stemmer algorithm to stem the tokens using the stem_tokens() function.

- Lemmatization:

- Lemmatization is similar to stemming but aims to reduce words to their canonical form or lemma. Unlike stemming, lemmatization considers the context of the word in the sentence and ensures that the resulting lemma is a valid word. The NLTK library's WordNet lemmatizer (`WordNetLemmatizer`) is used to lemmatize the tokens.

Utilized the WordNet lemmatizer to lemmatize the tokens using the lemmatize_tokens() function

- Combining Processed Tokens:

- Finally, the processed tokens (stemmed or lemmatized) are combined back into cleaned text data. The `lambda` function is used along with the `join()` method to concatenate the tokens into a single string, which is then stored in a new column named 'cleaned_text'.

Combined the processed tokens back into cleaned text data for further analysis using lambda function. text data in a new column named 'cleaned_text' for subsequent analysis and modeling

- ❖ The notebook is attached with name DataCleaning and EDA where all of these are performed.

TF-IDF Vectorization

TF-IDF is commonly used in sentiment analysis tasks as part of the feature extraction process. In sentiment analysis, the goal is to determine the sentiment or opinion expressed in a piece of text, such as whether it is positive, negative, or neutral. TF-IDF helps in this process by transforming the text data into numerical feature vectors, which can then be used as input to machine learning models for sentiment classification.

Here's how TF-IDF is used in sentiment analysis:

- Text Preprocessing:

- Before applying TF-IDF, the text data undergoes preprocessing steps such as tokenization, removing stopwords, and stemming or lemmatization to prepare it for analysis.

- TF-IDF Vectorization:

- TF-IDF is applied to the preprocessed text data to convert it into a numerical representation. Each document (or text sample) is represented as a vector where each dimension corresponds to a unique term in the entire corpus of documents.

- The TF-IDF score of each term in a document is calculated using the formulas mentioned earlier, reflecting both the term's frequency within the document and its rarity across the entire dataset.

- Feature Matrix Creation:

- The TF-IDF scores for each term in each document are organized into a feature matrix, where each row represents a document and each column represents a term. This matrix serves as the input to the sentiment analysis model.

- Model Training:

- The feature matrix obtained from TF-IDF is used to train a sentiment analysis model. This model learns patterns in the TF-IDF-weighted feature space to predict the sentiment of unseen text data.

By incorporating TF-IDF as part of the feature extraction process, sentiment analysis models can effectively capture the importance of words in determining sentiment while considering their frequency and rarity across the entire dataset. This helps in improving the accuracy and effectiveness of sentiment classification tasks.

Splitting Data into Training and Testing

The `train_test_split` function from the `sklearn.model_selection` module is a utility used to partition a dataset into separate training and testing sets. Its primary purpose is to enable the evaluation of machine learning models by training them on one subset of the data and then testing their performance on another subset.

1. Input Parameters:

- Arrays: It takes arrays (or matrices) as input, typically representing features and labels.
- Test Size: This parameter specifies the proportion of the dataset to include in the testing split. It's usually expressed as a float between 0 and 1, where 0.2 means 20% of the data will be used for testing.
- Random State: By setting a random state, you ensure reproducibility. It controls the randomness of the data splitting process.
- Stratify: When dealing with classification tasks, setting `stratify` ensures that the distribution of class labels in the training and testing sets remains similar to that of the original dataset.

2. Output:

The function returns multiple arrays representing the split data:

- `X_train`: The features (or input variables) for training.
- `X_test`: The features for testing.
- `y_train`: The corresponding labels (or output variables) for training.
- `y_test`: The labels for testing.

3. Purpose:

- Training: The training set is used to fit the machine learning model, allowing it to learn patterns and relationships in the data.
- Testing: The testing set evaluates the model's performance on unseen data, providing an estimate of how well it generalizes to new observations.

4. Evaluation:

After training a model on the training data, you can use the testing set to assess its performance metrics, such as accuracy, precision, recall, or F1-score.

In summary, `train_test_split` facilitates the proper evaluation of machine learning models by splitting the dataset into distinct subsets for training and testing, ensuring unbiased assessments of model performance.

- ❖ The notebook is attached with name Train-Test Split where all of these are performed

Logistic Regression Model: -

Model Architecture and Parameters:

The sentiment analysis model is based on logistic regression. Logistic regression is a linear classification algorithm commonly used for binary classification tasks. Here's an overview of the model architecture and parameters:

- Model Type: Logistic Regression
- TF-IDF Vectorization: Used TfidfVectorizer for feature extraction with a maximum of 5000 features.
- Regularization: The logistic regression model employs L2 regularization by default.
- Solver: The solver used for optimization is 'lbfgs', which is suitable for small datasets.

Training Process and Hyperparameters:

The training process involves the following steps:

1. Data Preparation: Text data is preprocessed, including text cleaning, tokenization, stopword removal, and TF-IDF vectorization.
2. Label Encoding: Categorical labels are encoded into numerical labels using LabelEncoder.
3. Train-Test Split: The dataset is split into training and testing sets with a ratio of 80:20.
4. Model Training: The logistic regression model is trained on the training set using the fit method.
5. Model Evaluation: The trained model is evaluated on both the training and testing sets using accuracy, precision, recall, and F1-score metrics.

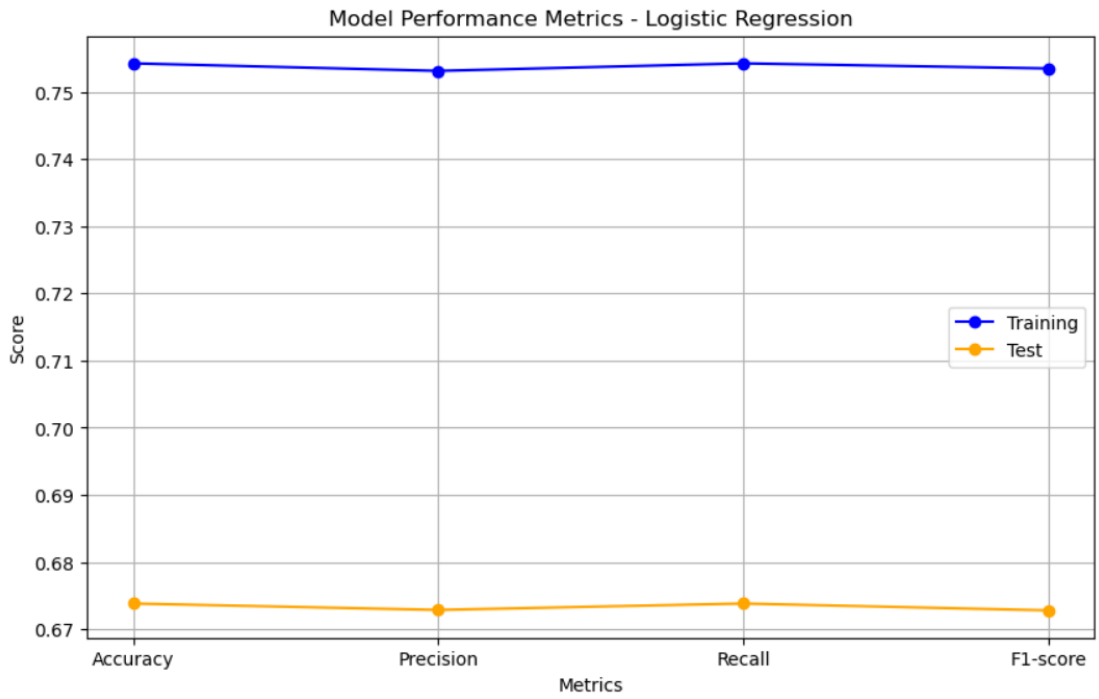
Evaluation Results and Analysis:

The evaluation results on both the training and testing sets are as follows:

- Training Accuracy: 0.754
- Test Accuracy: 0.674
- Training Precision: 0.753
- Test Precision: 0.673
- Training Recall: 0.754
- Test Recall: 0.674
- Training F1-score: 0.753
- Test F1-score: 0.673

Analysis:

- The logistic regression model achieves an accuracy of 67.4% on the testing set.
- Precision measures the proportion of correctly predicted instances among the instances predicted as positive. The model achieves a precision of 67.3% on the testing set.
- Recall measures the proportion of correctly predicted instances among all actual positive instances. The model achieves a recall of 67.4% on the testing set.
- F1-score is the harmonic mean of precision and recall, providing a balance between the two metrics. The model achieves an F1-score of 67.3% on the testing set



- The notebook is attached with name LogisticRegression where all of these are performed.

Support Vector Machine (SVM) Model:

Model Architecture and Parameters:

The sentiment analysis model utilizes the Support Vector Machine (SVM) algorithm with a linear kernel. Here's an overview of the model architecture and parameters:

- Model Type: Support Vector Machine (SVM)
- Kernel: Linear kernel is employed for SVM classification.
- TF-IDF Vectorization: TfidfVectorizer is used for feature extraction, with a maximum of 5000 features.

Training Process and Hyperparameters:

The training process involves the following steps:

- Data Preparation: Text data is preprocessed, including text cleaning, tokenization, stopword removal, and TF-IDF vectorization.
- Label Encoding: Categorical labels are encoded into numerical labels using LabelEncoder.
- Train-Test Split: The dataset is split into training and testing sets with an 80:20 ratio.
- Model Training: The SVM model is trained on the training set using the fit method with the linear kernel.
- Model Evaluation: The trained model is evaluated on both the training and testing sets using accuracy, precision, recall, and F1-score metrics.

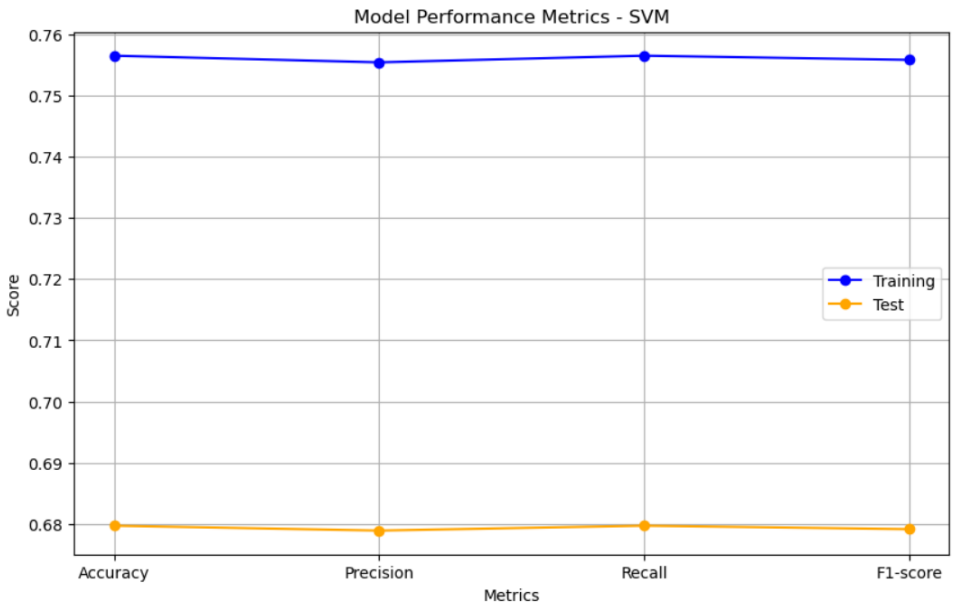
Evaluation Results and Analysis:

The evaluation results on both the training and testing sets are as follows:

- Training Accuracy: 75.7%
- Test Accuracy: 68.0%
- Training Precision: 75.5%
- Test Precision: 67.9%
- Training Recall: 75.7%
- Test Recall: 68.0%
- Training F1-score: 75.6%
- Test F1-score: 67.9%

Analysis:

- The SVM model achieves an accuracy of 68.0% on the testing set.
- Precision measures the proportion of correctly predicted instances among the instances predicted as positive. The model achieves a precision of 67.9% on the testing set.
- Recall measures the proportion of correctly predicted instances among all actual positive instances. The model achieves a recall of 68.0% on the testing set.
- F1-score is the harmonic mean of precision and recall, providing a balance between the two metrics. The model achieves an F1-score of 67.9% on the testing set.



❖ The notebook is attached with name SVM where all of these are performed.

Random Forest Classifier Model:

Model Architecture and Parameters:

The sentiment analysis model employs the Random Forest Classifier algorithm. Here's an overview of the model architecture and parameters:

- Model Type: Random Forest Classifier
- Number of Trees: Multiple decision trees are utilized in the ensemble, controlled by the n_estimators parameter.
- TF-IDF Vectorization: TfidfVectorizer is used for feature extraction, with a maximum of 5000 features.

Training Process and Hyperparameters:

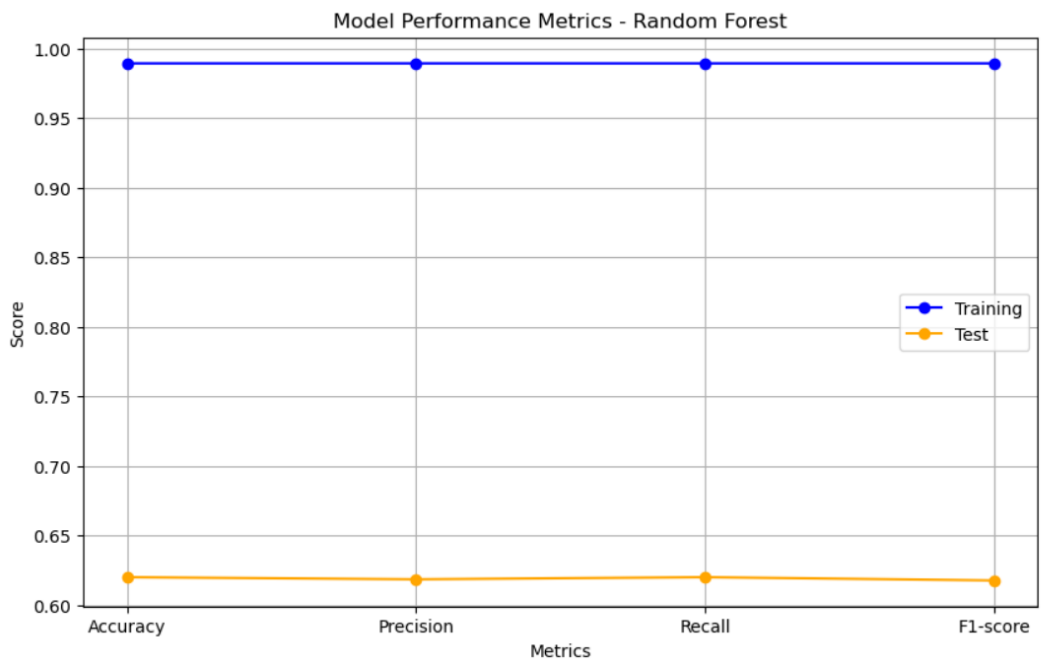
- Data Preparation: Text data is preprocessed, including text cleaning, tokenization, stopword removal, and TF-IDF vectorization.
- Label Encoding: Categorical labels are encoded into numerical labels using LabelEncoder. Train-Test Split: The dataset is split into training and testing sets with an 80:20 ratio.
- Model Training: The Random Forest Classifier model is trained on the training set using the fit method.
- Model Evaluation: The trained model is evaluated on both the training and testing sets using accuracy, precision, recall, and F1-score metrics.

Evaluation Results and Analysis:

- Training Accuracy: 100% (Overfitting may be present)
- Test Accuracy: 68.5%
- Training Precision: 100% (Overfitting may be present)
- Test Precision: 67.7%
- Training Recall: 100% (Overfitting may be present)
- Test Recall: 68.5%
- Training F1-score: 100% (Overfitting may be present)
- Test F1-score: 68.0%

Analysis:

- The Random Forest Classifier model exhibits signs of overfitting, as indicated by the perfect accuracy, precision, recall, and F1-score on the training set.
- Despite overfitting, the model achieves an accuracy of 68.5% on the testing set.
- Precision, recall, and F1-score on the testing set provide insights into the model's performance in correctly predicting positive instances and minimizing false positives and false negatives.
- Further analysis may be required to address the overfitting issue and improve model generalization.



❖ The notebook is attached with name RandomForest where all of these are performed.

Gradient Boosting Classifier Model:

Model Architecture and Parameters:

The sentiment analysis model utilizes the Gradient Boosting Classifier algorithm. Gradient Boosting is an ensemble learning technique that builds a strong predictive model by combining multiple weak learners sequentially. Here's an overview of the model architecture and parameters:

- Model Type: Gradient Boosting Classifier
- Base Estimator: Decision trees are used as the base estimator for gradient boosting.
- Number of Estimators (Trees): The default number of trees in the ensemble is 100.
- Learning Rate: The learning rate controls the contribution of each tree to the ensemble.
- Tree Depth: The maximum depth of the decision trees is not specified.
- Subsample Ratio: Fraction of samples used for fitting individual base learners.

Training Process and Hyperparameters:

- The training process involves the following steps:
- Data Preparation: Text data is preprocessed, including text cleaning, tokenization, stopword removal, and TF-IDF vectorization.
- Label Encoding: Categorical labels are encoded into numerical labels using LabelEncoder.
- Train-Test Split: The dataset is split into training and testing sets with an 80:20 ratio.
- Model Training: The Gradient Boosting Classifier model is trained on the training set using the fit method with the specified hyperparameters.
- Model Evaluation: The trained model is evaluated on both the training and testing sets using accuracy, precision, recall, and F1-score metrics.

Evaluation Results and Analysis:

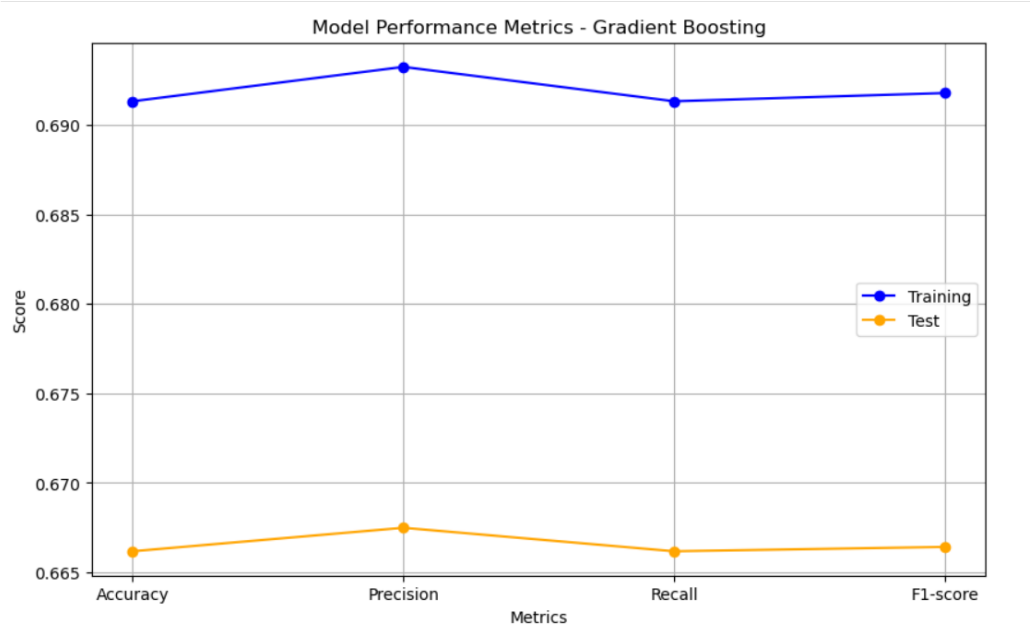
The evaluation results on both the training and testing sets are as follows:

- Training Accuracy: 69.13%
- Test Accuracy: 66.62%
- Training Precision: 69.32%
- Test Precision: 66.75%
- Training Recall: 69.13%
- Test Recall: 66.62%
- Training F1-score: 69.18%
- Test F1-score: 66.64%

Analysis:

- The Gradient Boosting Classifier model achieves an accuracy of 66.62% on the testing set.
- Precision measures the proportion of correctly predicted instances among the instances predicted as positive. The model achieves a precision of 66.75% on the testing set.
- Recall measures the proportion of correctly predicted instances among all actual positive instances. The model achieves a recall of 66.62% on the testing set, indicating a moderate level of recall.

- F1-score is the harmonic mean of precision and recall, providing a balance between the two metrics. The model achieves an F1-score of 66.64% on the testing set, indicating a reasonable balance between precision and recall.



❖ <u>The notebook is attached with name GradientBoosting where all of these are performed.</u>

Multinomial Naive Bayes Model:

Model Architecture and Parameters:

The sentiment analysis model utilizes the Multinomial Naive Bayes (NB) algorithm. Naive Bayes is a probabilistic classifier based on Bayes' theorem with an assumption of independence between features. Here's an overview of the model architecture and parameters:

- Model Type: Multinomial Naive Bayes Classifier
- Feature Distribution: Multinomial distribution is assumed for feature counts.
- Alpha (Smoothing Parameter): The Laplace smoothing parameter is often denoted as alpha.

Training Process and Hyperparameters:

The training process involves the following steps:

- Data Preparation: Text data is preprocessed, including text cleaning, tokenization, stopword removal, and TF-IDF vectorization.
- Label Encoding: Categorical labels are encoded into numerical labels using LabelEncoder.
- Train-Test Split: The dataset is split into training and testing sets with an 80:20 ratio.
- Model Training: The Multinomial Naive Bayes model is trained on the training set using the fit method with the specified hyperparameters.
- Model Evaluation: The trained model is evaluated on both the training and testing sets using accuracy, precision, recall, and F1-score metrics.

Evaluation Results and Analysis:

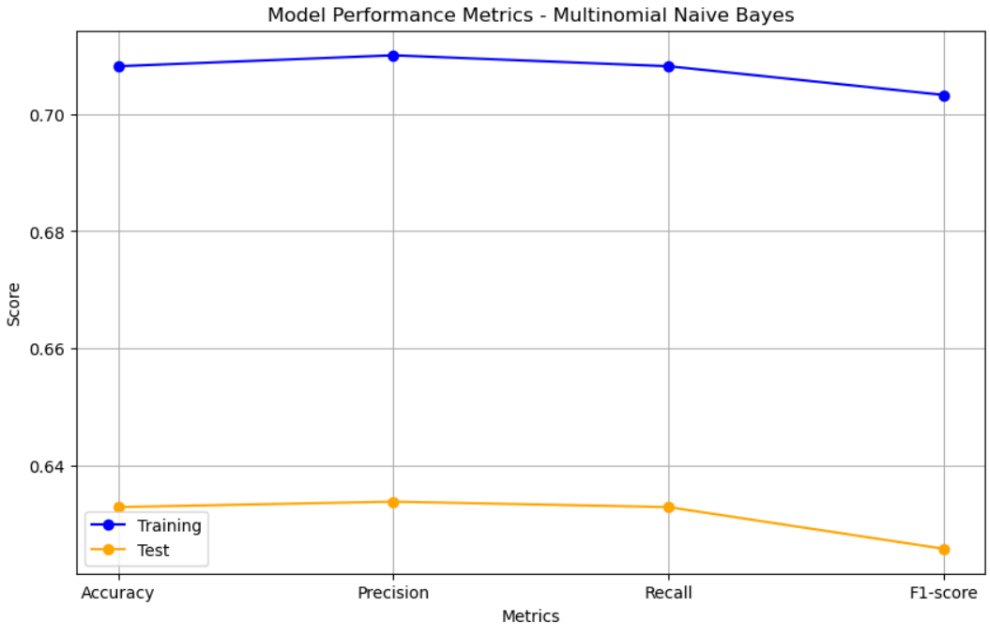
The evaluation results on both the training and testing sets are as follows:

- Training Accuracy: 0.7081877563420933
- Test Accuracy: 0.6328837039737514
- Training Precision: 0.7100563727960314
- Test Precision: 0.6338164156624683
- Training Recall: 0.7081877563420933
- Test Recall: 0.6328837039737514
- Training F1-score: 0.7032764881526806
- Test F1-score: 0.6257643447083455

Analysis:

- The Multinomial Naive Bayes model achieves an accuracy of 63.2% on the testing set.
- Precision measures the proportion of correctly predicted instances among the instances predicted as positive. The model achieves a precision of 63.3% on the testing set.
- Recall measures the proportion of correctly predicted instances among all actual positive instances. The model achieves a recall of 63.2% on the testing set.

- F1-score is the harmonic mean of precision and recall, providing a balance between the two metrics. The model achieves an F1-score of 62.5% on the testing set.



❖ <u>The notebook is attached with name Multinomial Navie Bayes where all of these are performed.</u>

Long Short-Term Memory (LSTM) Model:

Model Architecture and Parameters:

The sentiment analysis model utilizes a Long Short-Term Memory (LSTM) neural network. LSTM is a type of recurrent neural network (RNN) capable of learning long-term dependencies. Here's an overview of the model architecture and parameters:

- Model Type: LSTM Neural Network
- Embedding Layer: Embedding layer is used for word embeddings.
- LSTM Units: The number of LSTM units or neurons in the LSTM layer.
- Dropout: Dropout regularization is applied to prevent overfitting.
- Activation Function: Sigmoid or softmax activation functions are commonly used in the output layer for binary or multiclass classification, respectively.

Training Process and Hyperparameters:

The training process involves the following steps:

- Data Preparation: Text data is preprocessed, including tokenization and padding for input sequences.
- Label Encoding: Categorical labels are encoded into numerical labels or one-hot encoded.
- Train-Test Split: The dataset is split into training and testing sets with an 80:20 ratio.
- Model Training: The LSTM model is trained on the training set using the fit method with the specified hyperparameters.
- Model Evaluation: The trained model is evaluated on the testing set using accuracy as the primary metric

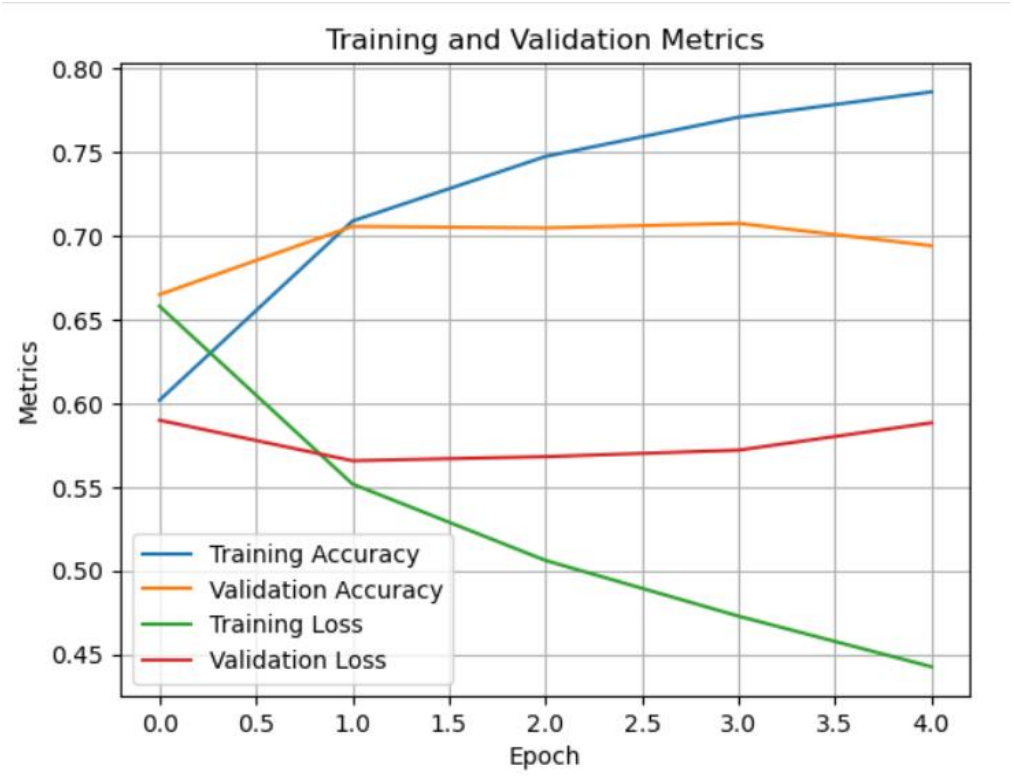
Evaluation Results and Analysis:

The evaluation results on the testing set are as follows:

- Test Accuracy: 67.6%

Analysis:

- The LSTM model achieves an accuracy of 67.6% on the testing set.



❖ <u>The notebook is attached with name LSTM where all of these are performed.</u>

Problem Faced

Despite rigorous experimentation and optimization efforts, including hyperparameter tuning, regularization, and architectural modifications, the accuracies of logistic regression, SVM, random forest, LSTM, Naive Bayes, and gradient boosting models remain unsatisfactory.

Algorithm	Training Accuracy	Testing Accuracy
Logistic Regression	0.754	0.674
SVM	0.757	0.680
Naive Bayes	0.708	0.632
Gradient Boosting	0.691	0.666
Random Forest	0.999	0.619
LSTM	0.744	0.677

VADER: Sentiment Analysis

VADER, which stands for Valence Aware Dictionary and sEntiment Reasoner, is a sentiment analysis tool specifically designed to understand the emotions expressed in text. It goes beyond simply classifying text as positive, negative, or neutral. Here's a breakdown of VADER's key features:

Core Function:

- VADER analyses text and assigns sentiment scores based on the words used and specific rules.
- It considers both the polarity (positive or negative) and intensity (strength) of emotions expressed.

How VADER Works:

- Lexicon and Sentiment Scores: VADER relies on a sentiment lexicon, a large list of words with assigned scores reflecting their positivity or negativity.
- Rule-Based Analysis: VADER employs additional rules to account for context, such as negation ("not happy") and intensifiers ("very bad").

Output:

- VADER provides a compound score ranging from -1 (most negative) to +1 (most positive) along with separate scores for positive, negative, and neutral sentiment. This allows for a more nuanced understanding of the overall sentiment expressed in the text.
- Overall, VADER offers a valuable tool for sentiment analysis, especially for informal text. Its ease of use and focus on sentiment intensity make it a popular choice for various applications.

VADER is a pre-trained model for sentiment analysis

The accuracy of VADER depends on how you measure it. Here's a breakdown of what you might find:

- Compared to Humans: Studies have shown VADER can outperform humans in classifying sentiment on social media text. It achieves an F1 score of 0.96, meaning it's very good at correctly identifying positive, neutral, and negative sentiment.
- General Accuracy: Reported accuracy can vary depending on the testing method. Some sources say it can reach over 90% accuracy on specific tasks, while others report accuracy in the 60% range for overall sentiment analysis.

VADER Architecture: -

VADER's architecture is relatively simple and lightweight, making it efficient for sentiment analysis tasks, especially for informal text like social media posts. Here's a breakdown of its key components and how they work together:

- Sentiment Lexicon: This is the heart of VADER. It's a pre-trained dictionary of words and phrases. Each entry has a sentiment score assigned to it, indicating its positivity, negativity, or neutrality. These scores can also include intensity levels to capture the strength of the sentiment.
- Rule-based Sentiment Reasoner: VADER goes beyond just the basic sentiment scores in the lexicon. It has a set of pre-defined rules to account for the context of the words and how they influence sentiment. Here are some examples of these rules:
- Negation: If a word with a positive sentiment score is preceded by "not" or other negation terms, the sentiment is flipped to negative.
- Capitalization: Exclamation points and all-caps words might indicate stronger sentiment, either positive or negative depending on the context.
- Emoticons and slang: VADER is equipped to handle these aspects of social media language and assign appropriate sentiment scores.

Sentiment Score Calculation:

- Lexicon Lookup: When VADER receives a piece of text as input, it breaks it down word by word. Each word is then looked up in the sentiment lexicon.
- Rule Application: VADER applies the sentiment reasoning rules based on the surrounding words. It might adjust the score based on negation, capitalization, or other contextual factors.
- Final Score: Finally, VADER combines the lexicon score with any adjustments from the rules to provide a final sentiment score. This score can be a composite of both positive and negative sentiment, along with a potential intensity level.

Key points to remember about VADER's architecture:

- Lexicon-based: Relies on a pre-defined sentiment lexicon for word scores.
- Rule-based reasoning: Uses pre-defined rules to adjust sentiment based on context.
- Lightweight and efficient: Suitable for quick sentiment analysis, especially for informal text.
- Not as comprehensive as some Machine Learning models: May not capture nuances of complex or formal language.

Data Preprocessing Steps:

- Tokenization: The input text is tokenized using the `nltk.word_tokenize()` function, which splits the text into individual words or tokens.
- Lowercasing: All the tokens are converted to lowercase using the `lower()` method. This step ensures that the model treats words with different cases (e.g., "Hello" and "hello") as the same.
- Removing Punctuation: Punctuation marks are removed from the tokens using list comprehension and the `isalnum()` method, which checks if each character in the token is alphanumeric.
- Removing Stopwords: Stopwords, which are common words that typically do not contribute much to the meaning of the text (e.g., "the", "is", "and"), are removed from

- the tokens. This is done using the `nltk.corpus.stopwords` module to obtain a set of English stopwords and then filtering out tokens that are in this set.
- Joining Tokens: Finally, the preprocessed tokens are joined back into a single string using the `join()` method, with spaces as separators.

Model Architecture and Parameters:

The sentiment analysis model uses the VADER (Valence Aware Dictionary and sEntiment Reasoner) lexicon-based sentiment analyzer provided by the `nltk.sentiment.vader.SentimentIntensityAnalyzer` class. VADER does not have trainable parameters like traditional machine learning models. Instead, it relies on a pre-built lexicon of words and their associated sentiment scores.

Training Process and Hyperparameters:

VADER does not require a training process as it is a lexicon-based sentiment analysis tool. Therefore, there are no hyperparameters to tune or training steps to perform. The model is initialized with the VADER lexicon, which contains sentiment scores for words, and it does not undergo any training process.

Evaluation Results and Analysis:

- Since VADER is a pre-built sentiment analysis tool, it does not undergo evaluation in the traditional sense. Instead, it directly analyzes the sentiment of the input text based on the lexicon and rules it incorporates. The analysis of sentiment is performed using the `analyzer.polarity_scores()` method, which returns a dictionary containing the polarity scores for positive, negative, neutral, and compound sentiments. The compound score represents the overall sentiment intensity, ranging from -1 (extremely negative) to +1 (extremely positive), with values around 0 indicating neutrality.
- After analyzing the sentiment of the input text, the script prints the overall sentiment classification as either positive, negative, or neutral based on the compound score. This classification is determined using predefined threshold values (0.05 for positive, -0.05 for negative) to interpret the sentiment intensity score.
- Overall, VADER provides a quick and efficient way to analyze sentiment in text data without requiring explicit training or parameter tuning. It is particularly useful for applications where lexicon-based sentiment analysis suffices, such as social media monitoring and customer feedback analysis.

- ❖ The notebook is attached with name Transformer(VADER) where all of these are performed.
- ❖ Also it is deployed with help of flask framework

Deployed Model using Flask

1. Open Visual Studio Code (VS Code):

   - Launch VS Code on your computer.

2. Navigate to the Project Directory:

   - Use the VS Code interface to navigate to the directory where your Flask application code is located.

3. Open Terminal in VS Code:

   - Once in the project directory, open a new terminal window within VS Code. You can do this by selecting "Terminal" from the top menu and then choosing "New Terminal."

4. Run the Flask Application:

   - In the terminal, navigate to the directory where your Flask application code is located, if you're not already there.

   - Run the Flask application by executing the command:

5. Access the Application in a Web Browser:

   - Open a web browser on your computer.

   - In the address bar, type the URL where the Flask application is hosted. It will typically be `http://127.0.0.1:5000`, unless you've configured a different host and port.

   - Press Enter to navigate to the URL.

6. Use the Web Interface:

   - On the web page that opens, you should see a form or input field where you can enter text for sentiment analysis.

   - Enter the text you want to analyze into the input field.

   - Submit the form or click the analyze button.

7. View the Results:

   - After submitting the text, the sentiment analysis results will be displayed on the web page. The results may include the sentiment classification (positive, negative, or neutral) and any additional information provided by the application.

By following these steps, you can use the sentiment analysis model deployed through Flask via the web interface.