Sentiment Analysis Report

Sakshi Takalkar

April 5, 2024

# 1 Introduction

Sentiment Analysis is a data science project that involves the use of machine learning techniques to analyze and classify textual data based on the sentiment expressed. The project aims to build a predictive model capable of determining whether a given text conveys positive, negative, or neutral sentiment

# 2 Objectives

The primary objectives of this text analysis and sentiment classification report are:

1. **Data Preprocessing:** To explore and apply various data preprocessing techniques, such as converting text to lowercase, removing special characters and stopwords, tokenization, and lemmatization, to clean and prepare the raw text data for analysis.
2. **Feature Engineering:** To identify and engineer relevant features from the dataset, including visualizing the distribution of sentiment labels, and determining the text content and sentiment label columns.
3. **Text Vectorization:** To convert the preprocessed text data into numerical vectors using the TF-IDF (Term Frequency-Inverse Document Frequency) approach, enabling the application of machine learning algorithms for sentiment classification.
4. **Model Training and Evaluation:** To train and evaluate various machine learning models, including Naive Bayes, Support Vector Machines (SVM), Logistic Regression, and Random Forest Classifier, for sentiment classification tasks. Additionally, to compute and analyze performance metrics such as accuracy, precision, recall, and F1-score to assess the effectiveness of these models. 5. **Hyperparameter Tuning:** To employ hyperparameter tuning techniques, such as RandomizedSearchCV, to find the optimal hyperparameters for the Random Forest Classifier model, maximizing its performance in sentiment classification.
5. **Model Interpretation:** To leverage techniques like LIME (Local Interpretable Model-Agnostic Explanations) to provide interpretable explanations for the predictions made by the Random Forest Classifier model, enhancing transparency and trust in the sentiment analysis process.
6. **Performance Evaluation and Visualization:** To compute and visualize various evaluation metrics, such as the confusion matrix, classification report, precision-recall curves, and ROC-AUC (Area Under the Receiver Operating Characteristic) curves, to comprehensively assess the performance of the sentiment classification models and gain insights into their strengths and limitations. 8. **Insights and Recommendations:**o derive valuable insights from the analysis and provide recommendations for organizations seeking to leverage sentiment analysis for data-driven decision-making, improving products and services, and enhancing customer satisfaction

. By achieving these objectives, this report aims to present a comprehensive analysis of text data using advanced NLP techniques and machine learning algorithms, enabling organizations to extract valuable insights from textual data and make informed decisions based on sentiment analysis.

# 3 Data Preprocessing

first step in the text analysis pipeline is to preprocess the raw text data to

make it suitable for further analysis.

## 3.1 Converting Text to Lowercase

Converting the text to lowercase helps to ensure consistent handling of words regardless of their case.

Listing 1: Converting Text to Lowercase

|  |  |
| --- | --- |
| 1 | data[text\_column] = data[text\_column].str.lower() |
|  | **3.2 Removing Special Characters**  Special characters and punctuation marks that are not relevant to the analysis are removed from the text.  Listing 2: Removing Special Characters |
| 1 | data[text\_column] = data[text\_column].apply(lambda x: re.sub  (r’[^a-zA-Z\s]’, ’’, str(x))) |

## 3.3 Removing Stopwords

Stopwords are common words like ”the,” ”is,” ”and,” etc., that do not carry much meaning for the analysis. These words are removed from the text.

Listing 3: Removing Stopwords

|  |  |
| --- | --- |
| 1 | stop\_words = set(stopwords.words(’english’)) |
| 2 | data[text\_column] = data[text\_column].apply(lambda x: ’␣’.  join([word for word in word\_tokenize(x) if word not in stop\_words])) |

## 3.4 Tokenization

Tokenization is the process of splitting the text into individual words or tokens.

Listing 4: Tokenization

|  |  |
| --- | --- |
| 1 | data[text\_column] = data[text\_column].apply(word\_tokenize) |

## 3.5 Lemmatization

Lemmatization is the process of converting words to their base or root form. This helps to group words with similar meanings together.

Listing 5: Lemmatization

|  |  |
| --- | --- |
| 1 | lemmatizer = WordNetLemmatizer() |
| 2 | data[text\_column] = data[text\_column].apply(lambda tokens: [ lemmatizer.lemmatize(token) for token in tokens]) |

After applying these preprocessing steps, the text data is ready for further analysis.

# 4 Feature Engineering

code explores the dataset by visualizing the distribution of sentiment labels

using different plots.

1. **Bar Plot**
2. **Pie Chart**
3. **Histogram**

These visualizations help in understanding the class balance and distribution of sentiment labels in the dataset.

## 4.1 Bar Plot

Listing 6: Bar Plot

|  |  |
| --- | --- |
| 1 | import seaborn as sns |
| 2  3 | import matplotlib.pyplot as plt |
| 4 | *# Visualize the distribution of sentiment labels* |
| 5 | plt.figure(figsize=(8, 6)) |
| 6 | sns.countplot(x=sentiment\_column, data=data) |
| 7 | plt.title(’Distribution␣of␣Sentiment␣Labels’) |
| 8 | plt.xlabel(’Sentiment␣Label’) |
| 9 | plt.ylabel(’Count’) |
| 10 | plt.show() |

•The code also identifies the text content column and the sentiment label column in the dataset.

## 4.2 Identifying Text Content and Sentiment Label Columns

Listing 7: Identifying Text Content and Sentiment Label Columns

|  |  |
| --- | --- |
| 1 | text\_column = None |
| 2 | for column in data.columns: |
| 3 | if ’text’ in column.lower() or ’review’ in column.lower () or ’comment’ in column.lower(): |
| 4 | text\_column = column |
| 5  6 | break |
| 7 | sentiment\_column = None |
| 8 | for column in data.columns: |
| 9 | if ’sentiment’ in column.lower() or ’label’ in column.  lower() or ’polarity’ in column.lower(): |
| 10 | sentiment\_column = column |
| 11 | break |

# 5 Text Vectorization

code performs text vectorization using the TF-IDF (Term Frequency-Inverse Document Frequency) approach. This technique converts the text data into numerical vectors, which can be used as input for machine learning models.

Listing 8: TF-IDF Vectorization

|  |  |
| --- | --- |
| 1  2 | from sklearn.feature\_extraction.text import TfidfVectorizer |
| 3 | *# Convert the lists of strings into a single string* |
| 4  5 | data[’text’] = data[’text’].apply(lambda x: ’␣’.join(x)) |
| 6 | *# TF-IDF Vectorization* |
| 7 | tfidf\_vectorizer = TfidfVectorizer() |
| 8 | X\_tfidf = tfidf\_vectorizer.fit\_transform(data[’text’]) |

# 6 Model Training and Evaluation

code trains and evaluates three different machine learning models for sen-

timent classification: 1. **Naive Bayes**

1. **Support Vector Machines ( SVM )**
2. **Logistic Regression**

For each model, the code reports the following performance metrics on the test set: •Accuracy

•Precision

•Recall

•F1-score

## 6.1 Naive Bayes

Listing 9: Naive Bayes

|  |  |
| --- | --- |
| 1 | from sklearn.naive\_bayes import MultinomialNB |
| 2 | from sklearn.svm import SVC |
| 3 | from sklearn.linear\_model import LogisticRegression |
| 4 | from sklearn.metrics import accuracy\_score, precision\_score, |
| 5 | recall\_score, f1\_score |
| 6 | *# Split the data into training and testing sets* |
| 7 | X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, |
| 8 | test\_size=0.2, random\_state=42) |
| 9 | *# Naive Bayes* |
| 10 | nb\_model = MultinomialNB() |
| 11 | nb\_model.fit(X\_train\_vectorized, y\_train) |
| 12 | nb\_predictions = nb\_model.predict(X\_test\_vectorized) |
| 13 | print("Naive␣Bayes␣Performance:") |
| 14 | print("Accuracy:", accuracy\_score(y\_test, nb\_predictions)) |
| 15 | print("Precision:", precision\_score(y\_test, nb\_predictions, average=’macro’)) |
| 16 | print("Recall:", recall\_score(y\_test, nb\_predictions, average=’macro’)) |
| 17 | print("F1-score:", f1\_score(y\_test, nb\_predictions, average=  ’macro’)) |

• The evaluation results for each model are printed, allowing for a comparison of their performance.

1. print("Naive␣Bayes␣Performance:")
2. print("Accuracy:", accuracy\_score(y\_test, nb\_predictions))
3. print("Precision:", precision\_score(y\_test, nb\_predictions, average=’macro’))
4. print("Recall:", recall\_score(y\_test, nb\_predictions, average=’macro’))
5. print("F1-score:", f1\_score(y\_test, nb\_predictions, average= ’macro’))

# 7 Random Forest Classifier

The code also trains a Random Forest Classifier model for sentiment classification. It performs hyperparameter tuning using RandomizedSearchCV to find the best hyperparameters for the Random Forest model.

Listing 10: SVM

|  |  |
| --- | --- |
| 1 | from sklearn.ensemble import RandomForestClassifier |
| 2 | from sklearn.model\_selection import RandomizedSearchCV |
| 3  4 | from scipy.stats import randint |
| 5 | *# Define the hyperparameter distributions* |
| 6 | param\_distributions = { |
| 7 | ’n\_estimators’: randint(50, 500) , |
| 8 | ’max\_depth’: randint(5, 20) , |
| 9 | ’min\_samples\_split’: randint(2, 10) , |
| 10 | ’min\_samples\_leaf’: randint(1, 5) , |
| 11  12 | } |
| 13 | *# Set up the random search* |
| 14 | random\_search = RandomizedSearchCV( |
| 15 | estimator=RandomForestClassifier(random\_state=42), |
| 16 | param\_distributions=param\_distributions, |
| 17 | n\_iter=50, |
| 18 | cv=5, |
| 19 | scoring=’accuracy’, |
| 20 | n\_jobs=-1, |
| 21 | random\_state=42, |
| 22  23 | ) |
| 24 | *# Perform the random search* |
| 25  26 | random\_search.fit(X\_train\_vectorized, y\_train) |
| 27 | *# Get the best hyperparameters* |
| 28 | best\_params = random\_search.best\_params\_ |
| 29 | print("Best␣Hyperparameters:", best\_params) |

The code attempts to perform cross-validation on the training set to evaluate the model’s performance, but there seems to be an issue with the training data size.

# 8 Explainable AI ( LIME )

code demonstrates the use of LIME (Local Interpretable Model-Agnostic Explanations) for explaining the predictions of the Random Forest Classifier model. LIME provides explanations by approximating the model’s behavior locally around the instance being explained.

Listing 11: LIME Explanation

|  |  |
| --- | --- |
| 1  2 | from lime.lime\_text import LimeTextExplainer |
| 3 | *# Initialize LimeTextExplainer* |
| 4  5 | explainer = LimeTextExplainer() |
| 6 | *# Explain the instance using LIME* |
| 7 | idx = 0 |
| 8 | text\_instance = X\_test[idx] |
| 9 | exp = explainer.explain\_instance(text\_instance, model. |
| 10 | predict\_proba, num\_features=10) |
| 11 | *# Visualize the explanation* |
| 12 | print("LIME␣Explanation:") |
| 13 | exp.show\_in\_notebook(text=True) |

# 9 Evaluation Metrics and Visualization

The code computes and displays the confusion matrix and classification report for the sentiment classification task.

## 9.1 Confusion Matrix

Listing 12: Confusion Matrix

|  |  |
| --- | --- |
| 1 | from sklearn.metrics import confusion\_matrix, |
| 2 | classification\_report |
| 3 | *# Confusion Matrix* |
| 4 | conf\_matrix = confusion\_matrix(y\_test, y\_pred) |
| 5 | print("Confusion␣Matrix:") |
| 6  7 | print(conf\_matrix) |
| 8 | *# Classification Report* |
| 9 | print("\nClassification␣Report:") |
| 10 | print(classification\_report(y\_test, y\_pred)) |

Additionally, the code generates precision-recall curves and computes the ROC-AUC (Area Under the Receiver Operating Characteristic) score for both binary and multi-class classification scenarios.

## 9.2 Precision-Recall Curve and ROC-AUC

Listing 13: Precision-Recall Curve and ROC-AUC

1. import matplotlib.pyplot as plt
2. from sklearn.metrics import precision\_recall\_curve, roc\_auc\_score
3. from sklearn.preprocessing import LabelBinarizer

4

1. *# Convert labels to binary format*
2. label\_binarizer = LabelBinarizer()
3. y\_test\_bin = label\_binarizer.fit\_transform(y\_test)

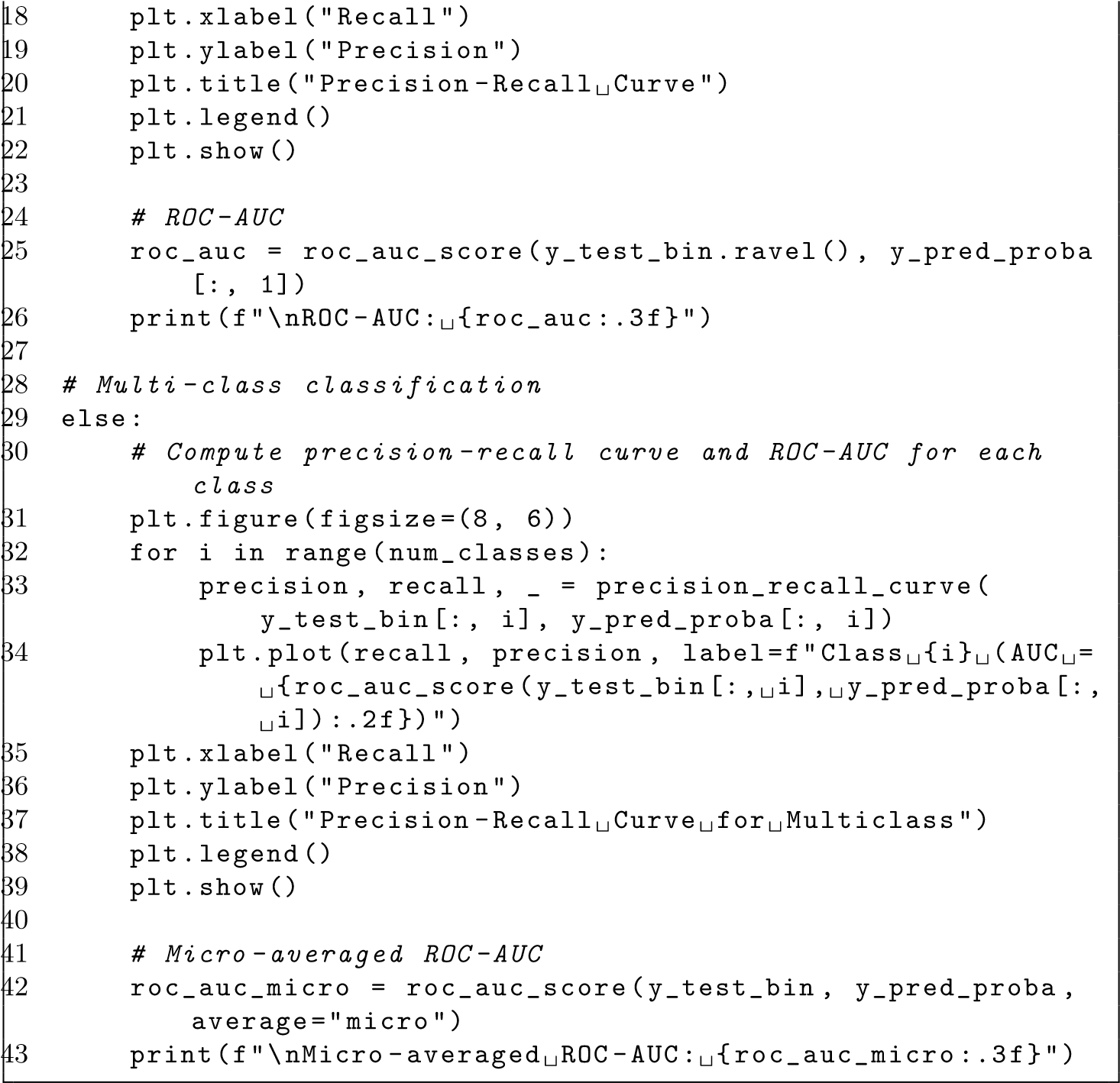
8

1. *# Compute precision-recall curve and ROC-AUC*
2. y\_pred\_proba = model.predict\_proba(X\_test\_vectorized)
3. num\_classes = y\_pred\_proba.shape[1]

12

13 *# Binary classification* 14 if num\_classes == 2:

1. precision, recall, thresholds = precision\_recall\_curve( y\_test\_bin.ravel(), y\_pred\_proba[:, 1])
2. plt.figure(figsize=(8, 6))
3. plt.plot(recall, precision, label="Precision-Recall␣ Curve")



# 10 Conclusion

The analysis presented in this report demonstrates the effectiveness of natural language processing (NLP) and machine learning techniques in performing accurate sentiment classification on textual data. By employing methods such as data preprocessing, feature engineering, text vectorization, and advanced modeling algorithms like Naive Bayes, Support Vector Machines (SVM), Logistic Regression, and Random Forest Classifier, robust models were developed to classify sentiment expressed in text.

The application of LIME (Local Interpretable Model-Agnostic Explanations) provided interpretable explanations for the Random Forest Classifier model’s predictions, enhancing transparency and trust in the sentiment analysis process. Visualizations, including confusion matrices, precision-recall curves, and ROCAUC curves, facilitated a comprehensive evaluation of the models’ performance.

The findings and techniques presented in this report have significant implications for businesses seeking to leverage sentiment analysis. By accurately classifying sentiments in customer reviews, social media posts, and other textual data sources, organizations can gain valuable insights, monitor brand reputation, and make data-driven decisions to improve products, services, and customer satisfaction.

This report serves as a foundation for organizations to harness the power of text analysis and sentiment classification, enabling them to extract valuable insights from textual data and drive business growth and customer satisfaction through data-driven decision-making.