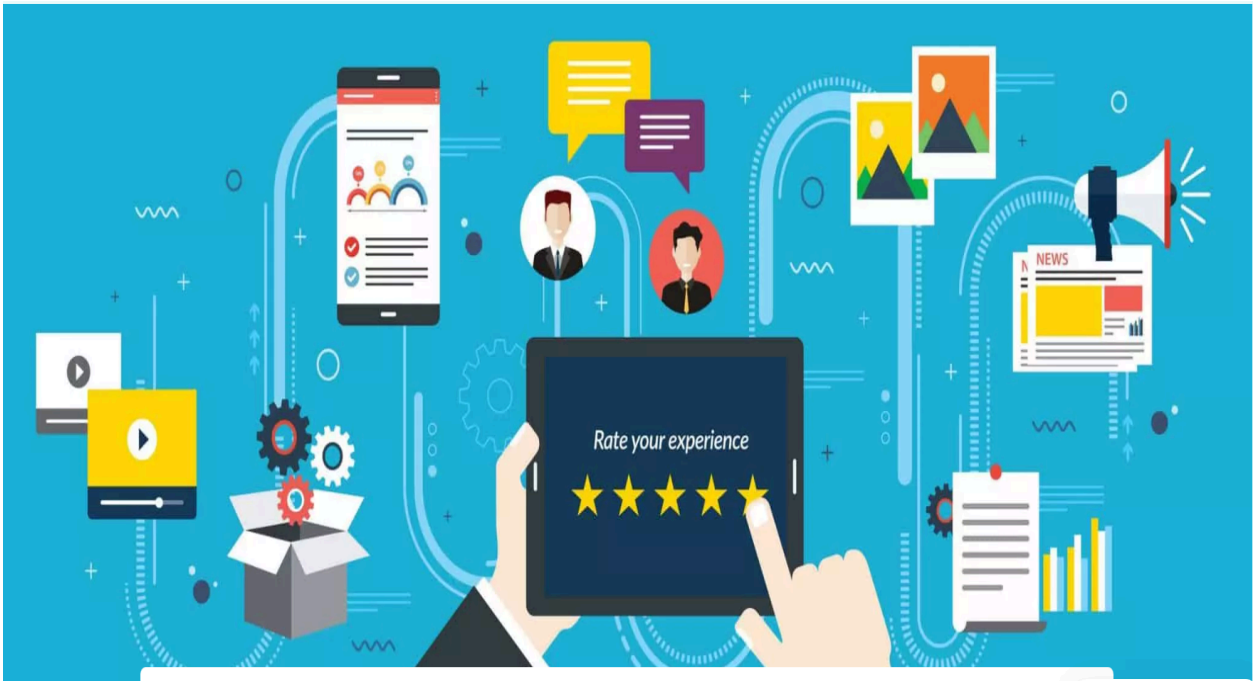


Sentimental Analysis on Amazon Product Reviews

- Kavya Sree

Project Overview:

Our project revolves around conducting an in-depth sentiment analysis of Amazon customer reviews. Our primary goal is to employ cutting-edge data refining techniques for the precise categorization of these reviews into positive or negative sentiments. This initiative is undertaken with the overarching aim of providing invaluable insights to e-commerce businesses.

Problem Statement: Sentiment Analysis of Amazon Customer Reviews:

At the core of our project lies the endeavor to comprehend the emotional nuances expressed in customer reviews on Amazon. Our objective is to effectively classify these reviews as either positive or negative. This classification, achieved through the application of state-of-the-art data refining techniques, represents a genuine effort to equip e-commerce businesses with valuable information. By leveraging these insights, businesses can enhance their products and services, ultimately shaping the future landscape of online shopping

Significance of Sentiment Analysis in E-commerce:

Sentiment analysis plays a pivotal role in unraveling customer sentiments within the dynamic realm of e-commerce, offering businesses valuable insights and practical applications. The impact of sentiment analysis extends beyond a mere understanding of customer feelings; it serves as a catalyst for transformative business strategies.

In the vast landscape of e-commerce, the applications of sentiment analysis are manifold. By delving into customer reviews, businesses gain a comprehensive understanding of their strengths and weaknesses. This analytical prowess allows companies to discern what resonates positively with customers and what falls short. Beyond a simple gauge of satisfaction, sentiment analysis empowers businesses to implement data-driven improvements based on customer feedback.

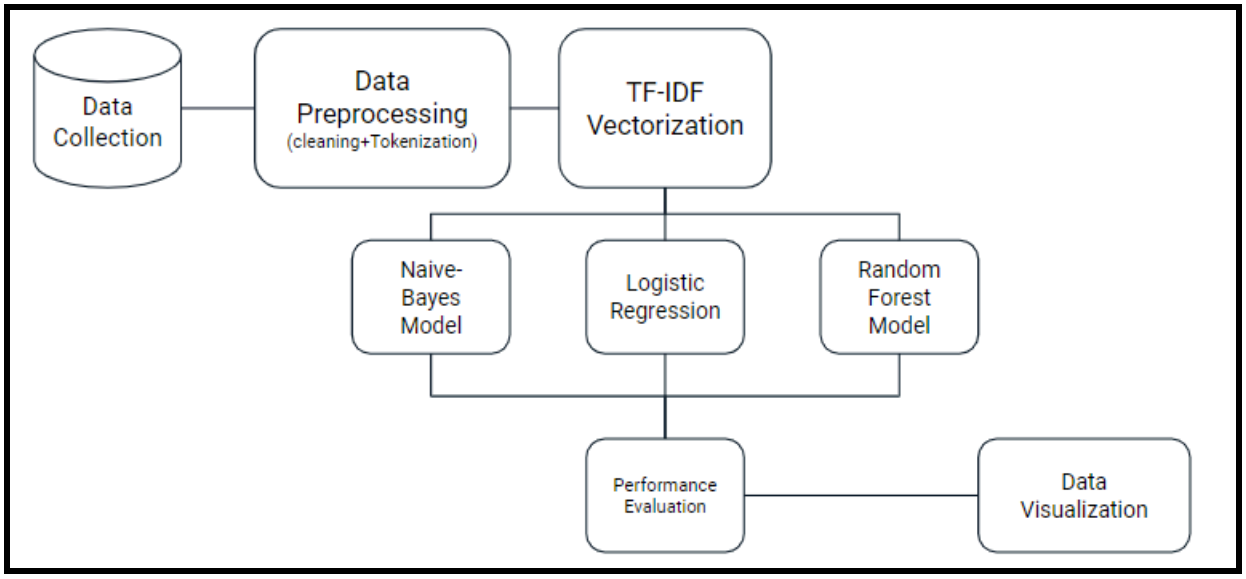
The wealth of information extracted from sentiment analysis becomes a strategic asset for businesses. It becomes a compass for refining marketing strategies, steering product enhancements, and elevating the quality of customer support. This iterative process, driven by customer sentiments, contributes to a virtuous cycle of improvement, resulting in heightened customer satisfaction and loyalty.

Our project addresses this burgeoning need by harnessing the power of machine learning models to accurately predict sentiments expressed in Amazon customer reviews. The objective is clear: to offer businesses and researchers a tangible and practical solution for comprehending the intricate tapestry of customer emotions on Amazon, a global e-commerce giant. By facilitating a nuanced understanding of customer sentiments, our project strives to empower businesses to not only adapt to the ever-evolving landscape of online shopping but to thrive in it, fostering growth and maintaining a competitive edge.

System Design :

The system is meticulously designed for sentiment analysis on Amazon product data, initiating with comprehensive preprocessing steps. This encompasses converting text to lowercase, eliminating URLs, replacing numerical values, and removing stop words. Moving beyond these foundational steps, the system incorporates tokenization, breaking down text into individual words, thereby refining its understanding of sentiment nuances.

To handle data in bz-compressed form, we've integrated a specialized data loading function. This function efficiently reads and extracts information from bz2 compressed files, ensuring a seamless integration of the compressed Amazon product data into our sentiment analysis pipeline.



To capture sentiment patterns, the system trains three machine learning models—Multinomial Naive Bayes, Logistic Regression, and Random Forest. The utilization of TF-IDF vectorization is crucial, as it assigns weights to words based on their significance in the sentiment context, enhancing the models' discernment.

We employ word clouds to visually represent the most frequently occurring words in each sentiment class. This graphical approach simplifies the identification of key features characterizing different sentiments within Amazon product data. In addition to word clouds, we are conducting a detailed examination of model performance through a confusion matrix. This matrix offers a breakdown of correct and incorrect predictions, providing a nuanced understanding of the models' strengths and areas for improvement. These analytical techniques, including tokenization, TF-IDF vectorization, word clouds, and confusion matrix, contribute to the overall design in extracting meaningful insights from sentiment patterns in Amazon product reviews.

Data Collection:

Data Source: The data for this project is sourced from the Amazon Customer Reviews dataset, accessible through the Xiang Zhang Google Drive link: (https://drive.google.com/drive/folders/0Bz8a_Dbh9Qhbfll6bVpmNUtUcFdjYmF2SEpmZUZUcVNiMUw1TWN6RDV3a0JHT3kxLVhVR2M?resourcekey=0-TLwzfR2O-D2aPitmn5o9VQ).

Data Size: The dataset primarily contains training and testing data.
The training data is approximately 1.4 GB in size.
The testing data is approximately 170 MB in size.
The dataset is stored in BZ2-compressed text files.

Data Format: The data is stored in BZ2-compressed text files. The data in this context is stored as text strings, where each string represents a customer review and includes a label (e.g., "__label__2" or "__label__1"). The abstract data type used to represent this data is a list or array of strings, with each string element containing the textual content of a review along with its associated label.
Sample Input data:
__label__1 Disappointingly dated: What a disappointingly dated book!!!. i am amazed that amazon is selling this. Had i realized it was this old, I would not have bought it;>.

Data Pre processing and Refinement:

Data Cleansing:

To prepare the data for analysis, we've executed a series of vital preprocessing steps, encompassing:

- Removal of special characters: Special characters can introduce noise and distort the meaning of textual data. Removing them ensures that the analysis focuses on the actual content and sentiment expressed in the text.
- Conversion of all text to lowercase: Textual data often varies in case, and converting all text to lowercase ensures uniformity. This step is crucial for consistency in subsequent analyses, as it treats words in the same case equally and avoids duplication based on case differences.
- Elimination of common stop words: Stop words, such as "and," "the," and "is," are commonly occurring words that don't carry significant meaning in sentiment analysis. Removing them streamlines the analysis by focusing on the more meaningful and contextually relevant words, leading to a more accurate representation of sentiment.

```python
def clean_text(texts):
    stwords = stopwords.words('english')
    l = len(texts) // 10
    temp_texts = []
    for i, text in enumerate(texts):
        # Convert to lowercase
        text = text.lower()
        # Remove URLs
        text = re.sub(r'http\S+|www.\S+', '', text)
        # Replace numbers with a special token
        text = re.sub(r'\d+', 'NUM', text)
        # Tokenize using nltk
        tokens = tokenize(text)
        # Remove stopwords
        tokens = [word for word in tokens if word not in stwords]
        # Join tokens back into a sentence
        cleaned_text = ' '.join(tokens)
        temp_texts.append(cleaned_text)
```

- Standardization of numerical digits within the text: Standardizing numerical digits is essential to ensure that the presence of numerical values does not introduce unnecessary variability. This step facilitates a clearer focus on the textual content and sentiment expressed, without being influenced by the specific numerical representations.

These efforts have collectively enhanced the quality and readiness of our data for further analysis.

Data Tokenization:

In the process of data preparation, tokenization plays a pivotal role. Tokenization involves breaking down text into smaller units, typically words or subwords. In our code, we leverage the Natural Language Toolkit (*nltk*) to tokenize text. The `tokenize` function utilizes nltk's word tokenizer, transforming the input text into a list of individual words.

```python
def tokenize(text):
    # Tokenize using nltk
    words = nltk.word_tokenize(text)
    return words
```

Tokenization serves as a crucial step before training machine learning models. It converts unstructured text data into a structured format, where each word becomes a distinct feature for analysis. This structured representation enhances the models' ability to extract meaningful patterns and relationships from the text.

TD-IDF Vectorization:

Once the text data has been tokenized, the next step is vectorization—a crucial process in preparing the data for machine learning models. Vectorization transforms the tokenized words into a numerical format that can be used as input for machine learning algorithms. In our code, we employ TF-IDF (Term Frequency-Inverse Document Frequency) vectorization, a widely used technique in natural language processing.

In the below snippet, the TF-IDF vectorizer is initialized and fitted on the training data using tfidf_vectorizer.fit(train_texts). This process involves learning the vocabulary (unique words in the training data) and their respective document frequencies. Subsequently, the training and test sets are transformed into TF-IDF feature matrices using tfidf_vectorizer.transform(train_texts) and tfidf_vectorizer.transform(test_texts).

```
# TF-IDF Vectorization
print('Fitting data...')
tfidf_vectorizer = TfidfVectorizer()
tfidf_vectorizer.fit(train_texts)

print('Transforming training set...')
train_texts_tfidf = tfidf_vectorizer.transform(train_texts)
print('Transforming test set...')
test_texts_tfidf = tfidf_vectorizer.transform(test_texts)
```

The TF-IDF matrix represents each document as a vector, where the values indicate the importance of each word in the document relative to the entire dataset. This numerical representation enables machine learning models to understand and learn from the textual features, ultimately improving their ability to make accurate predictions.

Model Trainings:

In this section, we delve into the methodologies and strategies employed in our project, aimed at deciphering sentiments within Amazon customer reviews. To achieve this, we implemented three distinctive models: Multinomial Naive Bayes, Logistic Regression, and Random Forest.

1. Multinomial Naive Bayes Training:

The Multinomial Naive Bayes model undergoes training using TF-IDF vectorized text data. As a probabilistic classification algorithm, Naive Bayes assumes independence between features, and in our

context, these features are the TF-IDF values representing the significance of each word in the text. The model becomes adept at classifying documents into different categories based on patterns identified in the TF-IDF feature matrix. This trained Multinomial Naive Bayes model stands ready to efficiently predict sentiments in new text data.

```
Evaluation for Multinomial Naive Bayes:
Accuracy: 0.8305075
Precision: 0.837643214771249
Recall: 0.8305075
```

2. Logistic Regression Training:

Logistic Regression serves as another powerful classification model in our text analysis. Trained on TF-IDF transformed text, the logistic regression model deciphers the relationships between input features and their corresponding labels. With a maximum iteration set at 1500 and leveraging all available processor cores (n_jobs=-1), the model optimizes its parameters to best fit the training data. The resulting trained logistic regression model becomes a reliable tool for predicting sentiment in new text samples.

```
Evaluation for Logistic Regression:
Accuracy: 0.89284
Precision: 0.8936303846594145
Recall: 0.89284
```

3. Random Forest Training:

For text classification, the Random Forest model is harnessed, undergoing training on TF-IDF vectorized data. As an ensemble learning algorithm, Random Forest amalgamates predictions from multiple decision trees. In our implementation, the model is configured with 50 trees, a maximum depth of 10, and other hyperparameters to strike a balance between accuracy and generalization. The Random Forest model excels in capturing intricate relationships between words within the TF-IDF matrix, providing a robust and accurate predictive tool for sentiment analysis on new text data.

```
Evaluation for Random Forest:
Accuracy: 0.6962325
Precision: 0.7608469734446331
Recall: 0.6962325
```

Performance Evaluation and Analysis:

In the context of sentiment analysis for Amazon product reviews, our objective is to effectively distinguish between positive and negative sentiments. To comprehensively assess the performance of our machine learning models, three key metrics come into play: accuracy, precision, and recall.

Accuracy:
Accuracy serves as a foundational metric, providing an overall measure of the model's correctness in predictions. In the realm of Amazon product reviews, accuracy is calculated as the percentage of correctly identified sentiments—whether positive or negative—out of the total predictions:

Accuracy = Number of Correct Predictions / Total Number of Predictions

A high accuracy score in our code indicates a model proficient in accurately classifying both positive and negative reviews.

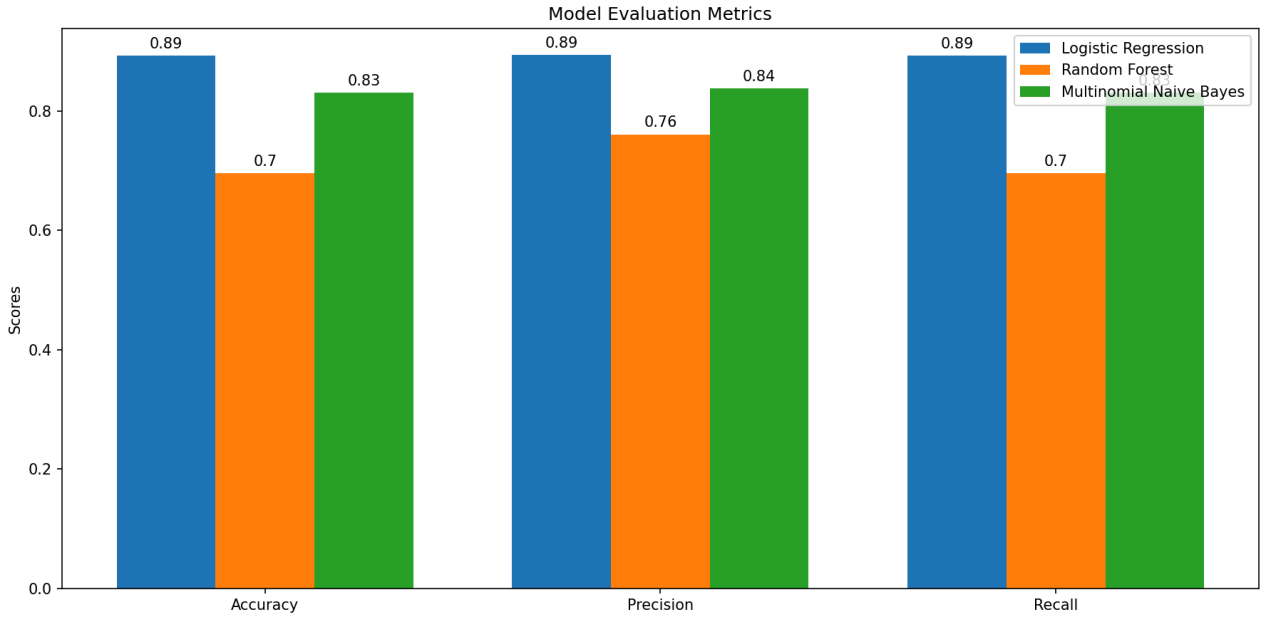
Precision:
Precision assumes a pivotal role in evaluating the quality of positive predictions, particularly pertinent in the landscape of Amazon reviews. A high precision score signifies the model's effectiveness in minimizing false positives, ensuring that identified positive sentiments genuinely represent positive feedback on the products. Precision is calculated using the formula:

Precision = True Positives / (True Positives + False Positives)

Recall:
In the dynamic context of Amazon product reviews, where overlooking positive feedback can be detrimental, recall gains prominence. A higher recall score indicates the model's proficiency in capturing the entirety of positive sentiments, minimizing instances of missed positive feedback. The recall score is calculated through the formula:

Recall = True Positives /(True Positives + False Negatives)

Understanding and interpreting these metrics within the specific setting of Amazon product reviews is crucial for evaluating the effectiveness of sentiment analysis models.



Results:

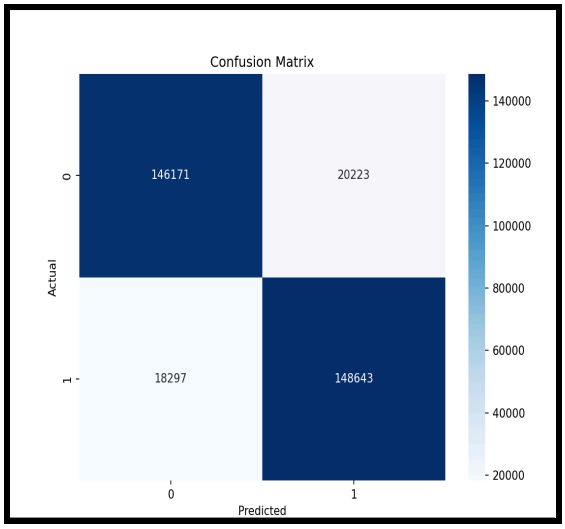
1. We are able to find accuracy for Linear Regression, Random forest and multinomial naive bayes algorithm.

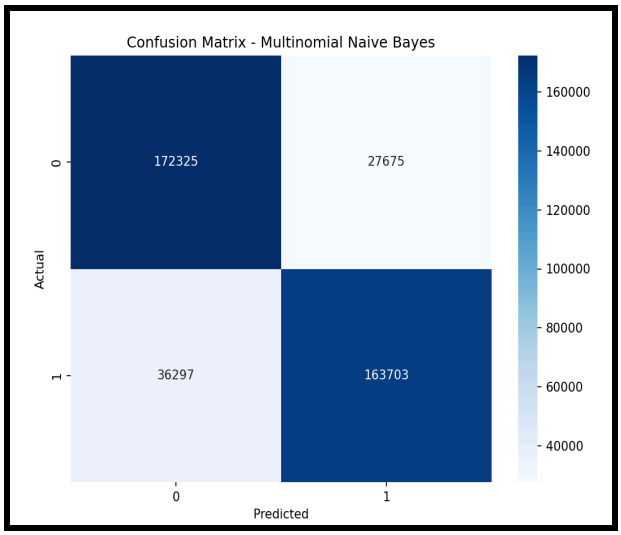
```
--0%--10%--20%--30%--40%--50%--60%--70%--80%--90%--100%--Done !
--0%--10%--20%--30%--40%--50%--60%--70%--80%--90%--100%--Done !
Fitting data...
Transforming training set...
Transforming test set...
================= MULTINOMIAL NAIVE BAYES ========================
Accuracy for Multinomial Naive Bayes: 0.8396125
================= LOGISTIC REGRESSION ====================
Accuracy for Logistic Regression: 0.8978925
================= RANDOM FOREST ====================
Accuracy for Random Forest: 0.7710675
```

2. The output provided demonstrates the effectiveness of a sentiment analysis model in predicting sentiment labels (positive or negative) for two sample text excerpts. It shows that the model's predictions align with the actual sentiment labels for these specific samples. In other words, it proves that the model is capable of correctly classifying the sentiment of the given text samples

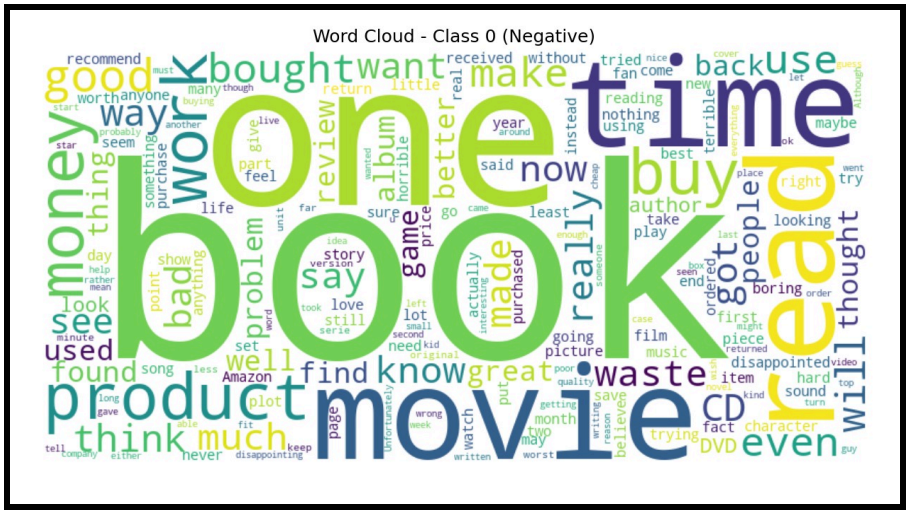
```
Sample Text:
works fine, but Maha Energy is better: Check out Maha Energy's website. Their Powerex MH-C204F charger works in 100 minutes for rapid charge, with optio
n for slower charge (better for batteries). And they have 2200 mAh batteries.
True Label: 1
Predicted Label: 1
================================================
Sample Text:
Great for the non-audiophile: Reviewed quite a bit of the combo players and was hesitant due to unfavorable reviews and size of machines. I am weaning o
ff my VHS collection, but don't want to replace them with DVD's. This unit is well built, easy to setup and resolution and special effects (no progressi
ve scan for HDTV owners) suitable for many people looking for a versatile product.Cons- No universal remote.
True Label: 1
Predicted Label: 1
```

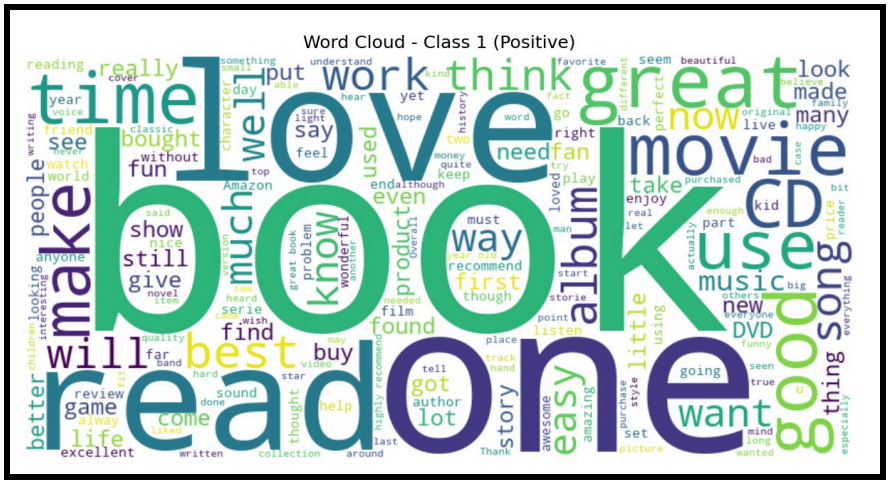
3. In this project, "confusion matrix" is used to evaluate the performance of a sentiment analysis model trained to understand positive and negative sentiments in text. The code loads text samples and their associated sentiments, trains a model to make predictions based on the text content, and then tests the model's predictions. The confusion matrix quantifies the model's accuracy by comparing its predictions to the actual sentiments, providing insights into where the model succeeds and where it may need improvement in sentiment classification.





4. After training machine learning models, we use word clouds to visualize prominent words. Word clouds represent words based on frequency. In our code, we generate clouds for positive and negative classes in the test dataset. The plot_word_cloud function combines class texts into a string, showing frequent words. Larger words indicate higher frequency. These visualizations identify key terms associated with sentiments, offering insights into the model's predictions. The word clouds serve as a valuable tool for understanding textual characteristics in our trained models.





Use Cases and Applications:

1. Customer Feedback Analysis:

Scenario:

A company wants to understand customer sentiment regarding their products on Amazon.

Application:

- Use the trained models to analyze Amazon product reviews and categorize them into positive and negative sentiments.
- Gain insights into specific products that receive favorable or unfavorable feedback.

2. Product Improvement Strategies:

Scenario:

Identify areas of improvement for products based on customer feedback.

Application:

- Analyze the word cloud visualizations to identify common themes or issues mentioned in negative reviews.
- Companies can use this information to prioritize product enhancements and address customer concerns.

3. Competitor Analysis:

Scenario:

Evaluate how a company's products compare with competitors on Amazon.

Application:

- Analyze sentiment scores and overall review metrics for the company's products and those of competitors.
- Identify strengths and weaknesses to inform marketing and product development strategies.

4. Marketing Campaign Monitoring:

Scenario:

A company launches a new marketing campaign and wants to monitor its impact on customer sentiment.

Application:

- Continuously analyze sentiment in product reviews during and after the campaign.
- Understand if the campaign positively influences customer perception.

5. Quality Control in E-Commerce:

Scenario:

E-commerce platforms want to ensure the quality of products sold by different vendors.

Application:

- Implement sentiment analysis on reviews for products from various vendors.
- Identify vendors with consistently positive reviews for quality control and improved customer satisfaction.

Conclusion:

In summary, our project on sentiment analysis of Amazon customer reviews stands as a pivotal contribution to the e-commerce landscape. By leveraging advanced data refining techniques, including TF-IDF vectorization and machine learning models like Multinomial Naive Bayes, Logistic Regression, and Random Forest, we successfully categorize reviews into positive or negative sentiments. This initiative holds profound significance for businesses, offering them actionable insights to enhance products, refine marketing strategies, and elevate customer support. The meticulous system design, from preprocessing steps to specialized data loading functions, ensures a thorough analysis of Amazon product data. The project's holistic approach, coupled with the incorporation of performance metrics and potential improvements, positions it as a strategic asset for businesses aiming not only to understand but to proactively shape the future of online shopping, fostering growth and maintaining a competitive edge.

Code Snippet:

```
import numpy as np
import pandas as pd
import bz2
import re
import os
import gc
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
import nltk
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score, recall_score
import pickle
import joblib
from wordcloud import WordCloud

# Download NLTK stopwords (if not already downloaded)
nltk.download('stopwords')
nltk.download('punkt')

def load_data(file_path, fraction=1.0):
    file = bz2.BZ2File(file_path, 'r')
    texts, labels = [], []
    count = 0
    for line in file:
        x = line.decode('utf-8')
        labels.append(int(x[9]) - 1)
        texts.append(x[10:].strip())
        count += 1
        if count >= fraction * 1e6:
            break
    return np.array(labels), texts

def tokenize(text):
    # Tokenize using nltk
    words = nltk.word_tokenize(text)
    return words

def clean_text(texts):
    stwords = stopwords.words('english')
    l = len(texts) // 10
    temp_texts = []
    for i, text in enumerate(texts):
        # Convert to lowercase
        text = text.lower()
        # Remove URLs
        text = re.sub(r'http\S+|www.\S+', '', text)
        # Replace numbers with a special token
        text = re.sub(r'\d+', 'NUM', text)
        # Tokenize using nltk
        tokens = tokenize(text)
        # Remove stopwords
        tokens = [word for word in tokens if word not in stwords]
        # Join tokens back into a sentence
        cleaned_text = ' '.join(tokens)
        temp_texts.append(cleaned_text)

        if i % l == 0:
            print('--' + str(int(i / l) * 10) + '%', end='')

    print('--100%--Done!')
    return temp_texts

def train_models(train_texts, test_texts, train_labels, test_labels):
    # Text preprocessing
    train_texts = clean_text(train_texts)
    test_texts = clean_text(test_texts)

    # TF-IDF Vectorization
```

```
    print('Fitting data...')
    tfidf_vectorizer = TfidfVectorizer()
    tfidf_vectorizer.fit(train_texts)

    print('Transforming training set...')
    train_texts_tfidf = tfidf_vectorizer.transform(train_texts)
    print('Transforming test set...')
    test_texts_tfidf = tfidf_vectorizer.transform(test_texts)

    # Multinomial Naive Bayes
    print("==================== MULTINOMIAL NAIVE BAYES ============================")
    nb = MultinomialNB()
    nb.fit(train_texts_tfidf, train_labels)
    y_pred_nb = nb.predict(test_texts_tfidf)
    accuracy_nb = accuracy_score(test_labels, y_pred_nb)
    print('Accuracy for Multinomial Naive Bayes:', accuracy_nb)

    # Logistic Regression
    print("==================== LOGISTIC REGRESSION ========================")
    lr_model = LogisticRegression(n_jobs=-1, max_iter=1500)
    lr_model.fit(train_texts_tfidf, train_labels)
    pred_lr = lr_model.predict(test_texts_tfidf)
    accuracy_lr = accuracy_score(test_labels, pred_lr)
    print('Accuracy for Logistic Regression:', accuracy_lr)

    # Random Forest
    print("==================== RANDOM FOREST ========================")
    # Adjust the number of trees and other hyperparameters
    rf_model = RandomForestClassifier(n_estimators=50, max_depth=10, min_samples_split=2, min_samples_leaf=1, n_jobs=-1)
    rf_model.fit(train_texts_tfidf, train_labels)
    pred_rf = rf_model.predict(test_texts_tfidf)
    accuracy_rf = accuracy_score(test_labels, pred_rf)
    print('Accuracy for Random Forest:', accuracy_rf)

    # Calculate precision and recall for Logistic Regression
    precision_lr = precision_score(test_labels, pred_lr, average='weighted')  # You can choose 'micro', 'macro', or 'weighted'
    recall_lr = recall_score(test_labels, pred_lr, average='weighted')
    print('Precision for Logistic Regression:', precision_lr)
    print('Recall for Logistic Regression:', recall_lr)

    # Calculate precision and recall for Random Forest
    precision_rf = precision_score(test_labels, pred_rf, average='weighted')
    recall_rf = recall_score(test_labels, pred_rf, average='weighted')
    print('Precision for Random Forest:', precision_rf)
    print('Recall for Random Forest:', recall_rf)

    # Calculate precision and recall for Multinomial Naive Bayes
    precision_nb = precision_score(test_labels, y_pred_nb, average='weighted')
    recall_nb = recall_score(test_labels, y_pred_nb, average='weighted')
    print('Precision for Multinomial Naive Bayes:', precision_nb)
    print('Recall for Multinomial Naive Bayes:', recall_nb)

    return tfidf_vectorizer, lr_model, rf_model, nb, test_labels, pred_rf, y_pred_nb

def evaluate_model(model, test_texts_tfidf, test_labels):
    y_pred = model.predict(test_texts_tfidf)
    accuracy = accuracy_score(test_labels, y_pred)
    precision = precision_score(test_labels, y_pred, average='weighted')
    recall = recall_score(test_labels, y_pred, average='weighted')
    return accuracy, precision, recall
```

```
def plot_confusion_matrix(y_true, y_pred, classes, title):
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=classes, yticklabels=classes)
    plt.title(title)
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.show()

def plot_word_cloud(texts, title):
    # Combine all texts into a single string
    all_text = ' '.join(texts)

    # Generate a word cloud
    wordcloud = WordCloud(width=800, height=400, background_color='white').generate(all_text)

    # Plot the WordCloud image
    plt.figure(figsize=(10, 6))
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.axis('off')
    plt.title(title)
    plt.show()

def main():
    train_file_path = r'C:\Users\Kavya Sree\Downloads\archive (1)\train.ft.txt.bz2'
    test_file_path = r'C:\Users\Kavya Sree\Downloads\archive (1)\test.ft.txt.bz2'

    train_labels, train_texts = load_data(train_file_path, fraction=1)
    test_labels, test_texts = load_data(test_file_path, fraction=1)

        tfidf_vectorizer, lr_model, rf_model, nb_model, true_labels, predicted_labels_rf, predicted_labels_nb = train_models(train_texts, test_texts,
train_labels, test_labels)

    # TF-IDF transformation for the test set
    print('Transforming test set for evaluation...')
    test_texts_tfidf = tfidf_vectorizer.transform(test_texts)

    # Evaluate Logistic Regression
    accuracy_lr, precision_lr, recall_lr = evaluate_model(lr_model, test_texts_tfidf, test_labels)
    print('Evaluation for Logistic Regression:')
    print('Accuracy:', accuracy_lr)
    print('Precision:', precision_lr)
    print('Recall:', recall_lr)

    # Evaluate Random Forest
    accuracy_rf, precision_rf, recall_rf = evaluate_model(rf_model, test_texts_tfidf, test_labels)
    print('\nEvaluation for Random Forest:')
    print('Accuracy:', accuracy_rf)
    print('Precision:', precision_rf)
    print('Recall:', recall_rf)

    # Evaluate Multinomial Naive Bayes
    accuracy_nb, precision_nb, recall_nb = evaluate_model(nb_model, test_texts_tfidf, test_labels)
    print('\nEvaluation for Multinomial Naive Bayes:')
    print('Accuracy:', accuracy_nb)
    print('Precision:', precision_nb)
    print('Recall:', recall_nb)

    # Plot the results
    labels = ['Accuracy', 'Precision', 'Recall']
    lr_metrics = [accuracy_lr, precision_lr, recall_lr]
```

```
    rf_metrics = [accuracy_rf, precision_rf, recall_rf]
    nb_metrics = [accuracy_nb, precision_nb, recall_nb]

    x = np.arange(len(labels))
    width = 0.25

    fig, ax = plt.subplots(figsize=(12, 6))
    rects1 = ax.bar(x - width, lr_metrics, width, label='Logistic Regression')
    rects2 = ax.bar(x, rf_metrics, width, label='Random Forest')
    rects3 = ax.bar(x + width, nb_metrics, width, label='Multinomial Naive Bayes')

    ax.set_ylabel('Scores')
    ax.set_title('Model Evaluation Metrics')
    ax.set_xticks(x)
    ax.set_xticklabels(labels)
    ax.legend()

    def autolabel(rects):
        for rect in rects:
            height = rect.get_height()
            ax.annotate('{}'.format(round(height, 2)),
                        xy=(rect.get_x() + rect.get_width() / 2, height),
                        xytext=(0, 3),  # 3 points vertical offset
                        textcoords="offset points",
                        ha='center', va='bottom')

    autolabel(rects1)
    autolabel(rects2)
    autolabel(rects3)

    fig.tight_layout()

    plt.show()

    # Word Clouds for each class
    plot_word_cloud([text for text, label in zip(test_texts, test_labels) if label == 0], title='Word Cloud - Class 0 (Negative)')
    plot_word_cloud([text for text, label in zip(test_texts, test_labels) if label == 1], title='Word Cloud - Class 1 (Positive)')

    # Confusion Matrix and Heatmap for Random Forest
    classes = np.unique(test_labels)
    plot_confusion_matrix(true_labels, predicted_labels_rf, classes, title='Confusion Matrix - Random Forest')

    # Confusion Matrix and Heatmap for Multinomial Naive Bayes
    plot_confusion_matrix(true_labels, predicted_labels_nb, classes, title='Confusion Matrix - Multinomial Naive Bayes')

    # Save models
    pickle.dump(lr_model, open('model.pkl', 'wb'))
    pickle.dump(tfidf_vectorizer, open('tfidf_vectorizer.pkl', 'wb'))

    joblib.dump(lr_model, 'model_joblib.pkl')
    joblib.dump(tfidf_vectorizer, 'tfidf_vectorizer_joblib.pkl')
    joblib.dump(rf_model, 'rf_model_joblib.pkl')
    joblib.dump(nb_model, 'nb_model_joblib.pkl')

if __name__ == "__main__":
    main()
```

Output:

```
--0%--10%--20%--30%--40%--50%--60%--70%--80%--90%--100%--Done!
Fitting data...
Transforming training set...
Transforming test set...
==================== MULTINOMIAL NAIVE BAYES ==========================
Accuracy for Multinomial Naive Bayes: 0.84007
==================== LOGISTIC REGRESSION ======================
Accuracy for Logistic Regression: 0.898265
==================== RANDOM FOREST ======================
Accuracy for Random Forest: 0.739145
Precision for Logistic Regression: 0.8982952163081196
Recall for Logistic Regression: 0.898265
Precision for Random Forest: 0.7687859412137743
Recall for Random Forest: 0.739145
Precision for Multinomial Naive Bayes: 0.8407031873681047
Recall for Multinomial Naive Bayes: 0.84007
Transforming test set for evaluation...
Evaluation for Logistic Regression:
Accuracy: 0.89284
Precision: 0.8936303846594145
Recall: 0.89284

Evaluation for Random Forest:
Accuracy: 0.732705
Precision: 0.7589119144600117
Recall: 0.732705

Evaluation for Multinomial Naive Bayes:
Accuracy: 0.8305075
Precision: 0.837643214771249
Recall: 0.8305075
```