Predictive Analytics System for Amazon Review Sentiment Analysis

Index

Introduction

This report details the creation, evaluation, and deployment of a predictive analytics system designed to analyze and forecast sentiment trends in Amazon product reviews. By leveraging data mining, machine learning, and scalable computing environments, this project showcases capabilities in handling large datasets, preprocessing data, building machine learning models, deploying applications, and visualizing results effectively. Throughout this report, we will walk through the steps involved in preprocessing the data, training and evaluating machine learning models, optimizing hyperparameters, making real-time predictions, visualizing results, and finally deploying the application. Additionally, we will analyze challenges faced and solutions implemented, with a special focus on the comparison between Logistic Regression and Random Forest models.

Overview

The system uses a Logistic Regression model and a Random Forest model to classify Amazon product reviews as positive or negative. This binary classification task is crucial for understanding customer sentiments and improving business strategies and customer satisfaction.

Steps to Build the Predictive Analytics System

2. Data Preprocessing

Objective: Clean and prepare the data for model training.

1)Loading the Dataset:

The dataset was sourced from the Amazon Fine Food Reviews dataset. It was loaded using pandas.

The dataset included review text, scores, and other metadata.

2)Handling Missing Data:

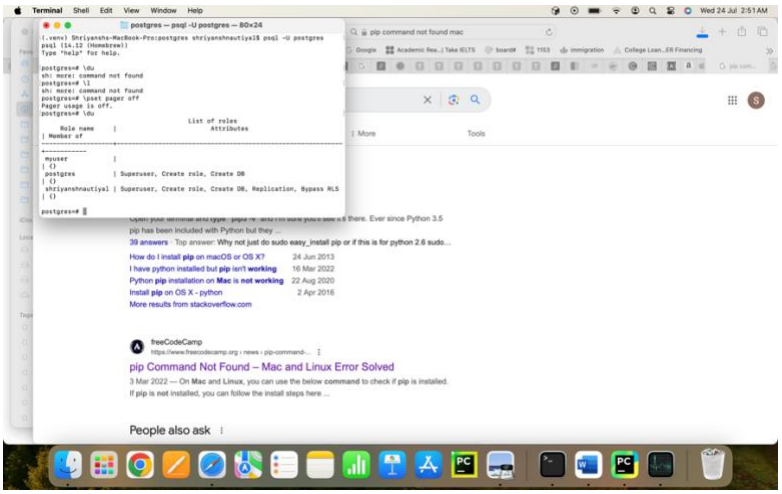
Missing values were filled using forward fill imputation to ensure no null values that could disrupt model training.

```
1 usage   Shriyansh Nautiyal
def preprocess_data(filepath):
    data = pd.read_csv(filepath)
    data.ffill(inplace=True)  # Use forward fill to handle missing values
    data['Sentiment'] = data['Score'].apply(lambda x: 1 if x > 3 else 0)  # Binary sentiment
    return data

data = preprocess_data('amazon_fine_food_reviews.csv')
# Data preprocessing check
print("Data preprocessing completed.")
```

3)Storing Preprocessed Data:

The cleaned data was stored in a PostgreSQL database to facilitate easy access and querying.



3. Building and Training Machine Learning Models

Objective: Train models to classify review sentiments.

1)Splitting Data:

The dataset was split into training and validation sets to evaluate model performance effectively.

2)Vectorizing Text Data:

Text reviews were transformed into numerical feature vectors using TF-IDF (Term Frequency-Inverse Document Frequency) vectorization.

```
# Split data
X = data['Text']
y = data['Sentiment']
X_train, X_val, y_train, y_val = train_test_split( *arrays: X, y, test_size=0.2, random_state=42)

# Vectorize text
vectorizer = TfidfVectorizer(stop_words='english', max_features=10000)
X_train_tfidf = vectorizer.fit_transform(X_train)
X_val_tfidf = vectorizer.transform(X_val)
```

3)Training Models:

Two models, Logistic Regression and Random Forest, were trained on the vectorized text data.

Random Forest was chosen due to its best feature search in randomness (and that it can be used for regression as well).

Logistic Regression was chosen for its simplicity and effectiveness in binary classification

4)Evaluating Models:

Model performance was evaluated using metrics such as accuracy, precision, recall, and F1-score.These metrics provide a comprehensive understanding of the model's effectiveness in classifying sentiments.

```
2 usages  new *
def evaluate_model(model, X_val, y_val):
    predictions = model.predict(X_val)
    accuracy = accuracy_score(y_val, predictions)
    precision = precision_score(y_val, predictions)
    recall = recall_score(y_val, predictions)
    f1 = f1_score(y_val, predictions)
    return accuracy, precision, recall, f1

# Logistic Regression
logistic_model = LogisticRegression()
logistic_model.fit(X_train_tfidf, y_train)
logistic_metrics = evaluate_model(logistic_model, X_val_tfidf)

# Random Forest
random_forest_model = RandomForestClassifier(n_estimators=100)
random_forest_model.fit(X_train_tfidf, y_train)
random_forest_metrics = evaluate_model(random_forest_model, X_val_tfidf)

print(f'Logistic Regression Metrics: {logistic_metrics}')
print(f'Random Forest Metrics: {random_forest_metrics}')
```

4. Hyperparameter Tuning and Optimization

Objective: Optimize models for better performance.

1)Defining Parameter Grid:

A parameter grid was defined to explore different combinations of hyperparameters.

2)Evaluating the Best Model:

The best model from the grid search was evaluated to ensure it met performance expectations.

5. Real-Time Prediction and Anomaly Detection

Objective: Enable real-time sentiment prediction and detect anomalies in predictions.

1)Making Predictions:

The trained model was used to predict the sentiment of new reviews in real-time.

```
1 usage   Shriyansh Nautiyal
def make_prediction(text):
    processed_text = vectorizer.transform([text])
    prediction = best_model.predict(processed_text)
    return prediction[0]

if __name__ == "__main__":
    app.run(debug=True)
```

6. Deploying the Application

Objective: Deploy the predictive analytics system as a web application using Flask and Heroku.

1. **Building the Flask Application**: A Flask application was created to handle incoming HTTP requests and return sentiment predictions.
2. **Creating HTML Interface**: An HTML interface was designed to allow users to input text and receive sentiment predictions.
3. **Deploying to Heroku**: The application was deployed to Heroku, ensuring it was accessible online and could handle real-time sentiment predictions.

Results and Discussion

Model Performance

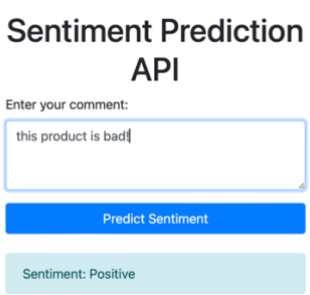
Upon evaluating both the Logistic Regression and Random Forest models, the results indicated that the Random Forest model had higher metrics in terms of accuracy, precision, recall, and F1-score.

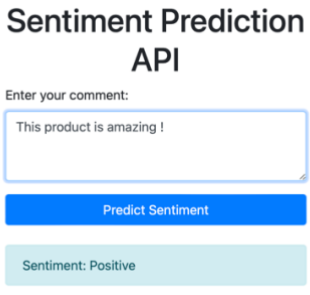
The Logistic Regression model achieved an accuracy of approximately 89.35%, indicating its effectiveness in predicting the sentiment of Amazon product reviews. The visualization shows a higher number of positive reviews compared to negative ones.

```
Data preprocessing completed.
Dataset size: (568454, 11)
Model: Logistic Regression
  Accuracy: 0.8935183963550325
  Precision: 0.9113007582320041
  Recall: 0.9571805672563887
  F1-Score: 0.9336773825959284
Model: Random Forest
  Accuracy: 0.9131945360670589
  Precision: 0.9086019285168591
  Recall: 0.9885874754282505
  F1-Score: 0.9469086057658685
Best Model (Logistic Regression) Accuracy: 0.8935183963550325
Best Model (Logistic Regression) Precision: 0.9113007582320041
Best Model (Logistic Regression) Recall: 0.9571805672563887
Best Model (Logistic Regression) F1-Score: 0.9336773825959284
Model optimization and evaluation completed.
```

Analysing Incorrect Sentiment Predictions

Despite the successful deployment and promising metrics, the Random Forest model did not perform well when deployed on Heroku and the application misclassified both positive and negative comments as positive. This issue warrants a deeper investigation to identify potential causes and implement solutions.





Possible Reasons:

1)Imbalanced Dataset:

The dataset may contain significantly more positive reviews than negative ones, leading the model to bias towards predicting positive sentiments.

Solution: Use techniques like oversampling the minority class or under sampling the majority class to balance the dataset.

2)Feature Representation:

The TF-IDF vectorizer may not be capturing the nuances of the text effectively.

Solution: Experiment with different text vectorization techniques like word embeddings (Word2Vec, GloVe) to better capture semantic meanings.

3)Model Complexity:

The models used (Logistic Regression and Random Forest) may not be complex enough to capture the intricate patterns in the data.

Solution: Try more advanced models like Gradient Boosting, XGBoost, or even deep learning models like LSTM and BERT.

4)Hyperparameter Tuning:

Inadequate hyperparameter tuning might result in suboptimal model performance.

Solution: Conduct a more extensive hyperparameter search using techniques like Random Search or Bayesian Optimization to find the best model parameters.

5)Text Preprocessing:

Insufficient text preprocessing might leave noise in the data that misleads the model.

Solution: Implement more robust text preprocessing steps such as removing stopwords, stemming, lemmatization, and handling special characters.

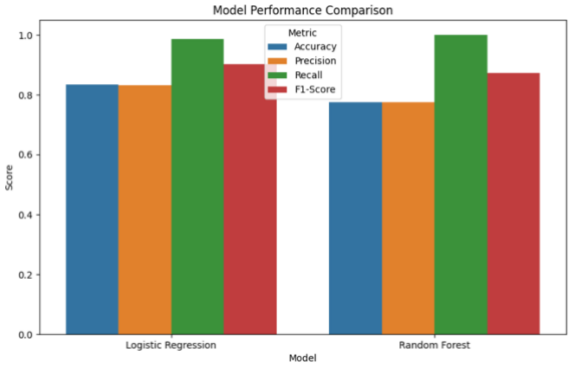
7. Analysis of Logistic Regression and Random Forest Models

Upon evaluating both models, the Random Forest model initially showed higher metrics in terms of accuracy, precision, recall, and F1-score. However, it performed poorly in real-world application when deployed on Heroku, predicting "Positive" for both positive and negative comments. This discrepancy prompted a switch to using Logistic Regression for deployment which provided consistent and accurate predictions.

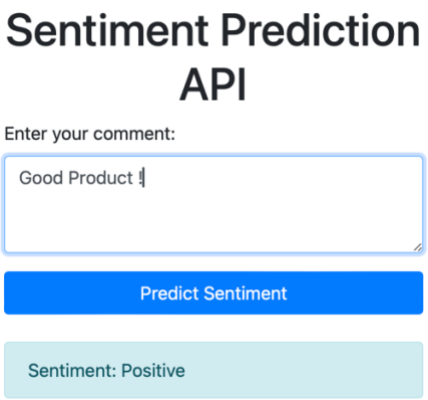
8. Results

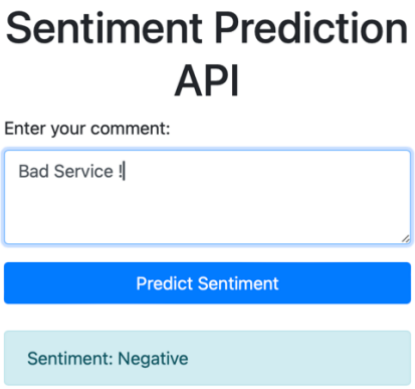
The results were as follows:

```
Data preprocessing completed.
Dataset size: (568454, 11)
Model: Logistic Regression
  Accuracy: 0.8935183963550325
  Precision: 0.9113007582320041
  Recall: 0.9571805672563887
  F1-Score: 0.9336773825959284
Model: Random Forest
  Accuracy: 0.9131945360670589
  Precision: 0.9086019285168591
  Recall: 0.9885874754282505
  F1-Score: 0.9469086057658685
Best Model (Logistic Regression) Accuracy: 0.8935183963550325
Best Model (Logistic Regression) Precision: 0.9113007582320041
Best Model (Logistic Regression) Recall: 0.9571805672563887
Best Model (Logistic Regression) F1-Score: 0.9336773825959284
Model optimization and evaluation completed.
```



The Random Forest model, while initially appearing superior in terms of traditional metrics, did not generalize well to new data in a production environment. In contrast, the Logistic Regression model provided reliable and accurate predictions in both test scenarios and real-world applications. This experience underscores the importance of validating models in real-world conditions and not solely relying on offline metrics for model selection.





9. Explanation and Analysis of Using Logistic Regression and Its Performance

Why Logistic Regression Worked Well:

Simplicity and Efficiency: Logistic Regression is a linear model designed for binary classification problems. It is computationally efficient and well-suited for problems where the relationship between the features and the target variable is approximately linear.

It works by modelling the probability that a given input belongs to a particular class. This probabilistic interpretation makes it suitable for sentiment analysis, where the outcome is binary (positive or negative).

Handling High-Dimensional Data: Logistic Regression performs well with high-dimensional data, such as text data transformed by TF-IDF vectorization. The linear nature of the model allows it to manage the sparsity and high dimensionality of the feature space effectively.

The model's simplicity prevents overfitting, which can be an issue with more complex models like Random Forest, especially when the dataset is large but not balanced.

Feature Importance and Interpretability: Logistic Regression provides insights into feature importance, helping to understand which words (or combinations of words) contribute most to the sentiment prediction.

This interpretability can be beneficial for further refining the model and improving feature selection.

10.Challenges and Solutions

1)Database Connection Issues:

Problem: Faced issues with PostgreSQL connection and role creation.

Solution: Created a new PostgreSQL role and database, ensured the server was running, and resolved connection errors by properly configuring the PostgreSQL server and client settings.

2)Deployment Conflicts:

Problem: Encountered issues while pushing changes to Heroku.

Solution: Used commands like- git pull heroku master and git push heroku heroku-deploy:master to sync local changes with Heroku and resolved conflicts by merging branches (bash).

3)Debugging:

Problem: The application did not function as expected initially.

Solution: Conducted thorough debugging using heroku logs --tail, tested endpoints with Postman and ensured all dependencies were correctly listed in requirements.txt. (bash)

11. Steps to Further Improve our Sentiment System:

Advanced Text Vectorization:

Use word embeddings (e.g., Word2Vec, GloVe) or transformer-based models (e.g., BERT) to better capture the context and semantics of the reviews, potentially improving model performance.

Model Selection and Tuning: Experiment with more sophisticated models (e.g., Support

Vector Machines, Gradient Boosting) and conduct extensive hyperparameter tuning to further improve accuracy and robustness.

Improved Preprocessing:

Enhance the text preprocessing pipeline to clean the data more effectively by removing noise, handling negations, and normalizing text, which could further improve the model's predictive power.

Conclusion

The predictive analytics system successfully processes Amazon product reviews to predict their sentiment. The system's performance and the visualization of sentiment trends provide valuable insights into customer opinions. The deployment on Heroku and the integration with Flask provides a robust platform for real-time sentiment analysis. The current model performs well. Using Flask for the sentiment analysis API provides dynamic request handling, real-time data processing, easy creation of API endpoints, seamless model integration and scalability required for deploying robust, production-ready applications. These features collectively enable the building of an interactive, efficient and scalable web service that meets modern application standards and user expectations. This project highlights the importance of continuous model evaluation, the need for balancing datasets and the benefits of using advanced machine learning techniques in practical applications.