Sentiment Polarity Classification

CS683 (*Natural Language Processing*)

Akshat Lohia

Roll No - 2101024

B.Tech CSE - 7th Semester

Problem Statement:

Take the following data with 5,331 positive and 5,331 negative sentences in it: https://www.cs.cornell.edu/people/pabo/movie-review-data/rt-polaritydata.tar.gz. Create a training set with the first 4,000 positive and the first 4,000 negative texts. Create a validation set with the next 500 positive and the next 500 negative texts. Create a test set with the final 831 positive and the final 831 negative texts. Create a binary (polarity) classifier using any ML/DL approach of your choice.

Objective:

The objective of this assignment is to develop an accurate binary sentiment classifier for movie reviews, specifically focusing on the following aspects:

- **Model Development**: To achieve this, three distinct machine learning models are explored:
    - **Naive Bayes**: Implement a probabilistic model based on Bayes' theorem, which assumes independence among features, ideal for text classification.
    - **Logistic Regression**: Utilise a widely-used binary classification model that predicts outcomes based on the linear relationship between features.
    - **Recurrent Neural Network (RNN)**: Explore a deep learning model capable of capturing sequential dependencies in text, enhancing the understanding of contextual relationships.
- **Training and Evaluation**: Training each model using the provided dataset and evaluating each model performance on the test set.
- **Performance Metrics**: Identifying the model that achieves the highest accuracy along with optimal metrics, including:
    - Precision
    - Recall
    - F1-score

Data Preparation for Sentiment Classification:

1. **Dataset Overview**: The dataset consists of 5,331 positive and 5,331 negative movie reviews, sourced from the provided dataset.

2. **Dataset Splitting**:
    - **Training Set**:
        - 4,000 positive reviews
        - 4,000 negative reviews
    - **Validation Set**:
        - 500 positive reviews
        - 500 negative reviews
    - **Test Set**:

    - 831 positive reviews
    - 831 negative reviews

3. **Feature Extraction**: Text data is converted into a numerical format suitable for machine learning models using **CountVectorizer**.

4. **Key Steps in the Code**:
    - Loading the dataset from a URL.
    - Extracting the contents of the downloaded tar file.
    - Reading the positive and negative reviews into lists.
        - Positive and negative reviews are read from their respective files and stored in lists for processing. The `latin-1` encoding is used to handle special characters.
    - Splitting the reviews into training, validation, and test sets.
    - Creating binary features from the text data for model training.
        - Labels are created for each review, where `1` represents positive sentiment and `0` represents negative sentiment.

Comparison of Sentiment Classification Models:

Model	TP	TN	FP	FN	Precision	Recall	F1-Score
Naive Bayes	629	673	158	202	0.80	0.76	0.78
Logistic Regression	620	651	180	211	0.78	0.75	0.76
RNN	550	655	176	281	0.76	0.66	0.71

- **Naive Bayes** emerged as the best-performing model with the highest F1-score (0.78) and precision (0.80), making it the most reliable choice for sentiment classification for the provided dataset.
- **Logistic Regression** followed closely, while the **RNN** model performed the least well, particularly in recall and F1-score.

Naive Bayes Classifier

Naive Bayes is a probabilistic classifier based on Bayes' theorem, commonly used for text classification tasks. It is particularly effective for sentiment analysis due to its simplicity and ability to handle high-dimensional data.

Approach of Naive Bayes Classifier:

1. **Bayes' Theorem**: The classifier is based on Bayes' Theorem, which describes the probability of a label C (e.g., positive or negative sentiment) given a set of features X (e.g., words in a review). Bayes' Theorem is mathematically represented as:

$$P(C|X) = \frac{P(X|C) \cdot P(C)}{P(X)}$$

2. **Conditional Independence Assumption**: Naive Bayes assumes that features are independent, which simplifies the calculation of the likelihood P(X|C) to the product of individual probabilities of each feature. For text classification, X is represented by the words in a document, and P(X|C) is calculated as the product of the individual word probabilities P(xi|C).

3. **Multinomial Naive Bayes**: In the context of text classification, the **Multinomial Naive Bayes** algorithm is used. It models the frequency of words in a document and uses the multinomial distribution to calculate the likelihood.

4. **Class Prediction**: The predicted class for an input document is the class C that maximizes the posterior probability P(C|X):

$$\hat{C} = \arg\max_C P(C) \prod_{i=1}^{n} P(x_i|C)$$

Steps in the Code:

1. Training Phase: The Naive Bayes classifier (MultinomialNB) was selected as the model due to its effectiveness for text-based tasks, especially for problems involving discrete data such as word frequencies.

- The training dataset (X_train and train_labels) is used to teach the model the relationships between the input features (word occurrences) and the output labels (positive or negative).
- The model is trained using nb_model.fit(X_train, train_labels). During this process, Naive Bayes estimates the probabilities of each word appearing in a positive or negative review.
- **CountVectorizer** was used to convert the text data into a feature matrix (X_train), which allowed the Naive Bayes model to compute word probabilities.

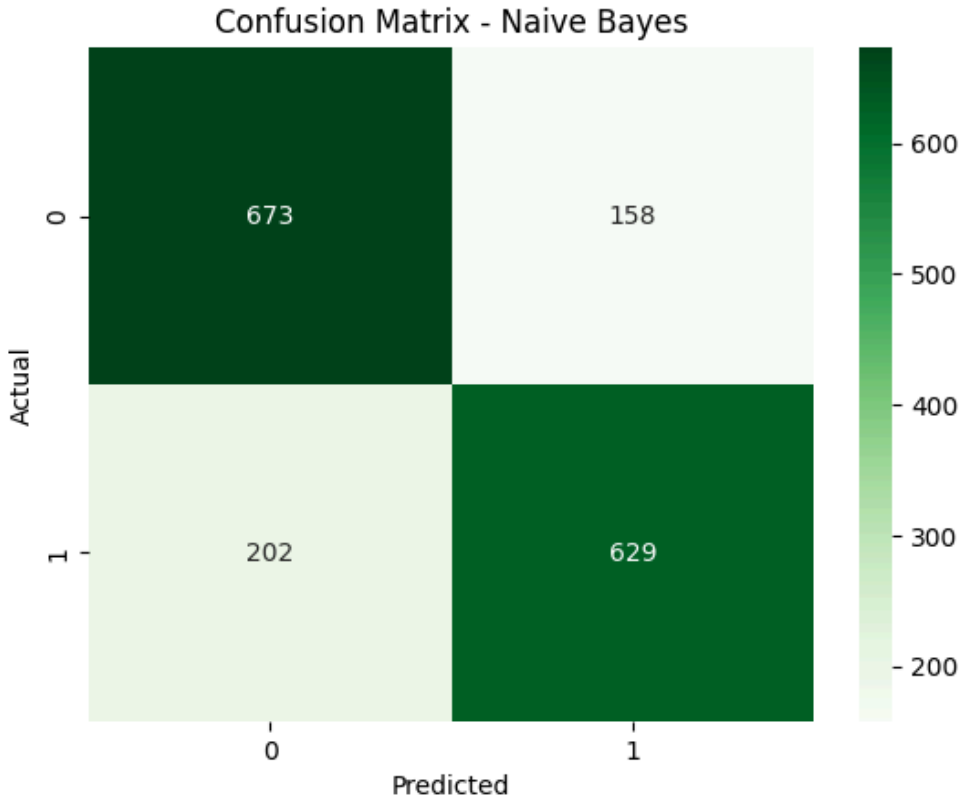
2. Validation Phase: The validation set is employed to tune hyperparameters and prevent overfitting. In this assignment, the validation set (val_data and val_labels) were created to adjust the hyperparameters like alpha.

3. Testing Phase:

- **Model Prediction**: After training, the model is evaluated on the test dataset (X_test) using nb_model.predict(X_test).
- The test set (test_data and test_labels) is entirely separate from the training set to ensure that the model's performance is evaluated on unseen data.
- The output (nb_predictions) contains the predicted labels for each test sample.

4. Evaluation Phase:

- **Confusion Matrix**: The confusion matrix compares predicted and actual results, showing:
  - **True Positive (TP)**: Correctly predicted positive reviews (629).
  - **True Negative (TN)**: Correctly predicted negative reviews (673).
  - **False Positive (FP)**: Incorrect positive predictions (158).
  - **False Negative (FN)**: Incorrect negative predictions (202). These values are printed and visualized using a Seaborn heatmap for clarity.



- **Precision, Recall, and F1-Score**:
  - **Precision**: Proportion of correct positive predictions (0.80).
  - **Recall**: Proportion of actual positives correctly predicted (0.76).
  - **F1-Score**: Harmonic mean of precision and recall (0.78), indicating balance.
- **Classification Report**: Provides a summary of precision, recall, and F1-score for each class, helping assess model performance comprehensively.

Steps for Executing the Code

1. Software Requirements

- **Python**: Version 3.7 or above.

Install Python: Download and install Python from python.org.

2. Required Python Packages

Install the following packages using `pip`:

```
pip install scikit-learn matplotlib seaborn tensorflow numpy
```

3. Execute the Code

- Once you have installed all the dependencies, save the provided code in a `.py` file (e.g., `sentiment_analysis.py`).
- Run the script from your terminal/command prompt:
  ```
  python sentiment_analysis.py
  ```

This will execute the entire process, from downloading the dataset to training the model and evaluating its performance.

Repository:

Google Colab Links: NLP_Assg1.ipynb

Github Link: https://github.com/Akshatlohia/NLP_Assg1

Report Link: NLP_Assg1