Project Final Report

Team Members:

Dev Patel - dzp0074@auburn.edu

Shyam Patel - sjp00590@auburn.edu (Coordinator)

Subeen Kim - szk0239@auburn.edu

Introduction

The dataset we worked on is a Movie Review Dataset, which contains 5,331 positive and 5,331 negative sentences from Rotten Tomatoes movie reviews - Rotten Tomatoes is a popular website and review aggregation service for films and television.

Problem Description

The primary challenge with the original dataset is its binary classification system, which labels reviews as either "good" or "bad." However, the reality of movie reviews often encompasses a spectrum of opinions, not all of which fit neatly into these categories. To address this, we've refined the classification to include a third category, "conflicted," recognizing that many reviews express nuanced or mixed sentiments.

To enhance our analysis, we introduced a Sentiment Score for each review. This score quantifies the extent of positivity or negativity expressed in a review, providing a percentage-based weight to indicate whether a review is predominantly "good," "bad," or "conflicted." The classification thresholds are set as follows:

- Bad: Scores ranging from -1 to -0.1
- Conflicted: Scores between -0.1 and 0.1
- Good: Scores from 0.1 to 1

To increase the accuracy of the Sentiment Score, we incorporated techniques such as N-grams and negation handling. N-grams help capture contextual nuances by considering pairs or triples of consecutive

words that might alter sentiment interpretation, such as the phrase "not good." Negation, the presence of words or phrases that can reverse or significantly alter the sentiment conveyed, is vital for accurate sentiment analysis. It ensures that expressions like "not bad" or "couldn't be better" are appropriately accounted for, thus providing a more accurate depiction of sentiment in text data. This enhanced approach allows us to more effectively capture the diverse range of sentiments expressed in movie reviews, acknowledging the complex emotional responses that films can evoke.

ML Theory and Implementation

Logistic Regression:

Logistic regression is a statistical model that uses a logistic function to model a binary dependent variable. In the machine learning context, logistic regression is used for binary classification tasks meaning categorizing data into one of two classes. The sigmoid function is also known as a logistic function, it takes any real-valued number and maps it into a value between 0 and 1, but never exactly at those limits. These values are interpreted as probabilities that the input data belong to one class or the other, typically the “positive” class if the probability is greater than 0.5 and the “negative” class otherwise.

The model is based on the concept of odds ratio and log-odds, where the logarithm of the odds of the positive class is modeled as a linear combination of the predictors. The coefficients of the linear combination are estimated from the training data, typically using maximum likelihood estimation. The mathematical form of the logistic regression model is:

$$log(\frac{p}{1-p}) = \beta_0 + \beta_1 x_1 + \ldots + \beta_n x_n$$

Where p is the probability of the positive class, β_0 is the intercept, β_1, …, β_n are the coefficients, and x_1, …, x_n are the features.

In our implementation, logistic regression was applied to a multi-class classification problem using the one-vs-rest strategy. The ‘TfidfVectorizer’ from ‘scikit-learn’ was used to convert the text data into a matrix of TF-IDF features. These features served as input to the ‘LogisticRegression’ model. The

‘max_iter’ parameter was set to 1000 to give the optimization algorithm sufficient iterations to converge to a solution.

Multi-Layer Perceptrons (MLP)

Multi-layer perceptrons are a class of feedforward artificial neural networks. An MLP consists of at least three layers of nodes: an input layer, a hidden layer, and an output layer. Except for the input nodes, each node is a neuron with a nonlinear activation function. MLP uses backpropagation for training, which involves a forward pass where the inputs are passed through the network to generate an output, and a backward pass where the error is propagated back through the network to update the weights. The error in this case is the difference between the observed and predicted values.

MLPs can effectively capture complex relationships between inputs and outputs by adjusting the weights of the connections in the network during the training phase. The general phase form of the model for a single neuron can be written as:

$$y = f(\sum_{i=1}^{n} w_i x_i + b)$$

Where y is the output, f is an activation function (such as the sigmoid function), w_i are the weights, x_i are the inputs, and b is a bias term.

The MLP classifier in the project uses the ‘MLPClassifier’ class from ‘scikit-learn’. The hidden layer sizes and activation functions are parameters that can be adjusted according to the complexity of the task. In this case, the ‘random-state’ was set for reproducibility, and ‘max_iter’ was set to 300 to allow the learning algorithm to converge.

Both of our models were evaluated using precision, recall, and F1-score metrics, thus providing a comprehensive assessment of their performance. The output classes were encoded using ‘LabelEncoder’ to transform them into suitable format for the models.

Instructions to Run Program

Before running the script, ensure your Python environment is set up with the necessary libraries. You need libraries like ‘pandas’, ‘sklearn’, textblob’, and others installed. Use ‘pip install’ commands to get any missing libraries.

Instructions to follow:

Input- You will be asked to enter search queries related to movie reviews. This could include actor names, movie titles, or any keywords related to the reviews in the dataset. You can input multiple queries and to exit the program, simply put “quit”.

Output- For each query, the script displays the filtered reviews and their predicted sentiment classes. Outputs include the review text, its class, and sentiment score. The performance metrics of the models are displayed at the end.

What the program does:

1. Loading the Dataset:

   The script starts by loading the ‘rotten_tomatoes’ dataset. Make sure you have internet access if the dataset needs to be fetched online.

2. Data Preprocessing:

   Extracts the review texts and computes their sentiment scores using TextBlob, which assigns a polarity score to each text. Labels are assigned based on the sentiment score using predetermined thresholds.

3. Feature Extraction:

   The script uses TF-IDF vectorization, considering up to 5000 features with unigrams and bigrams, to convert text data into a format suitable for machine learning models.

4. Model Training:

   The Logistic Regression and MLP models are trained using the vectorized training data.

5. User Interaction:

   The script enters a loop where it continuously prompts the user to input search queries (such as actor names or movie titles).

For each query, it filters reviews containing the query, predicts their sentiment class using both trained models, and displays the results. You can type ‘quit’ to exit the loop.

6. Performance Evaluation:

   After exiting the search loop, the script evaluates the performance of the Logistic Regression and MLP models against a random baseline, showing metrics like precision, recall, and F-1score.

Results

We created a random baseline performance and used that to compare the performances for both the Logistic Regression and Multilayer Perceptron Model’s performances. We observed that the random baseline performance achieves an accuracy of 37%. This performance sets a low benchmark for other models. The precision and recall across categories hover around 34% on average, indicating a relatively uniform but poor performance across all sentiment classes. The f1-score, which balances precision and recall, reflects similar results.

The Logistic Regression model shows a significant improvement with an accuracy of 62%, suggesting it's considerably better at capturing the underlying patterns in the data compared to the random baseline. It performs best on the "positive" category with a precision of 67% and recall of 82%, The model struggles relatively more with the "negative" category, achieving only a 28% recall, and the "conflicted" category sees a balanced performance with a precision of 51% and recall of 54%.

The Multilayer Perceptron model achieves an accuracy of 60%, which is close to that of logistic regression but slightly lower. It offers more balanced performance across all categories compared to logistic regression, with macro average precision and recall around 57%. The model performs best in the "positive" category with precision and recall of 72% and 74%, It shows a competitive precision in the "negative" category (51%) but with a lower recall than logistic regression.

Overall, these results illustrate the challenges and nuances of multi-class sentiment analysis in text, particularly in distinguishing between nuanced expressions like those in the "conflicted" category.

Area of Improvement

When testing the program, we noticed that some areas of improvement can come from fine tuning our Sentiment Scores because some reviews were classified in the wrong categories. If we spent more time adding depth to our Sentiment Scores these classifications would improve.

Team Contributions

Each team member contributed to many different aspects of the development and documentation process to help ensure the project's success. Here is a breakdown of each member's specific contributions:

Dev

Dev was primarily responsible for the coding and implementation of the machine learning models used in our project. He undertook the task of writing the initial code, integrating the machine learning algorithms, and ensuring their correct execution within the project framework. He played a crucial role in implementing the logistic regression and multi-layer perceptron models.

Shyam

Shyam played a key role in testing and debugging the code. He worked closely with others to identify and resolve errors within the implementation, ensuring the code was not only functional but also efficient. In addition to his contributions to coding, Shyam was responsible for writing the Problem Description section of the report. His work here focused on clarifying the objectives and scope of our analysis, as well as discussing the results and observations that were derived from applying the two methods to our dataset. As team coordinator, Shyam was in charge of setting up meetings and holding lines of communication with all team members.

Subeen

Subeen also focused on testing and validating the code. Her precise approach to testing helped to further ensure that our software was robust and reliable. Additionally, Subeen was tasked with creating the software documentation. She detailed the setup, execution, and functionality of our machine learning models, provided clear instructions and insights that would allow others to run or work on our work.