CS4830 Group Project: Yelp Review Rating Prediction using Basic Natural Language Processing and Machine Learning Techniques

Ishan Chokshi - BE19B018
Department of Biotechnology
Indian Institute of Technology Madras
Chennai, India
be19b018@smail.iitm.ac.in

Devansh Sanghvi - BE19B002
Department of Biotechnology
Indian Institute of Technology Madras
Chennai, India
be19b002@smail.iitm.ac.in

Siddharth Betala - BE19B032
Department of Biotechnology
Indian Institute of Technology Madras
Chennai, India
be19b032@smail.iitm.ac.in

***Abstract*—In this project, we aim to predict the star ratings of Yelp reviews using natural language processing (NLP) techniques, machine learning algorithms, and Kafka Streaming for real-time data processing. The Yelp dataset was pre-processed, cleaned, and features were engineered. We implemented an NLP pipeline and applied machine learning algorithms such as logistic regression and naive Bayes to predict the star ratings. Kafka Streaming was used to process real-time review data and make predictions using the trained models. The results were evaluated using appropriate performance metrics.**

I. Introduction

A. Background and Motivation

Online reviews have become a significant factor influencing customers' decisions and businesses' reputations. Yelp is a popular platform where users post reviews and rate businesses. These ratings impact businesses' online presence and potential customer engagement. Analyzing and predicting the ratings of user reviews can provide valuable insights for businesses to enhance their services and understand customer sentiments.

The motivation behind this project is to leverage natural language processing (NLP) techniques, machine learning algorithms, and real-time data processing using Kafka Streaming to predict Yelp review ratings. This can help businesses gain a competitive edge by identifying areas of improvement and understanding customer feedback

B. Objective

The primary objective of this project is to predict the star ratings of Yelp reviews using NLP techniques and machine learning algorithms. Our approach involves pre-processing the Yelp dataset, feature engineering, implementing an NLP pipeline, and training machine learning models such as logistic regression and naive Bayes. We will also incorporate Kafka Streaming for real-time data processing and making predictions using the trained models.

C. Dataset Description

The Yelp dataset consists of reviews posted by users on the Yelp website. The dataset has been pre-processed to be compatible with Spark SQL load commands. The training dataset provided contains multiple features, including review id, business id, user id, review text, and star ratings. Additionally, there are columns for the number of votes for cool, useful, and funny reviews. After pre-processing, the star ratings range from 0 to 5 and serve as the target variable for our prediction models. Note that we are using the provided training dataset and not the dataset directly from the website, as the real-time demo will pre-process the test set in the same manner as the training set.

D. Steps used in the project

The following flowchart summarizes the steps followed ahead:

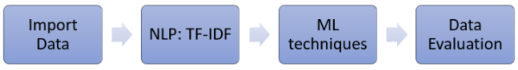

Fig. 1. First five rows of the dataframe

II. Data Preprocessing

A. Importing the Data

The Yelp dataset was imported into a Spark DataFrame from Google Cloud Storage using the gsutil command. The dataset was stored in a bucket located at gs://bdl2023-final-proj/YELP_train.csv. We utilized the following code to load the data into a Spark DataFrame:

B. Dropping unnecessary columns

We dropped the columns corresponding to the review id, business id, and user id to simplify the dataset and focus on

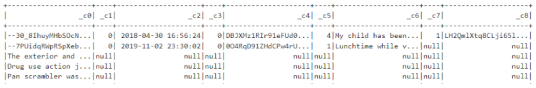

Fig. 2. First five rows of the dataframe

relevant features for our prediction task. This step is crucial for reducing noise in the data and improving model performance.

C. Handling Missing Values

In this stage, we dealt with missing values in the dataset. Rows with missing star values (our target variable) were dropped, as they would not contribute to the model's training. For other columns with missing values, we imputed them with their respective median values. This approach preserves the central tendency of the data and avoids introducing biases.

D. Data Filtering

We filtered the dataset based on the rating column, ensuring that the star rating values were within the valid range of 0 to 5. This step is essential to prevent the inclusion of erroneous or outlier data points that may negatively impact model performance.

E. Visualizations

To gain insights into the dataset and understand the relationships between features, we created visualizations such as:

1) Heatmap of correlation between features: This visualization helps identify the relationships between different features and the target variable (star ratings). It can reveal potential multicollinearity issues and guide feature selection. Based on the weak correlations observed in the heatmap, we can infer that these features are relatively independent and do not share much redundant information. This can be beneficial for our predictive model, as it reduces the risk of multicollinearity, which occurs when two or more features are highly correlated and provide similar information to the model. Multicollinearity can negatively impact model performance and interpretability.
2) Bar plot of counts for each rating value (0-5): This plot provides an overview of the distribution of star ratings in the dataset. It can help identify any class imbalances that may need to be addressed during model training. Based on the provided counts for each rating value, we can infer that the dataset is imbalanced, with more reviews having 4-star and 5-star ratings compared to reviews with lower ratings. This indicates that users tend to leave more positive reviews on Yelp. Since the dataset is imbalanced, it could affect the performance of the machine learning model. To mitigate this, one can try one of the following techniques:
    a) Resampling: One can either oversample the minority class (lower rating counts) or undersample the majority class (higher rating counts) to balance the dataset. SMOTE can also be used.

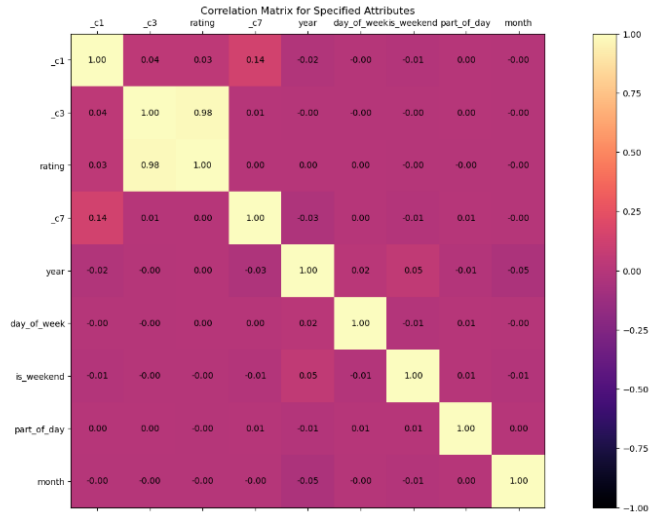

Fig. 3. Correlation Matrix for specified attributes

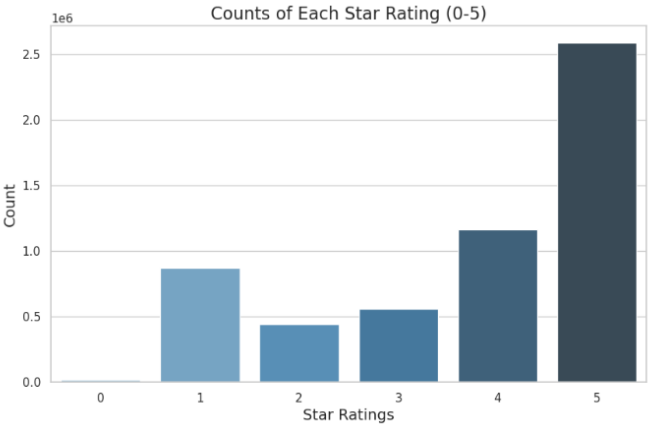

Fig. 4. Counts for each star rating (0-5)

    b) Using an appropriate evaluation metric: Using evaluation metrics such as F1-score, precision, recall, or area under the ROC curve (AUC-ROC) instead of accuracy, as they better handle imbalanced datasets. We'll be using the F1 score in this project.
    c) One can consider using algorithms that are less sensitive to class imbalance, such as tree-based models or ensemble methods (e.g., Random Fores). We'll try using this if memory and computing resources permit.

III. Feature Engineering

Feature engineering is a crucial step in the machine learning pipeline, as it involves creating new features or modifying existing ones to improve model performance. In this project, we performed feature engineering on both the existing attributes and the text data from the Yelp reviews.

A. Features extracted from the date-time column

We extracted useful information from the date-time feature to create new features that can potentially improve the model's predictive power. The new date-time features include:

1) Month: The month when the review was posted can provide insights into seasonal trends and customer behavior.
2) Year: The year of the review can help capture changes in user preferences and business performance over time.
3) Weekend or not: A binary feature indicating whether the review was posted on a weekend (Saturday or Sunday) or a weekday. This feature can help understand the relationship between the time of the week and the user's ratings.
4) Day of the week: The specific day of the week when the review was posted. This can reveal any patterns associated with particular days of the week.
5) Part of the day (morning, afternoon, evening, night): The time of day when the review was posted can potentially capture differences in user sentiment and behavior throughout the day.

B. Additional Features

While we haven't incorporated these features into our training because of limited memory on the jupyter notebook, which was run on the cluster, to further improve the model's performance, one can consider creating additional features such as:

1) Review length: The number of words or characters in the review. Longer reviews might provide more detailed feedback and correlate with the user's sentiment and rating.
2) Sentiment polarity: A score representing the sentiment polarity of the review text. This can be obtained using sentiment analysis tools or libraries, such as TextBlob or VADER. Sentiment polarity might correlate with the user's rating and can provide additional information to the model.
3) Number of unique words: The count of distinct words in the review. This feature can help capture the diversity of the user's feedback.
4) The number of exclamation marks or question marks: The frequency of exclamation marks or question marks in the review can indicate the user's enthusiasm or confusion, potentially relating to the rating

These additional features can be added to the dataset to enhance the information available for the model to learn from and potentially improve its predictive accuracy.

_c1	_c3	rating	review	_c7	year	day_of_week	is_weekend	part_of_day	month
0	0	4	My child has been...	1	2018	2	0	1.0	4
0	0	1	Lunchtime while v...	0	2019	7	1	0.0	11
0	0	2	Eh.. not that gre...	6	2015	3	0	0.0	6
0	0	1	4706 Paris Ave, N...	0	2015	5	0	0.0	8
0	0	5	We recently had o...	0	2019	5	0	1.0	3

Fig. 5. Top 5 rows after preprocessing

IV. NLP Pipeline for Reviews

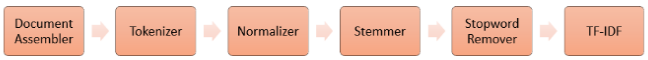

Fig. 6. NLP Pipeline for reviews column

The NLP pipeline is an essential part of the project, as it helps process and transform the raw review text into a format that the machine learning models can use. In this project, we implemented the following NLP pipeline using Spark NLP:

A. Document Assembler

The DocumentAssembler is the first stage in the NLP pipeline. It takes the raw review text from the input column ("review") and converts it into a document format, storing the result in the output column ("document").

B. Tokenizer

The Tokenizer stage takes the document format from the previous stage and splits it into individual words (tokens). The resulting tokens are stored in the output column ("token").

C. Normalizer

The Normalizer stage processes the tokens by removing punctuations and special characters and converting all characters to lowercase. This helps create a consistent format for the text data. The normalized tokens are stored in the output column ("normalizer").

D. Stemmer

The Stemmer stage reduces the tokens to their root forms (stems) by removing affixes. This helps group similar words together and reduces the overall dimensionality of the text data. The stemmed tokens are stored in the output column ("stem").

E. Stopword Remover

The Stopword Remover stage removes common stop words (such as "a", "an," and "the") from the tokenized text. Stop words typically do not carry much meaning and can be removed to reduce noise in the data. The filtered tokens are stored in the output column ("filtered").

F. Feature Extraction (TF-IDF)

After pre-processing the text data, we used the Term Frequency-Inverse Document Frequency (TF-IDF) technique to convert the filtered tokens into numerical features. The CountVectorizer computes the term frequencies, while the IDF function calculates the inverse document frequencies. The resulting TF-IDF features are stored in the output column ("features").
This NLP pipeline transforms the raw review text into a set of numerical features that can be used alongside the other engineered features in the dataset to train and evaluate machine learning models for rating prediction.

TF-IDF

The primary intuition behind TF-IDF is that words that frequently occur within a single document but rarely across the entire document collection are likely to be more significant and informative. The TF-IDF score for a term is calculated using the following formula:

$$tfidf(t, d, D) = tf(t, d) * idf(t, D) \tag{1}$$

Where,

1) t represents a term (word) in a document.
2) d is the document containing the term.
3) D is the collection of all documents.

The formula consists of two components:

1) **Term Frequency (tf)**: It measures the frequency of a term 't' in the document 'd'. A higher term frequency indicates that the term appears more often in the document. There are different ways to calculate term frequency, but the most common one is the raw count of the term in the document:
   - tf(t, d) = term t in document d
2) **Inverse Document Frequency (idf):** It measures the importance of a term across the entire document collection 'D'. If a term appears in many documents, it is less informative and less relevant for classification or clustering tasks. The idf is calculated using the following formula:

$$idf(t, D) = log(N/df(t)) \tag{2}$$

   Where:
   a) N is the total number of documents in the collection D.
   b) df(t) is the number of documents containing the term t

   The final TF-IDF score is the product of the term frequency (tf) and the inverse document frequency (IDF). Words with a high TF-IDF score are more important and relevant, as they frequently appear within a document but rarely across the entire document collection. This makes TF-IDF a useful feature extraction technique for text data, as it can help identify meaningful patterns and relationships in the text that are relevant for classification.

V. Model Training and Evaluation

After the pre-processing and feature engineering, we use the modified dataset to train our ML models. We tried training using Naive Bayes, Logistic Regression, Decision Trees, and Random Forest.

However, we kept running into limited memory errors in every case except with Naive Bayes. We tried training other models with as little as a 1 percent random sample of the data and with just the column pertaining to reviews. However, the memory issues wouldn't go away. There might also be compatibility issues with the current version of the sparknlp library and pyspark. Another reason might be that using tf-idf on the reviews column led to the creation of sparse matrices as embeddings for the reviews. We also tried using PCA on these embeddings; however, that led to memory issues as well. Eventually, we tried to see if the training was possible with word2vec embeddings and whether we'll get more compact embeddings, but we kept getting recurring memory errors while using the word2vec-based NLP pipeline.

Naive Bayes is a probabilistic machine learning algorithm used for classification tasks. It is based on the Bayes theorem, which describes the probability of an event based on prior knowledge of conditions that might be related to the event.

The "naive" part of the algorithm's name comes from the assumption that all features in the data are independent of each other, which may not always be true in real-world scenarios, but it simplifies the algorithm and makes it computationally efficient.

Naive Bayes works by first training on a labeled dataset, where each data point is assigned a class label. It then uses the Bayes theorem to calculate the probability of a new data point belonging to each class label based on the features it possesses. The class label with the highest probability is then assigned to the new data point.

The algorithm works by calculating the conditional probability of each feature given a class label, which is called the likelihood. It also calculates the prior probability of each class label, which is based on the frequency of that label in the training data. Finally, it uses the Bayes theorem to calculate the posterior probability of each class label given the new data point and its features.

The formula for the posterior probability for a class label C given a set of features x is:

$$P(C|x) = P(C) * P(x|C)/P(x) \tag{3}$$

where P(C) is the prior probability of the class label, P(x—C) is the likelihood of the features given the class label, and P(x) is the probability of the features occurring in the data.

Naive Bayes is a popular algorithm because it is fast, efficient, and works well with high-dimensional data. However, its performance may suffer if the independence assumption is violated or if the dataset is imbalanced or noisy.

The training was done after doing an 80-20 train-validation split on the prep-processed dataset. The features used for training were as follows: _c1, _c3, review, _c7, day_of_week, is_weekend, part_of_day, month, year

These features represent the following realistically:

VI. Kafka Streaming

For the Kafka streaming task, we have created the publisher and subscriber. Code for these tasks can be found attached. The publisher and subscriber run simultaneously. The subscriber takes the data that is provided by the publisher. Upon receiving the data, the subscriber runs the model and predicts the output which can be seen in the log of the subscriber job.

Features
Number of Useful Votes for review (Numerical)
Number of Cool Votes for review (Numerical)
Number of Funny Votes for review (Numerical)
Review Text (Numerical Embeddings generated using tf-idf)
Which Day of the week? (Categorical)
Is that day a weekend or not? (Categorical)
Part of the day when the review was posted (Morning, Afternoon, evening, night) (Categorical)
Month (Categorical)
Year (Categorical)

Fig. 7. Features and their type

1) Publisher: A kafka VM was created and the publisher code was run on the virtual machine SSH window.
2) Subscriber: We establish a subscriber on a separate Dataproc cluster simultaneously. The outputs are seen in batches. So as we update the publisher (or when it does while going over the dataset), each review is taken in one batch and the accuracy is calculated cumulatively over the tested reviews.

VII. Results and Conclusion

The results we achieved on the validation set are as follows:

Model	Accuracy	F1 Score
Naive Bayes	0.51	0.56

Fig. 8. Naive Bayes Result