Text Mining and Natural Language Processing 2022-2023

Chiara Fantoni 505079

Rowyda Askalani 502219

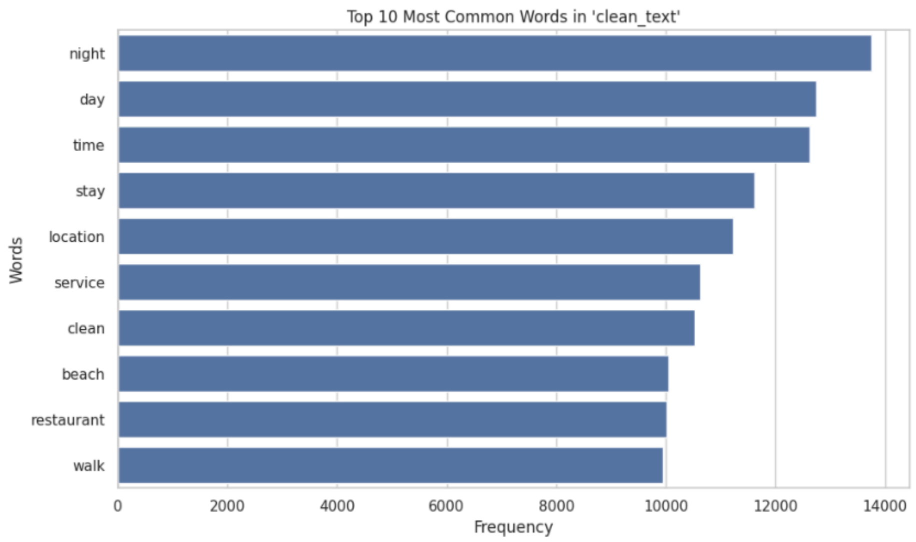
TripAdvisor Hotel Reviews

Introduction

This project focuses on the Text Classification of tripadvisor reviews based on the rating associated with it. Different steps were considered to make this project a reality. First, we pre-processed the dataset to have a cleaner dataframe and to make it easier for the model to interpret the data. We used some of the following techniques: removing punctuations, stopwords (with exceptions), stemming procedure, tokenization,etc. Second, we split the data into training and testing sets and then tested multiple models to find the most suitable one for our project, settling for logistic regression. In addition, we also chose to represent the text using *Tf-idf* to help build the text classification model. Next, we decided to define a parameter grid for the hyper parameter tuning and then perform a *GridSearchCV* to find the best hyperparameters for the logistic regression model. Then, we trained the final model on the entire dataset using the best hyperparameters we found and also evaluated it on the test set. Next, we interpret the model by calculating some statistics to show which words contributed the most to our model(in our case they were mostly the negative words). Finally, we allowed the user to insert their own reviews and receive the rating based on the previous steps applied to the original data.

Data

As stated in the introduction, we used a TripAdvisor reviews dataset for the project. We performed some statistical analysis on the data to see the overall number of reviews present in the data, the number of words and sentences present in the dataset, and the average number of words and sentences per review. We obtained the following data:
Number of Reviews: 20491
Total Number of Words: 2138765
Average Number of Words per Review: 104.38
Total Number of Sentences: 44415
Average Number of Sentences per Review: 2.17
In addition, we also created a barplot to visualize the most common words present in the data after cleaning and preprocessing it.
These are the results of the top 10 words present in the data:



Methodology

The project is divided into 11 important steps. In this section we will go into details of how each of them was implemented and how it contributed to the project.

1. **Imports and setup**:
    - The first step in the project was to import and install all the necessary libraries that are used throughout the project, this includes: *pandas, ntlk, sklearn, matplotlib, textblob, spellchecker, swifter, pyspellchecker*.
    - We then make use of the *gdown function* to upload the dataset from google drive without having to mount the drive itself.
    - The dataset is then loaded into a Pandas DataFrame for further preprocessing.
2. **Dataset statistics**:
    - We define a function called "*calculate_dataset_statistics*" that allows us to calculate and then print the following:
        - Number of reviews(num_reviews)
        - Total number of words(total_words)
        - Average number of words per review(avg_words_per_review)
        - Total number of sentences(total_sentences)
        - Average number of sentences per review(avg_sentences_per_review)
3. **Text cleaning**:
    - Text cleaning is a crucial preprocessing step as it improves the quality of the data and increases the effectiveness of the models and analysis. The following are some of the functions that we implemented:
        - *Lower()*: it's a function that allows us to set the text of the data in lowercase and therefore having a uniform case over the whole data
        - *remove_punctuations()*: this function is used to remove the punctuation to help us focus on the words and reduce data noise.
        - *remove_stopwords()*: this function is used to remove frequent words that often don't carry meaning. This helped us reduce the data and allowed us to focus on more important words. However, in our case we made exceptions to some words which are "not" and "but" as they are more meaningful for our dataset. In this function also tokenization was applied.
        - *remove_freq_words()*: this function is used to remove the top 10 most frequent words in the data to reduce the data size and because they tend to negatively affect the data.
        - *remove_rare_words()*: this function is used to remove the 1000 most rare words that appear in the dataset to reduce the data as well and they also have a negative effect rather than a positive one.

        - *remove_spl_chars()*: this function is used to remove special characters as they're irrelevant and makes it difficult to focus on the important words and reduce data noise
        - *stem_words()*: this function is used to apply stemming to the words to allow us to extract the roots of the words so they would have a common base
            - Initially we intended to use lemmatization, however it produced less efficient results when it comes to data prediction.
        - *removal_url()*: it's a function used to remove any URL that could be found inside the data.
        - *remove_html_tags()*: this function is used to remove unnecessary HTML tags present in the data.
        - *correct_spelling()*: this function is mainly created to correct the spelling mistakes found inside the data using textblob, swifter and pyspellchecker. However since the dataset was huge we had to show that it works only on a subset of the data due to time and resources limitations.
          However by uncommenting a certain line of code
          (#df['clean_text']=df['clean_text'].swifter.apply(correct_spelling))
          and commenting another
          (df['spelling'] = subset_df['clean_text'].swifter.apply(correct_spelling)) the spelling checker could be applied to the whole dataset.
    - After cleaning, we created a barplot that visualizes the top 10 words present in the data set after cleaning it
4. **Splitting and preparing the data**:
    - We split the data into a training set and a testing set, this is to ensure that the model is trained on 80% of the data and then could be evaluated on the test set to assess its performance. The label that was taken into consideration was the *'Rating'* column.

5. **Choosing the model**:
   - Before settling on the final model we tested various ones before choosing the one that provided the best accuracy for our project. We'll go through a few of the ones we tested and their results and before settling down:
     - *Naive Bayes Multinomial Classifier*: it is a probabilistic classifier based on Bayes' theorem. It is often used for classification problems, where the goal is to assign a class or category to a set of data based on certain characteristics. The "naive" (naive) approach derives from the assumption of conditional independence of characteristics, which simplifies the calculation of probabilities. The results are the following:

       -
         ```
         RESULTS:   precision    recall   f1-score   support
         accuracy                           0.44      4099
         macro avg       0.31        0.20      0.12      4099
         weighted avg  0.29        0.44      0.27      4099
         ```
     - *Support Vector Machine (SVM)*:it is good at solving binary classification problems, which require classifying the elements of a data set into two groups.it tries to find an optimal hyperplane to separate the different classes in our feature space. The results are the following:

       -
         ```
         RESULTS:   precision     recall    f1-score   support
         accuracy                                0.61      4099
         macro avg       0.56        0.48           0.50      4099
         weighted avg  0.59        0.61            0.59      4099
         ```
     - *Logistic Regression*: uses a logistics function to estimate the probability of belonging to a class. It uses a logistic function to model the dependent variable. The results are the following:

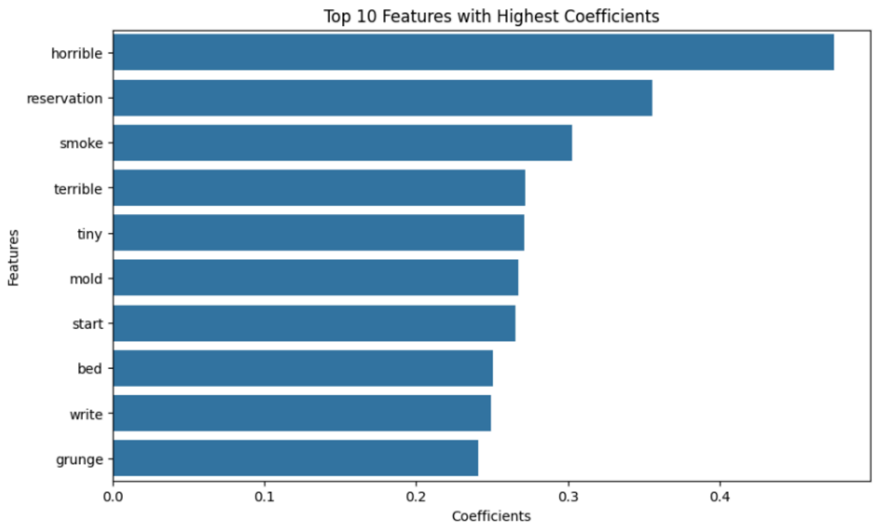
       -
         ```
         RESULTS:   precision     recall    f1-score   support
         accuracy                            0.61      4099
         macro avg     0.57         0.50        0.52      4099
         weighted avg  0.59         0.61       0.59      4099
         ```

    - *Random Forest Classifier*: is a machine learning algorithm which is an ensemble of decision trees,combining the output of multiple decision trees to reach a single result. It can be robust and suitable for complex datasets. The results are the following:
        - RESULTS: precision recall f1-score support
          accuracy 0.50 4099
          macro avg 0.45 0.30 0.29 4099
          weighted avg 0.45 0.50 0.41 4099
    - *Gradient Boosting*: is a functional gradient algorithm that combines several weak learning models to produce a powerful predicting model. It is an ensemble method that builds sequential trees, correcting the errors of previous trees. The results are the following:
        - RESULTS: precision recall f1-score support
          accuracy 0.55 4099
          macro avg 0.51 0.40 0.42 4099
          weighted avg 0.53 0.55 0.52 4099
    - *MLP Classifier*: neural networks can be used for complex classification problems, especially if you have a large number of data. It can be used for sentiment analysis and text categorization. The results are the following:
        - RESULTS: precision recall f1-score support
          accuracy 0.55 4099
          macro avg 0.50 0.47 0.49 4099
          weighted avg 0.54 0.55 0.55 4099
    - *K-Nearest Neighbors (KNN)*: Assigns a class label based on the majority of the labels of its closest observations. It uses proximity to make classifications or predictions about the grouping of an individual data point. The results are the following:
        - RESULTS: precision recall f1-score support
          accuracy 0.48 4099
          macro avg 0.38 0.35 0.36 4099
          weighted avg 0.45 0.48 0.46 4099
- After testing the various models we chose to use the **Logistic regression model** as it was the one that produced the best scores as shown previously and was the most optimal for this case.

6. **Logistic Regression model**:
    - We create a pipeline that takes the logistic regression model into account as well as incorporating a *Tf-idf vectorizer* having a max iteration 1000 due to the program reaching the default max iterations quickly.
    - Next, we fit and predict the model on the training and testing sets and then evaluate the performance accordingly and visualize the scores, including accuracy and a classification report, obtained.
7. **Hyperparameter Tuning**:
    - We create a parameter grid that takes into consideration the following various parameters:
        - C: [0.001, 0.01, 0.1, 1, 10, 100]
        - Max_iter: [50, 100]
        - n_jobs: [-1]
        - penalty: ['l2']
        - random_state: [42]
    - We define then the *GridSearchCV function* that is used to find the optimal values for our logistic regression model; The *GridSearchCV* takes as parameters the model, the parameters grid, the cross validation (we chose 5) and a scoring metric which is accuracy in our case.
    - Finally we train the model using the *GridSearchCV* and print the best parameters and the best accuracy found. Which are:
        - Best parameters for Logistic Regression: {'logisticregression__C': 1, 'logisticregression__max_iter': 50, 'logisticregression__n_jobs': -1, 'logisticregression__penalty': 'l2', 'logisticregression__random_state': 42}
        - Best accuracy for Logistic Regression: 0.600902911477833
8. **Train the final model**:
    - We created a pipeline having the *Tf-Idf vectorizer* and the *logistic regression model* that takes as parameters the best hyperparameters we have previously found
    - Using this final model, we train the full dataset
    - We then adapt and train the *Tf-Idf vectorizer* on our data to have a complete model
9. **Evaluate on the test set**:
    - Using the final model, we use it to predict the ratings on the test set and then evaluate the model's performance using the accuracy and the classification report to print the results

10. **Interpretation of the model**:

- To interpret the model we decided, to access the coefficients of the logistic regression model and then map the corresponding words in the *Tf-Idf vectorizer* to determine the top 10 words that contributed the most to the model's prediction having the highest coefficients
- We then print the results using a barplot to have a more visually appealing output of the results. This is the result of the model interpretation

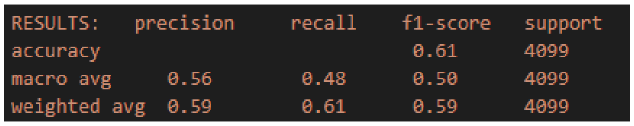


○

11. **New data**:

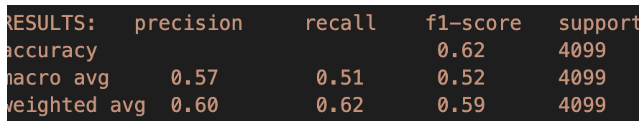
- We decided that we wanted to test our model also on new data that is inserted by the user allowing them to obtain the rating based on their inputted reviews.
- The new input undergoes the same preprocessing steps that have been applied to the original dataset and then the rating is predicted using the final model that we created
- Finally the rating is printed based on the review inserted.

Result

- Some of the results we wanted to examine are the results the various models produce and the results of the new data prediction:
    - As explained in step 5 of the 'Methodology', we tested various models on our data to try and find the one that produces the best results. Observing the scores we obtained, Logistic regression and support vector machine classifier have very similar results but since we wanted to focus on one model and choose the most accurate for our case, we went ahead with Logistic regression
        - Results of logistic regression and SVM side by side:
            - 

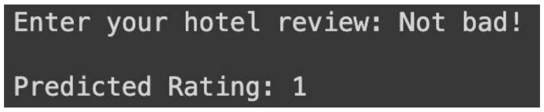
```
RESULTS:    precision    recall    f1-score    support
accuracy                           0.61        4099
macro avg     0.56         0.48    0.50        4099
weighted avg  0.59         0.61    0.59        4099
```

SVM results

            - 

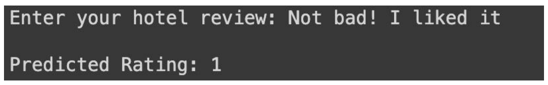
```
RESULTS:    precision     recall    f1-score    support
accuracy                            0.62        4099
acro avg      0.57         0.51     0.52        4099
eighted avg   0.60         0.62     0.59        4099
```

LR results

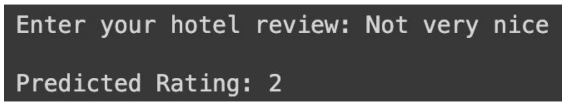
    - Due to our model not having a very high accuracy, the predictions on new input appears to be slightly inaccurate as it doesn't always take into account negative values especially when the review refers to something 'not' negative and/or even praising it. For example:
        - 

```
Enter your hotel review: Not bad!

Predicted Rating: 1
```

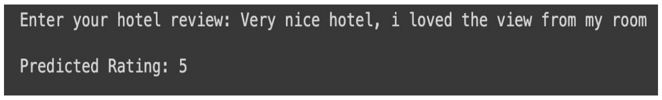
(could have been a 3)

        - 

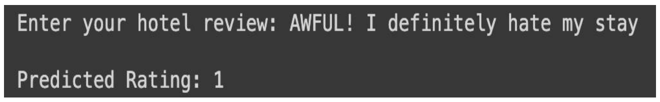
```
Enter your hotel review: Not bad! I liked it

Predicted Rating: 1
```

    - However it seems that the model takes into consideration negative values when they are accompanied by positive adjectives and produces better results.
        - 

```
Enter your hotel review: Not very nice

Predicted Rating: 2
```

    - Lastly completely positive or negative reviews however are correctly classified and receive the correct rating. For examples:
        - 

```
Enter your hotel review: Very nice hotel, i loved the view from my room

Predicted Rating: 5
```

        - 

```
Enter your hotel review: AWFUL! I definitely hate my stay

Predicted Rating: 1
```

Conclusion

The program consistently demonstrates commendable efficacy in accurately assigning ratings to new reviews. Its robust functionality is evident across various scenarios, ensuring precise evaluations in the majority of cases. However, an area warranting attention is the nuanced challenge posed by reviews incorporating negations of words already imbued with negative meanings. In such instances, the program encounters occasional complexities, highlighting the need for further refinement in handling nuanced linguistic structures to enhance overall performance and reliability.

-This project was divided in half among the team members. This allowed us to focus on the parts we each implemented and then we integrated both parts together to achieve the complete project.-