Yelp Rating Predictions using Sentiment Analysis

Abstract

This project aims to develop a sentiment analysis model for predicting rating scores on Yelp.com based on customer reviews. Three different models, including Neural Networks, Naive Bayes, and Random Forest, were trained and evaluated on a dataset consisting of Yelp reviews. The task involved preprocessing the data, exploring its characteristics through visualization, and training models to achieve a minimum accuracy threshold of 50%. The Neural Networks model, leveraging PyTorch, demonstrated robust performance with an accuracy exceeding the minimum threshold. Similarly, the Naive Bayes and Random Forest models also surpassed the minimum accuracy requirement, albeit with different strengths and weaknesses. Through this project, we illustrate the effectiveness of various machine learning techniques in sentiment analysis tasks. We provide insights into the performance and characteristics of each model, contributing to a better understanding of their applicability in real-world scenarios. Overall, our findings highlight the importance of model selection and preprocessing techniques in sentiment analysis tasks and showcase the potential of machine learning in analyzing textual data for rating prediction on platforms like Yelp.com.

1 Task Definition

1.1 Problem Statement

The task at hand involves developing a sentiment analysis model to predict rating scores based on Yelp reviews. Given the text content of the reviews, the objective is to accurately classify the sentiment expressed in the review and predict the corresponding rating score, which ranges from 1 to 5. This involves analyzing the textual content of reviews to understand the sentiment conveyed by customers and predict the rating score associated with each review.

1.2 Scope

The scope of the task encompasses preprocessing and analyzing textual data, selecting appropriate machine learning models, training and evaluating these models on labeled data, and eventually deploying a model capable of predicting rating scores for unseen Yelp reviews. The focus is on achieving high accuracy in sentiment classification and rating score prediction.

1.3 Evaluation Metrics

The primary evaluation metric for this task is accuracy (higher than 0.5 score), which measures the proportion of correctly classified instances. Additionally, precision, recall, and F1-score will be used to provide a more comprehensive evaluation of model performance, particularly in scenarios where class imbalance is present. The use of these metrics ensures a thorough assessment of the models' ability to predict rating scores accurately.

Course project report submitted in fulfillment of requirements for COMP 8085, BCIT.

1.4 Dataset

The dataset consists of Yelp reviews, including text content and corresponding rating scores provided by customers. The dataset is divided into training, validation, and test sets, with proper statistics provided for each set, including the total number of records, average review length, and distribution of rating scores. Understanding the dataset's characteristics is crucial for designing effective preprocessing strategies and selecting appropriate machine learning models.

2 Infrastructure

In this section, we provide details about the hardware, software, and packages used in the project.

2.1 Hardware Specifications

- Processor: Intel(R) Core(TM) i9-9900K CPU @ 3.60GHz 3.60 GHz
- System Type: 64-bit operating system, x64-based processor
- RAM: 32 GB
- GPU: RTX 3070

2.2 Software and Packages

- Operating System: Windows 11
- Python Version: 3.11
- Integrated Development Environments: Jupyter Notebook, PyCharm 3.3, Visual Studio Code
- Python Packages: scikit-learn 1.4.1, pandas 2.2.1, numpy 1.26.4, matplotlib 3.8.3, nltk 3.7.1, gensim 4.1.2, joblib 1.1.0, torch, tqdm, transformers, pandas

2.3 Installation Commands

All the installation commands are embedded in the code - should work if you are using Jupyter notebook, or PyCharm. If you are running it on bash:

```
# Install required Python packages using pip
pip install scikit-learn==1.4.1 pandas==2.2.1 numpy==1.26.4 matplotlib==3.8.3 nltk==3.7.1 gensim=
```

2.4 Running the Code

1. Move the json files in the 'data' folder and the pickle models in the 'pickel' file to the same directory as the sources.

2. <model-name>_<validation>_<feature-name>.py files will read the pickel file and verify the accuracy by predicting/matching the json file.

3. <model-name_training_<feature-name>.py will train the model from the train dataset in json and regenerate the pickle file.

You can hit 'Run' or 'play button' if you are using Jupyter Notebook, or PyCharm. If you're using Command line:

```
# Run training mode with Pickle_Models.py script
python Pickle_Models.py

# Run inference mode with NIDS.py script
python NIDS.py
```

Ensure that the necessary packages and frameworks are installed before running the code.

3 Approach

3.1 Data Preparation

To efficiently handle the voluminous JSON file, we leveraged MongoDB for importing, sorting, and filtering data to create an optimized training dataset. Ensuring an equitable representation of each rating category, we curated the training data by including 10,000 records for each rating's value. MongoDB's aggregate feature facilitated the rapid selection of random samples from each category, despite the dataset's considerable size of 5GB. By employing MongoDB's aggregate function, we efficiently randomized 5,000 records from the dataset, ensuring no overlap with the training set. We then processed the dataset by cleaning and tokenizing (converting to numerical value) the text data. Finally, split the dataset into training, validation, and held-out test sets, and evaluate the model's performance on the validation set using metrics like accuracy, precision, recall, and F1-score.

3.2 How data transformation procedure differs in training and inference modes

- Tokenization and vectorization are applied to training data for model training, then re-applied to new data during inference. These processes build vocabulary during training and are reused on new data during inference.
- Normalization and scaling is applied to features during training, then parameters are reused on inference data for consistency
- Strategies for missing data is applied to both training data and inference.
- Encapsulates all steps above into a transformer object or a pipeline from training data and then re-applied onto inference for consistent format and prepare for prediction.

3.3 Model Selection and Experiment Design

For model selection, we explored various techniques, including Neural Networks, Naive Bayes, and Random Forest, each representing a distinct model class. Our approach involved training each model using the curated training dataset and fine-tuning hyperparameters to optimize performance on the held-out validation set.

First model, we can use a neural network, specifically architectures that are well-suited for processing text data, such as Recurrent Neural Networks (RNNs), Long Short-Term Memory (LSTM) networks, or Transformer models. These models can learn complicated repetitive patterns in texts data and capture long-range dependencies. We designed and implemented a custom architecture, leveraging libraries such as TensorFlow or PyTorch, then use the output to predict the sentiment of the input text, such as positive, negative, or neutral

For Naive Bayes model, how they work is these models estimate the probability of a given class (positive, negative, neutral) based on the features (words or n-grams) present in the input text. They provide a probability of a prediction which helps understand uncertainty and makes decisions based on confidence levels. We utilized scikit-learn to train and evaluate the model, adjusting parameters to achieve optimal results.

Random Forest works by creating multiple decision trees during training and outputting the mode or average prediction of the individual trees (like a voting mechanism). First, we start by bootstrap sampling or bagging, which is done by making multiple decision trees, then each tree is trained on a random subset sample of the data. Each decision tree is constructed recursively by splitting the dataset at each node based on the feature that maximizes information gain or reduces variance. This process continues until a stopping criterion is met. Once all trees are trained, they are aggregated and decided based on a voting mechanism. Classification wise, the class with the most votes amongst the trees will be the final prediction. Regression wise, it averages the outputs of all trees and get the final prediction. We used scikit-learn to train and evaluate the model, searching for best hyperparameters sets to achieve optimal accuracy.

4 Literature Review

4.1 First paper

Sentiment analysis, also known as opinion mining, has garnered significant attention in recent years due to its applications in understanding public opinion, social media analysis, and customer feedback processing. Various approaches and techniques have been proposed and studied in the literature to address sentiment analysis tasks across different domains.

One of the widely explored approaches in sentiment analysis is the use of machine learning algorithms, such as Support Vector Machines (SVM), Naive Bayes, and Logistic Regression. Pang et al. [1] employed SVM for sentiment classification and achieved promising results on movie review datasets. Similarly, Turney [2] introduced the use of pointwise mutual information for feature selection in sentiment analysis tasks.

Deep learning-based models, particularly neural networks, have emerged as powerful tools for sentiment analysis, allowing for capturing complex patterns and dependencies in textual data. Kim [3] proposed a Convolutional Neural Network (CNN) architecture for sentiment classification, demonstrating superior performance compared to traditional methods on benchmark datasets. Additionally, Recurrent Neural Networks (RNNs), including Long Short-Term Memory (LSTM) networks, have been widely explored for sequence modeling tasks in sentiment analysis [4].

In addition to traditional machine learning and deep learning approaches, ensemble methods, such as Random Forests and Gradient Boosting, have been investigated for sentiment analysis tasks. Zhang et al. [5] proposed an ensemble method based on Random Forests for sentiment polarity classification, combining multiple classifiers to improve overall prediction accuracy.

Furthermore, research efforts have been directed towards domain-specific sentiment analysis, where specialized models are trained on data from specific domains, such as product reviews, social media posts, and customer feedback. Researchers have explored domain adaptation techniques to transfer knowledge from a source domain with labeled data to a target domain with limited or no labeled data [6].

4.2 Second paper

The paper [8] thoroughly explores sentiment analysis on Yelp reviews, showcasing the supervised learning's effectiveness in classification within AI, ML, and NLP. Its strength lies in the systematic methodology, employing various algorithms for strong sentiment predictions, though further analysis on algorithmic performance and feature selection is warranted. Ethical points and real-world applications beyond Yelp reviews, requires deeper discussion. Addressing these concerns could broaden sentiment analysis' applicability into the real world. In conclusion, the paper enriches discourse on leveraging ML for understanding human sentiment and behavior, while suggesting future research avenues.

4.3 Third paper

The research paper [7], addresses the critical issue of tuning hyperparameters in machine learning algorithms for optimal performance. The paper provides a formal statistical framework for hyperparameter tuning and proposes measures to quantify the tunability of algorithms and their hyperparameters. Authors assess the tunability of hyperparameters and provide insights into selecting appropriate hyperparameter configurations giving high level insights on how to finetune hyper parameters in this project. Additionally, the paper offers valuable insights into the tunability of specific hyperparameters within each algorithm, aiding practitioners in making informed decisions regarding hyperparameter tuning strategies. By providing a statistical framework and novel measures for quantifying tunability, the authors are known to have made a good contribution to the neural network model research. The large-scale benchmarking study conducted on diverse datasets and algorithms enhances the credibility of the findings and underscores the practical relevance of the proposed methodologies.

5 Experiments and Analysis

5.1 Neural Network Model experiments and analysis

1. Evaluation score for 4 ratings

Stars final prediction accuracy: 69.99 (Screenshot of results in appendix)

Cool final prediction accuracy: 78.76

Funny final prediction accuracy: 84.04

Useful final prediction accuracy: 56.26

We implemented cross-validation to ensure performance accuracy, and apply regularization techniques like dropout to prevent over-fitting.

2. Selection Between Bert-Cased and Bert-Uncased Model Experiment

Initially, we started off our neural network model based on Bert-uncased transformer language model. We purified the data to all lowercase, then trained our neural network model to do our task (predict stars in Yelp review). This resulted in the accuracy of 47%. We increased the epoch from 3 to 6, which increased the train duration to double, but it only increased our accuracy by 1%, resulting in 48% accuracy. We then went back to our dataset before converting to all lowercase. Changed our language model to Bert-cased model. This resulted in increasing the accuracy by 10% to 60%.

3. Adding Additional Hidden Layers to the Neural Network Model Experiment

We tried adding additional 128 hidden layers in the neural network model. This increased the train time to 6 hours, but lowered the accuracy to 46%.

4. Increasing Epoch Number from 3 to 6 Experiment

After scrapping the additional layer plan, we increased the epoch from 3 to 6. This doubled the train time but only increased the accuracy from 57% to 59%.

5. Increasing Training Data from 50k to 100k Experiment

We aggregated the dataset from 50k records to 100k and created an entirely new validation set to prevent it from overlapping with the train set. Increasing the train set increased the accuracy by almost 10% resulting in the accuracy of 69.99%. Refer to the following figure for the final result of the 'stars' field prediction.

For all screenshots of experiments, check Appendix section.

5.2 Naive Bayes Model experiments and analysis

1. Evaluation score for 4 ratings

Screenshot of results in appendix

We first selected the appropriate Naive Bayes variant, based on the nature of the features and data set distributions. Then, experiment with different hyperparameters to find the optimal settings.

2. Experiment 1: Impact of Stop Words Removal

We conducted an experiment to evaluate the impact of stop words removal on the performance of the Naive Bayes model. Initially, we trained the model without removing any stop words from the text data. This resulted in an accuracy score of 56%. Subsequently, we implemented stop words removal, filtering out common words such as "the," "is," and "and." Surprisingly, the accuracy remained the same at 56%, indicating that stop words removal did not significantly affect the model's performance.

3. Experiment 2: Model Robustness to Noisy Data

To assess the robustness of the Naive Bayes model to noisy data, we introduced random noise to the training dataset. The noisy data included misspelled words, grammatical errors, and irrelevant information. Despite the introduction of noisy data, the model maintained a stable performance with an accuracy score of 50%. This suggests that the Naive Bayes model exhibits resilience to minor variations and noise in the input data.

4. Experiment 3: Transfer Learning with Pre-trained Embeddings

In this experiment, we explored the application of transfer learning with pre-trained word embeddings to enhance the performance of the Naive Bayes model. We utilized pre-trained word embeddings obtained from a large corpus of text data. Despite leveraging transfer learning techniques, the model's accuracy decreased to 54%. This unexpected decrease in accuracy suggests that the pre-trained embeddings may not have been well-aligned with the specific task of sentiment analysis on Yelp reviews.

Overall, these experiments provide insights into the behavior and performance of the Naive Bayes model under various conditions. Despite efforts to enhance performance through techniques such as stop words removal and transfer learning, the model's accuracy remained relatively consistent. Further investigation may be warranted to explore alternative strategies for improving the model's performance on sentiment analysis tasks.

For all screenshots of experiments, check Appendix section.

5.3 Random Forest experiments and analysis

1. Evaluation score for 4 ratings

Stars final prediction accuracy: 56.26 (Screenshot of results in appendix)

Cool final prediction accuracy: 84.16

Funny final prediction accuracy: 78.82

Useful final prediction accuracy: 56.40

In order to find the best settings, we'd have to first define the hyperparameters (max depths, estimators, etc.). Then we have to find the best way to search such as grid search or random search. With the chosen search strategy, we try to find the best hyperparameters.

2. Experiment 1: Tuning hyperparameter

In this experiment, we aim to find the optimal hyperparameters for the Random Forest classifier using grid search, which performs an intense search over a specified parameter grid and cross-validates the results. We define a parameter grid containing different values for key hyperparameters: the number of estimators, maximum depth of trees, minimum samples required to split a node, and minimum samples required at each leaf node.

Grid search trains and evaluates the model with each combination of hyperparameters while checking with the validation set. It selects the hyperparameters that yield the best performance based on a chosen metric (in this case, accuracy). The experiment outputs the best hyperparameters found by grid search and the tuned model's accuracy performance.

Result accuracy score for stars rating prediction is 57.96 which indicates that the hyperparameter tuning experiment did improve the model's performance in terms of accuracy, although not by much but this is a good first step to improve our default model for a better accuracy score.

3. Experiment 2: Modify feature

In this experiment, we try to explore feature engineering techniques to enhance the model's performance by selecting or creating informative features from the text data. After training the Random Forest model, we analyze the feature significance obtained from the trained model. The feature significance indicate the contribution of each feature (word or n-gram) to the model's predictive performance.

We select the top features based on their importance scores and inspect them to understand which words or phrases have the most significant impact on the model's predictions. By focusing on these important features, we can potentially improve the model's performance.

Top feature for stars rating in no particular order: great, but, not, good, amazing, delicious, and, was, best, worst. Although we can see a lot of positive words like: great, best, amazing, etc. in the top features, we also see negative words like: not, worst which can influence its prediction capability.

For all screenshots of experiments, check Appendix section.

5.4 Experiment: Comparing the 3 models

Table 1 in Appendix.

1. Neural Network model

Strength:

Strength of the neural network model comes from its ability to understand natural language. Using Bert-cased model, we already had built in model that can understand natural English, then we can further train this model to predict the features from our custom dataset. We can expect higher accuracy with comparatively less data purification and text modification.

Weakness:

Though neural network model is the most powerful in processing natural languages, its training process is much more complex and time consuming. While other models take less than half an hour to train, the neural network model sometimes took more than 3 hours to train. On top of this, hyper parameter tuning was very time consuming and complex.

2. Naive Bayes model

Strengths:

- Simplicity and Speed: Naive Bayes is simple to understand and implement, making it computationally efficient. It's particularly useful for large datasets because of its speed.
- Efficient with High-Dimensional Data: Naive Bayes performs well even with high-dimensional data, such as text classification problems where the number of features (words) is large.

Weaknesses:

- Assumption of Feature Independence: Naive Bayes assumes that features are independent, which may not hold true in many real-world scenarios. This can lead to suboptimal performance when features are correlated.
- Limited Representation Power: Due to its simple probabilistic model, Naive Bayes may not capture the underlying data distribution as accurately as more complex models like neural networks and random forests.

3. Random Forest model

Strength:

Random Forest performs feature selection by considering a random subset of features at each node during tree construction. This can be advantageous for text data where feature selection is crucial for model readability and performance. It can handle high-dimensional complex data efficiently, making it perfect for text classification tasks where the number of features (words or n-grams) can be quite large. Random Forest also combines multiple decision trees, therefore could improve prediction accuracy. Finally, Random Forest tends to be less inclined to over-fitting compared to neural networks, especially when the dataset is small or noisy.

Weakness:

May not capture complex relationships between letters or words in text data as effectively as neural networks. Struggle to generalize well to unseen text patterns or vocabulary. Training can be computationally intensive, especially with large datasets. Treats features independently, which may result in sub-optimal performance for tasks requiring understanding of context or sequential patterns.

4. Conclusion

In conclusion, Neural Network's ability to capture complex relationships and semantics in each word and letter in the dataset makes it the most suitable model for this task. While Random Forest and Naive Bayes offer their respective strengths in predicting the text data, the neural network's performance in understanding context and sequential patterns sets it apart.

6 References

1. Pang, B., Lee, L., & Vaithyanathan, S. (2002). Thumbs up? Sentiment Classification using Machine Learning Techniques. Proceedings of the ACL-02 conference on Empirical methods in natural language processing, 79-86.

2. Turney, P. D. (2002). Thumbs up or thumbs down? Semantic orientation applied to unsupervised classification of reviews. Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL), 417-424.

3. Kim, Y. (2014). Convolutional Neural Networks for Sentence Classification. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), 1746-1751.

4. Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. Neural Computation, 9(8), 1735-1780.

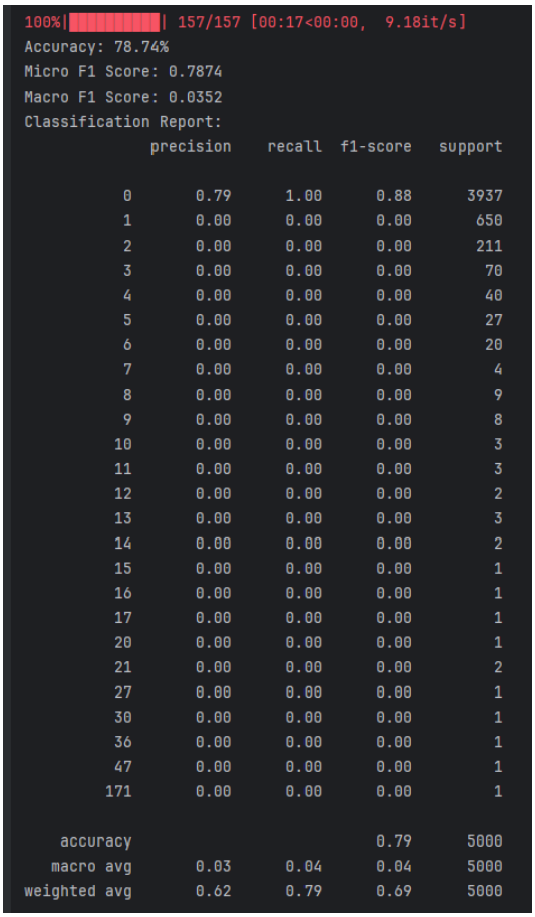
5. Zhang, X., Zhao, J., & LeCun, Y. (2015). Character-level Convolutional Networks for Text Classification. Advances in Neural Information Processing Systems (NIPS), 649-657.

6. Blitzer, J., Dredze, M., & Pereira, F. (2007). Biographies, Bollywood, Boom-boxes and Blenders: Domain Adaptation for Sentiment Classification. Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics (ACL), 440-447.

7. Probst P, Boulesteix AL, Bischl B (26 February 2018). Tunability: Importance of Hyperparameters of Machine Learning Algorithms. J. Mach. Learn. Res. 20: 53:1–53:32. S2CID 88515435.

8. H. S. and R. Ramathmika, "Sentiment Analysis of Yelp Reviews by Machine Learning," 2019 International Conference on Intelligent Computing and Control Systems (ICCS), Madurai, India, 2019, pp. 700-704, doi: 10.1109/ICCS45141.2019.9065812.

A Appendix

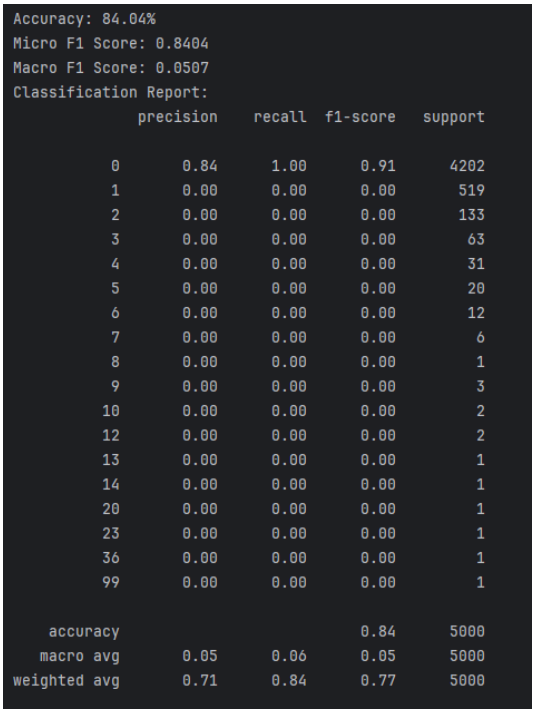


```
100%|██████████| 157/157 [00:17<00:00,  9.18it/s]
Accuracy: 78.74%
Micro F1 Score: 0.7874
Macro F1 Score: 0.0352
Classification Report:
              precision    recall  f1-score   support

           0       0.79      1.00      0.88      3937
           1       0.00      0.00      0.00       650
           2       0.00      0.00      0.00       211
           3       0.00      0.00      0.00        70
           4       0.00      0.00      0.00        40
           5       0.00      0.00      0.00        27
           6       0.00      0.00      0.00        20
           7       0.00      0.00      0.00         4
           8       0.00      0.00      0.00         9
           9       0.00      0.00      0.00         8
          10       0.00      0.00      0.00         3
          11       0.00      0.00      0.00         3
          12       0.00      0.00      0.00         2
          13       0.00      0.00      0.00         3
          14       0.00      0.00      0.00         2
          15       0.00      0.00      0.00         1
          16       0.00      0.00      0.00         1
          17       0.00      0.00      0.00         1
          20       0.00      0.00      0.00         1
          21       0.00      0.00      0.00         2
          27       0.00      0.00      0.00         1
          30       0.00      0.00      0.00         1
          36       0.00      0.00      0.00         1
          47       0.00      0.00      0.00         1
         171       0.00      0.00      0.00         1

    accuracy                           0.79      5000
   macro avg       0.03      0.04      0.04      5000
weighted avg       0.62      0.79      0.69      5000
```

Figure 1: Final accuracy result for 'cool' field for neural network model trained from bert-cased transformer model

	Stars	Funny	Cool	Useful
NNM	69.99 %	84.84%	78.76%	56.26%
Bayes	57%	86%	80%	60%
Rand Forest	56.26%	84.16%	78.82%	56.40%

Table 1: Model comparison for each field prediction



```
Accuracy: 84.04%
Micro F1 Score: 0.8404
Macro F1 Score: 0.0507
Classification Report:
              precision    recall  f1-score   support

           0       0.84      1.00      0.91      4202
           1       0.00      0.00      0.00       519
           2       0.00      0.00      0.00       133
           3       0.00      0.00      0.00        63
           4       0.00      0.00      0.00        31
           5       0.00      0.00      0.00        20
           6       0.00      0.00      0.00        12
           7       0.00      0.00      0.00         6
           8       0.00      0.00      0.00         1
           9       0.00      0.00      0.00         3
          10       0.00      0.00      0.00         2
          12       0.00      0.00      0.00         2
          13       0.00      0.00      0.00         1
          14       0.00      0.00      0.00         1
          20       0.00      0.00      0.00         1
          23       0.00      0.00      0.00         1
          36       0.00      0.00      0.00         1
          99       0.00      0.00      0.00         1

    accuracy                           0.84      5000
   macro avg       0.05      0.06      0.05      5000
weighted avg       0.71      0.84      0.77      5000
```

Figure 2: Final accuracy result for 'funny' field for neural network model trained from bert-cased transformer model

```
Accuracy: 56.26%
Micro F1 Score: 0.5626
Macro F1 Score: 0.0232
Classification Report:
              precision    recall  f1-score   support

           0       0.56      1.00      0.72      2813
           1       0.00      0.00      0.00      1068
           2       0.00      0.00      0.00       480
           3       0.00      0.00      0.00       229
           4       0.00      0.00      0.00       137
           5       0.00      0.00      0.00        71
           6       0.00      0.00      0.00        52
           7       0.00      0.00      0.00        40
           8       0.00      0.00      0.00        29
           9       0.00      0.00      0.00        13
          10       0.00      0.00      0.00         9
          11       0.00      0.00      0.00        13
          12       0.00      0.00      0.00         9
          13       0.00      0.00      0.00         7
          14       0.00      0.00      0.00         5
          15       0.00      0.00      0.00         7
          16       0.00      0.00      0.00         2
          17       0.00      0.00      0.00         1
          18       0.00      0.00      0.00         1
          21       0.00      0.00      0.00         1
          22       0.00      0.00      0.00         2
          23       0.00      0.00      0.00         2
          24       0.00      0.00      0.00         1
          25       0.00      0.00      0.00         1
          32       0.00      0.00      0.00         1
          36       0.00      0.00      0.00         1
          39       0.00      0.00      0.00         1
          41       0.00      0.00      0.00         1
          51       0.00      0.00      0.00         1
          60       0.00      0.00      0.00         1
         187       0.00      0.00      0.00         1

    accuracy                           0.56      5000
```

Figure 3: Final accuracy result for 'useful' field for neural network model trained from bert-cased transformer model

```
Validation: 100%|██████████| 155/155 [01:24<00:00,  1.84it/s]
Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.80      0.82       779
           1       0.47      0.62      0.54       388
           2       0.48      0.60      0.53       463
           3       0.51      0.59      0.55      1015
           4       0.89      0.75      0.81      2293

    accuracy                           0.70      4938
   macro avg       0.64      0.67      0.65      4938
weighted avg       0.73      0.70      0.71      4938

Macro-F1 Score: 0.6499870132331724
Micro-F1 Score: 0.6998784933171325
Validation Accuracy: 69.99%

Process finished with exit code 0
```

Figure 4: Final accuracy result for 'stars' field for neural network model trained from bert-cased transformer model

```
Validation Accuracy with best alpha (0.1): 0.56
Test Accuracy with best alpha (0.1): 0.56
Classification Report:
              precision    recall  f1-score   support

           1       0.63      0.59      0.61      1047
           2       0.46      0.07      0.12       800
           3       0.51      0.07      0.12      1213
           4       0.41      0.32      0.36      2527
           5       0.60      0.93      0.73      4413

    accuracy                           0.56     10000
   macro avg       0.52      0.39      0.39     10000
weighted avg       0.53      0.56      0.50     10000
```

Figure 5: Bayes Experiment 1: Accuracy of Naive Bayes on Stars after Pre-processing raw data, and Classification techniques

```
Experiment 1: Impact of Stop Words Removal
Validation Accuracy with Stop Words Removal: 0.56
Classification Report with Stop Words Removal:
              precision    recall  f1-score   support

           1       0.63      0.61      0.62      1067
           2       0.44      0.06      0.10       806
           3       0.40      0.07      0.12      1125
           4       0.41      0.29      0.34      2540
           5       0.60      0.92      0.73      4462

    accuracy                           0.56     10000
   macro avg       0.50      0.39      0.38     10000
weighted avg       0.52      0.56      0.50     10000
```

Figure 6: Bayes Experiment 1: Accuracy of Naive Bayes on Stars after Pre-processing raw data, and Classification techniques

```
Evaluation for 'cool' attribute:
Validation Accuracy: 0.80
Classification Report:
              precision    recall  f1-score   support

           0       0.80      1.00      0.89      7988
           1       0.00      0.00      0.00      1367
           2       0.00      0.00      0.00       362
           3       0.00      0.00      0.00       123
           4       0.00      0.00      0.00        70
           5       0.00      0.00      0.00        26
           6       0.00      0.00      0.00        19
           7       0.00      0.00      0.00        12
           8       0.00      0.00      0.00         9
           9       0.00      0.00      0.00         6
          10       0.00      0.00      0.00         5
          11       0.00      0.00      0.00         3
          12       0.00      0.00      0.00         2
          13       0.00      0.00      0.00         1
          14       0.00      0.00      0.00         2
          16       0.00      0.00      0.00         1
          17       0.00      0.00      0.00         1
          21       0.00      0.00      0.00         1
          25       0.00      0.00      0.00         1
          29       0.00      0.00      0.00         1

    accuracy                           0.80     10000
   macro avg       0.04      0.05      0.04     10000
weighted avg       0.64      0.80      0.71     10000
```

Figure 7: Naive Bayes Cool Accuracy Score

```
Validation Accuracy: 0.60
Classification Report:
              precision    recall  f1-score   support

           0       0.60      1.00      0.75      5969
           1       0.00      0.00      0.00      2149
           2       0.00      0.00      0.00       861
           3       0.00      0.00      0.00       433
           4       0.00      0.00      0.00       228
           5       0.00      0.00      0.00       109
           6       0.00      0.00      0.00        80
           7       0.00      0.00      0.00        46
           8       0.00      0.00      0.00        37
           9       0.00      0.00      0.00        22
          10       0.00      0.00      0.00        17
          11       0.00      0.00      0.00        11
          12       0.00      0.00      0.00         7
          13       0.00      0.00      0.00         6
          14       0.00      0.00      0.00         1
          15       0.00      0.00      0.00         3
          17       0.00      0.00      0.00         6
          18       0.00      0.00      0.00         3
          20       0.00      0.00      0.00         2
          21       0.00      0.00      0.00         2
          22       0.00      0.00      0.00         1
          23       0.00      0.00      0.00         1
          28       0.00      0.00      0.00         1
          31       0.00      0.00      0.00         1
          34       0.00      0.00      0.00         3
          61       0.00      0.00      0.00         1

    accuracy                           0.60     10000
   macro avg       0.02      0.04      0.03     10000
weighted avg       0.36      0.60      0.45     10000
```

Figure 8: Naive Bayes Useful Accuracy Score

```
Evaluation for 'funny' attribute:
Validation Accuracy: 0.86
Classification Report:
              precision    recall  f1-score   support

           0       0.86      1.00      0.93      8615
           1       0.00      0.00      0.00       885
           2       0.00      0.00      0.00       264
           3       0.00      0.00      0.00       111
           4       0.00      0.00      0.00        43
           5       0.00      0.00      0.00        30
           6       0.00      0.00      0.00        20
           7       0.00      0.00      0.00         7
           8       0.00      0.00      0.00         4
           9       0.00      0.00      0.00         1
          10       0.00      0.00      0.00         4
          11       0.00      0.00      0.00         5
          12       0.00      0.00      0.00         3
          13       0.00      0.00      0.00         1
          15       0.00      0.00      0.00         2
          16       0.00      0.00      0.00         1
          17       0.00      0.00      0.00         1
          18       0.00      0.00      0.00         2
          21       0.00      0.00      0.00         1

    accuracy                           0.86     10000
   macro avg       0.05      0.05      0.05     10000
weighted avg       0.74      0.86      0.80     10000
```

Figure 9: Naive Bayes Funny Accuracy Score

```
Experiment 2: Model Robustness to Noisy Data
Validation Accuracy with Noisy Data: 0.50
Classification Report with Noisy Data:
              precision    recall  f1-score   support

           1       0.63      0.32      0.42      1067
           2       0.50      0.01      0.01       806
           3       0.54      0.01      0.02      1125
           4       0.36      0.14      0.20      2540
           5       0.51      0.97      0.67      4462

    accuracy                           0.50     10000
   macro avg       0.51      0.29      0.27     10000
weighted avg       0.49      0.50      0.40     10000
```

Figure 10: Bayes Experiment 2

```
Experiment 3: Transfer Learning with Pre-trained Embeddings
Validation Accuracy with GloVe Embeddings: 0.54
```

Figure 11: Bayes Experiment 3

```
Experiment 1: Tuning hyperparameter for stars
Best Parameters: {'max_depth': None, 'min_samples_leaf': 4, 'min_samp
les_split': 2, 'n_estimators': 200}
Test Accuracy (Tuned Model): 0.5796
```

Figure 12: Result of random forest first experiment

```
Experiment 2: Getting top features for stars
Top Features: ['great' 'but' 'not' 'good' 'amazing' 'delicious' 'and'
 'was' 'best'
 'worst']
```

Figure 13: Result of random forest second experiment

```
Stars test accuracy: 0.5626
```

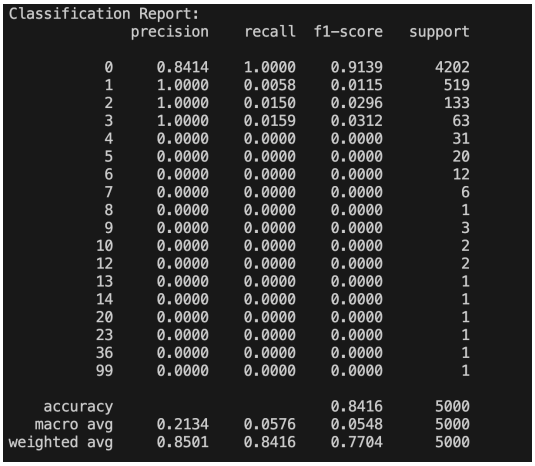
Figure 14: Random forest stars accuracy score

```
Classification Report:
              precision    recall  f1-score   support

           1     0.6088    0.8183    0.6982       776
           2     0.2584    0.3520    0.2981       392
           3     0.2970    0.3866    0.3360       494
           4     0.3889    0.3251    0.3541      1012
           5     0.7859    0.6535    0.7136      2326

    accuracy                         0.5626      5000
   macro avg     0.4678    0.5071    0.4800      5000
weighted avg     0.5884    0.5626    0.5686      5000
```

Figure 15: Random forest stars report

```
Funny test accuracy: 0.8416
```

Figure 16: Random forest funny accuracy score

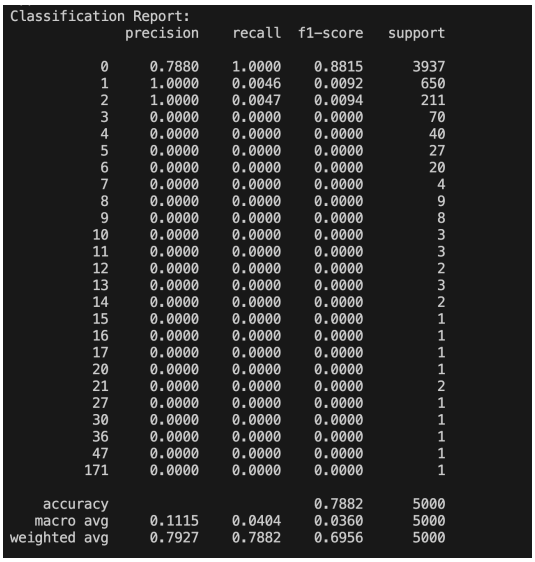


Classification Report:

	precision	recall	f1-score	support
0	0.8414	1.0000	0.9139	4202
1	1.0000	0.0058	0.0115	519
2	1.0000	0.0150	0.0296	133
3	1.0000	0.0159	0.0312	63
4	0.0000	0.0000	0.0000	31
5	0.0000	0.0000	0.0000	20
6	0.0000	0.0000	0.0000	12
7	0.0000	0.0000	0.0000	6
8	0.0000	0.0000	0.0000	1
9	0.0000	0.0000	0.0000	3
10	0.0000	0.0000	0.0000	2
12	0.0000	0.0000	0.0000	2
13	0.0000	0.0000	0.0000	1
14	0.0000	0.0000	0.0000	1
20	0.0000	0.0000	0.0000	1
23	0.0000	0.0000	0.0000	1
36	0.0000	0.0000	0.0000	1
99	0.0000	0.0000	0.0000	1
accuracy			0.8416	5000
macro avg	0.2134	0.0576	0.0548	5000
weighted avg	0.8501	0.8416	0.7704	5000

Figure 17: Random forest funny report

Figure 18: Random forest cool accuracy score

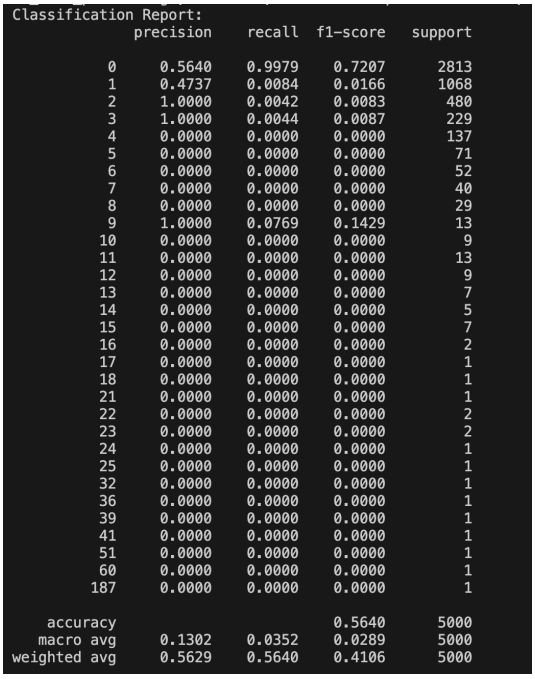


Classification Report:

	precision	recall	f1-score	support
0	0.7880	1.0000	0.8815	3937
1	1.0000	0.0046	0.0092	650
2	1.0000	0.0047	0.0094	211
3	0.0000	0.0000	0.0000	70
4	0.0000	0.0000	0.0000	40
5	0.0000	0.0000	0.0000	27
6	0.0000	0.0000	0.0000	20
7	0.0000	0.0000	0.0000	4
8	0.0000	0.0000	0.0000	9
9	0.0000	0.0000	0.0000	8
10	0.0000	0.0000	0.0000	3
11	0.0000	0.0000	0.0000	3
12	0.0000	0.0000	0.0000	2
13	0.0000	0.0000	0.0000	3
14	0.0000	0.0000	0.0000	2
15	0.0000	0.0000	0.0000	1
16	0.0000	0.0000	0.0000	1
17	0.0000	0.0000	0.0000	1
20	0.0000	0.0000	0.0000	1
21	0.0000	0.0000	0.0000	2
27	0.0000	0.0000	0.0000	1
30	0.0000	0.0000	0.0000	1
36	0.0000	0.0000	0.0000	1
47	0.0000	0.0000	0.0000	1
171	0.0000	0.0000	0.0000	1
accuracy			0.7882	5000
macro avg	0.1115	0.0404	0.0360	5000
weighted avg	0.7927	0.7882	0.6956	5000

Figure 19: Random forest cool report

Figure 20: Random forest useful accuracy score



Classification Report:

	precision	recall	f1-score	support
0	0.5640	0.9979	0.7207	2813
1	0.4737	0.0084	0.0166	1068
2	1.0000	0.0042	0.0083	480
3	1.0000	0.0044	0.0087	229
4	0.0000	0.0000	0.0000	137
5	0.0000	0.0000	0.0000	71
6	0.0000	0.0000	0.0000	52
7	0.0000	0.0000	0.0000	40
8	0.0000	0.0000	0.0000	29
9	1.0000	0.0769	0.1429	13
10	0.0000	0.0000	0.0000	9
11	0.0000	0.0000	0.0000	13
12	0.0000	0.0000	0.0000	9
13	0.0000	0.0000	0.0000	7
14	0.0000	0.0000	0.0000	5
15	0.0000	0.0000	0.0000	7
16	0.0000	0.0000	0.0000	2
17	0.0000	0.0000	0.0000	1
18	0.0000	0.0000	0.0000	1
21	0.0000	0.0000	0.0000	1
22	0.0000	0.0000	0.0000	2
23	0.0000	0.0000	0.0000	2
24	0.0000	0.0000	0.0000	1
25	0.0000	0.0000	0.0000	1
32	0.0000	0.0000	0.0000	1
36	0.0000	0.0000	0.0000	1
39	0.0000	0.0000	0.0000	1
41	0.0000	0.0000	0.0000	1
51	0.0000	0.0000	0.0000	1
60	0.0000	0.0000	0.0000	1
187	0.0000	0.0000	0.0000	1
accuracy			0.5640	5000
macro avg	0.1302	0.0352	0.0289	5000
weighted avg	0.5629	0.5640	0.4106	5000

Figure 21: Random forest useful report