­­Shangmin Chen

CS506

**Submission Model (KNN)**

The dataset used for this competition was skewed towards the ratings with a higher score, with an overwhelming majority being 5-star reviews. This skew led to challenges in model performance, especially with a K-Nearest Neighbors (KNN) model, which relies on neighborhood density for predictions. The starter code did not prepare anything preprocessing steps to address this imbalance, meaning the model inherently leaned toward predicting the dominant class—5-star ratings. As shown in the dataset histogram, 5-star reviews vastly outnumbered other ratings, which impacted the distribution of neighbors in KNN. The clustering effect toward 5-star ratings is evident in the confusion matrix, where the model overwhelmingly predicted higher ratings regardless of true labels. This imbalance issue substantially limited KNN’s ability to capture nuanced patterns among less frequent ratings, such as 1- and 2-star reviews.

A graph with blue squares

Description automatically generatedA graph of a number of different colored squares

Description automatically generated with medium confidence

To address this issue, the solution would be to manipulate the dataset such that the dataset would be spread out evenly uniformly. To do this, we could under sample the dataset to eliminate the bias towards 4-star and 5-star ratings. One major shortcoming of KNN is that the model is ineffective when encountering a dataset with high dimensions. In this case, our dataset has a lot more than just low-dimensional data like “Helpfulness”. Features like “Text” and “Summary” are extremely valuable when evaluating the potential score, however, combination of all those features are not KNN’s strong suit, because KNN relies on distance-based calculations in feature space. Text data, like “Text” and “Summary,” even when vectorized, creates high-dimensional representations that KNN struggles to handle effectively. As a result, KNN would not be a suitable model to approach this dataset.

**Attempt to improve**

To improve upon the initial KNN approach, I introduced a Random Forest model, which better handles complex, high-dimensional data and is less sensitive to class imbalance. The Random Forest model leverages an ensemble of decision trees, making it more robust in capturing non-linear relationships and providing more reliable predictions across classes compared to KNN. To further enhance predictive accuracy, I experimented with combining both KNN and RF in an ensemble method. This ensemble approach aimed to balance the strengths of each model: KNN’s simplicity for straightforward cases and RF’s power for more intricate patterns.

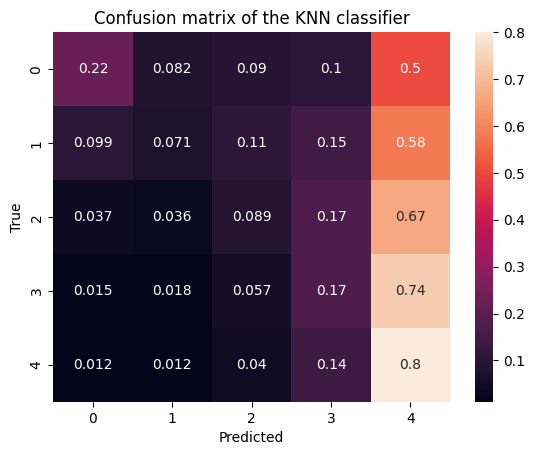
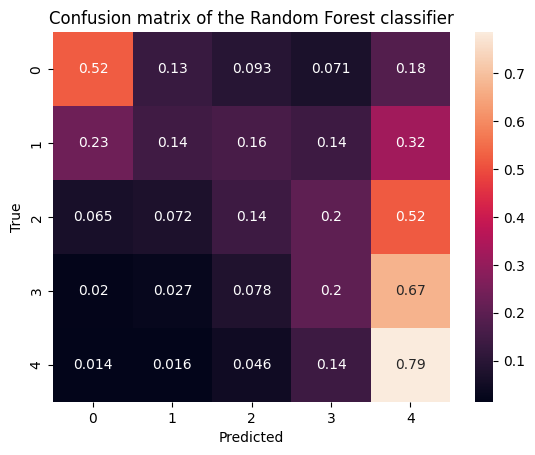
However, this approach led to significant runtime challenges due to the computational demands of training multiple models and managing the ensemble. To be specific, when trying to implement scikit’s TfidfVectorizer[[1]](#footnote-1) to turn “Text” and “Summary” into feature vectors, the cell that contained that segment of code ran over night to no avail. To address this, I had an idea to utilize multithreading to parallelize this process, I implemented, joblib[[2]](#footnote-2), joblib creates a pool of workers that process portions of your data independently and simultaneously. Each worker processes a chunk of text data, applying transformations like tokenization and vectorization. With the help of multithreading and cutting down the overall dataset, I was able to efficiently use 6 cores to reducing runtime to under a single minute and enabling the model to handle larger data subsets more efficiently. Threading allowed each model to run in parallel, making the ensemble training feasible within the time constraints of the competition. Another parallelization library I used is called pandarallel[[3]](#footnote-3). Pandarallel is a simple and efficient tool to parallelize Pandas operations on all available CPUs. Using this approach significantly sped up the process of applying DataFrame operations, enables parallelized sentiment analysis that would otherwise take a considerable amount of run time. There were also attempts to independently try to manipulate the data to increase the efficiency of the program. An example was when I explored the possibility of using sparse matrices instead of dense matrices. The reason for this is that while dense matrices work better with Pandas for extensive data manipulation, sparse matrices are more efficient when there isn’t a lot of manipulation needed, as they consume less memory and handle large, sparse datasets more effectively.

In addition to runtime issues, I encountered several type errors during the implementation. These errors stemmed from inconsistent data types and null values in the dataset, which disrupted model training. To resolve these, I conducted thorough data cleaning, replacing or imputing missing values and ensuring consistent data types across all features. This step involved explicit type casting and adjustments to the preprocessing pipeline, which ultimately stabilized the models and improved performance.

As for feature engineering, I wanted to enhance the model’s ability to capture useful patterns from the data, I did so by adding new features that provided more context about each review. Specifically, on top of the Helpfulness feature derived from the ratio of HelpfulnessNumerator to HelpfulnessDenominator, which helped quantify how other users rated the helpfulness of each review, I also extracted temporal information by creating year and is\_weekend features based on the Time column, where year captures the year of the review, and is\_weekend identifies if the review was posted on a weekend. Furthermore, I included sentiment scores from the Summary and Text fields, using the TextBlob library to assess the sentiment polarity of each text. These features helped the model interpret both the timing and sentiment of reviews, providing a more nuanced understanding of user feedback.

Ultimately, I encountered an issue due to unfamiliarity with the format of Kaggle submissions. To quickly train my model, I reduced the dataset size by 50%. However, due to a flawed implementation, the resulting submission.csv file ended up with only 50% of the required rows. Given more time, I would ensure that the output dimensions remain consistent throughout the process, particularly in the final submission file, to meet Kaggle’s requirements accurately.

In the confusion matrix for the ensemble model, we observe improved accuracy compared to the initial KNN submission, particularly in identifying ratings that are not 5 stars. For instance, 1-star reviews achieved a notably higher accuracy, showing the ensemble’s capability to better distinguish between different ratings than the original KNN model. This improvement highlights that combining KNN and Random Forest leveraged each model’s strengths, resulting in more balanced and accurate predictions across classes. This performance boost demonstrates the effectiveness of the ensemble approach over the standalone KNN model initially submitted.

1. https://scikit-learn.org/1.5/modules/generated/sklearn.feature\_extraction.text.TfidfVectorizer.html [↑](#footnote-ref-1)
2. https://joblib.readthedocs.io/en/stable/generated/joblib.Parallel.html [↑](#footnote-ref-2)
3. https://github.com/nalepae/pandarallel [↑](#footnote-ref-3)