Binary Prediction of Food Rating Using Logistic Regression

CSE 158/258: Web Mining and Recommender Systems
UC San Diego

ABSTRACT - Cooking is an activity done in all parts of the world, often associated with a healthier lifestyle and sometimes a hobby in itself. As it is always prevalent in the majority's everyday lives, in this paper, we study a predictive task centered around a kaggle dataset related to cooking and build a variety of models, eventually leading to a logistic regression model for binary prediction.

I. INTRODUCTION

Our goal is to predict the user's rating on a specific recipe. We have two predictive tasks both aim to predict the rating a specific user gave to a recipe with different inputs. The first task is to predict the rating using the user's review text and Natural Language Processing (NLP), while the second manages to predict by only knowing user ID and recipe ID. The model that achieved 84% accuracy for the first predictive task was logistic regression using a bag of words. Other models such as non-parametric tree based models (Decision Tree Classification, Gradient Boosting, XGBoost, etc) did not perform well because the severe class imbalance led to overfitting.

II. DATA SET

For this project, a kaggle dataset related to cooking was used. The dataset is constructed to be used in the paper "Generating Personalized Recipes from Historical User Preferences"[5]. It features tables that include users and their reviews of specific recipes, the recipe names, description, steps of preparation, and list of ingredients for each recipe. The abundance of text data is influential in orienting the mostly NLP-centered direction of this project. Furthermore, it was hoped that using a cooking related text recommendation dataset would have a lesser problem of encountering distracting proper nouns compared to other text-based recommendation datasets - like fiction. This will lead to a better desired effect of using an NLP-centered model. Additionally, the dataset is large enough with 1,132,367 observations.

However, there are several potential biases for this dataset. If the review text column was being used for example to predict ratings for example, a potential issue is that the data is only of the users who were willing to write a review for it, let alone rate it. Another bias is with the ratings itself: The multinomial distribution of these ordinal ratings are severely skewed to higher ratings - 73% of reviews give 5-stars ratings, 16% give 4 star ratings, while the remaining 11% of reviews are 3 and below (Fig. 1). There are some possible reasons behind this, and my guess is that people tend to not cook things that they expect to not like. People might also be preferentially biased towards their own cooking skills. It would also call into question how sincere people are when they rate a recipe 4 stars or above, since it could also be from politeness on not giving a bad rating, or being too lazy to give a thoughtful and honest rating.

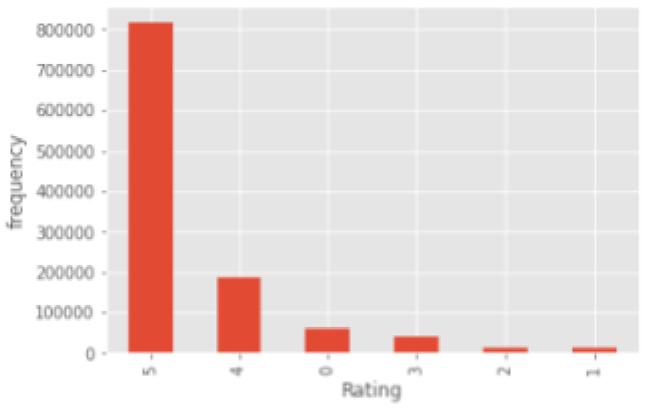

Fig. 1. Rating Distribution

III. PREDICTIVE TASK

The task is to predict the recipe's ratings, using a given user's review of a recipe as text data for NLP. Since the percent of good ratings in the data is so high, we modified the ratings with binary classification. We consider a rating that has value less than or equal to three to be in category 1, and anything above three to be in category 2 (3 out of 5 stars and below = category 1; anything above 3 out of 5 stars = category 2). Basically, we are looking for bad reviews instead of good reviews. This way is intended to address the underrepresentation of bad reviews, as described above: if classification was the goal, it would be more effective to find inferences to identify the minority outlier category, rather than the obvious majority. Thus all feature columns that we must use, should be directed towards being positively correlated with the incidents of bad to subpar reviews. We would compare our validity and accuracy to that of a baseline model. This baseline model will simply be a predictor that utilizes the historical ratings of the user in question. We would assign 1 in the case of a user having a mean rating greater than 3 and 0 otherwise.

IV. **MODELS**

The current working model for Task A uses logistic regression for binary prediction. The input data table's features is a bag of words set-up: 100 columns for count vectorization from a set of 100 specific words. Assuming the feature engineering was done correctly, these 100 words in particular should ideally be able to uniquely target and identify if a review has "negative sentiments". Non-parametric tree based models such as Decision Tree Classification, Gradient Boosting, and XGBoost did not perform well because the severe class imbalance led to overfitting and therefore poor accuracy.

These are specific steps with data preparation, and feature engineering that led to getting these 100 words are as follows:

1. Do preprocessing on the review text column like lowercasing, and replacement of non alphanumeric characters with whitespace.
2. TF-IDF vectorization on the processed review text column, but before doing so, prepare the stop words list.
3. Use NLTK's English stop-words list, but deliberately throw out any "negative words" inside it manually (like "no", or "weren't"). The reviews text contains a lot of otherwise "emotion-less" or neutral cooking-related words (like potato) or even proper words (like Mediterranean) that otherwise contaminate the reviews column as data noise. Conveniently, the raw recipes CSV's ingredients column contains enough of these potentially distracting words all in 1 place, so these words are added too to the stop words list.
4. Performing the TF-IDF vectorization on the review text column with the stopwords should produce a matrix with column count as the vocabulary size. Group the review rows of the new TF-IDF matrix separately by the corresponding binary rating category. Then within each group, collapse the rows with a column-wise mean (i. e. np.mean with axis = 0). So the resulting new array has a row count of 2 and a column count of vocabulary size.
5. Do a column-wise argmax (i. e. axis = 0) on the new array, and filter only for the columns where the argmax row index corresponds to the category for bad ratings. Use these column indices and the vocabulary mapping of the saved TF-IDF vectorizer object to retrieve the words.
6. The 2 previous steps basically filtered for the words that have a stronger "belongingness" to bad reviews compared to good reviews, using TF-IDF. But now, we should trim down these words to just 100. While all of these words have stronger relevance to bad reviews, some are stronger than others, so these words are sorted by the largest absolute difference (between the 2 rows), and the topmost 100 are picked.
7. These 100 words came out to be:
    a. 'sorry', 'bland', 'like', 'ok', 'would', 'taste', 'maybe', 'try', '039', 'didn', 'okay', 'something', 'think', 'wasn', 'wrong', 'bad', 'thought', 'care', 'disappointed', 'don', 'though', 'found', 'quot', 'tried', 'won', 'tasted', 'however', 'know', 'work', 'needs', 'much', 'sounds', 'nothing', 'missing', 'br', 'perhaps', 'reviews', 'probably', 'pretty', 'turn', 'ended', 'seemed', 'directions', 'rating', 'give', 'rate', 'texture', 'followed', 'may', 'waste', 'might', 'guess', 'expected', 'terrible', 'unfortunately', 'awful', 'help', 'horrible', 'felt', 'ingredients', 'expecting',

'anyone', 'kind', 'say', 'could', 'see', 'add', 'us', 'adding', 'enough', 'mushy', 'looks', 'either', 'lacking', 'worst', 'runny', 'cook', 'hoping', 'mess', 'excited', 'watery', 'lacked', 'gt', 'consistency', 'disappointing', 'trying', 'idea', 'lt', 'overall', 'weren', 'rather', 'tasteless', 'isn', 'supposed', 'still', 'soggy', 'instructions', 'read', 'seems', 'calls'.

8. Finally, the training data's review text column was transformed with a count vectorizer with the vocabulary set to the 100 words.

Apart from this model were other failed models, which either were modifications of the current one, or completely different models, data preparation, and feature engineering altogether. These included the following:

- One of the older models follows the exact same plan as the current working version, but the main difference was how the matrix with 2 rows and vocabulary-size columns was made. Rather than applying TF-IDF vectorization to the review text columns and getting the group-wise column averaging, the whole text review column would have been combined into 2 "mega strings," according to category, and then TF-IDF vectorization would have been applied. This idea was scrapped because the operation to combine the sentences was computationally inefficient and frequently crashed the notebook.
- An even older model for rating prediction which uses an entirely different feature engineering set-up, which makes use of gensim's Word2Vec. 2 lookup tables would be created: one that mapped each user id to a list of every recipe id that they've ever reviewed; and another table that mapped every recipe id to a list of every user id that reviewed it. Both tables would be converted into 2 Word2Vec matrices (with the list of ids in the value being the sentence of words), which creates vector embeddings of both the recipe id's and the user id's. They would get re-joined according to the interaction data, which has 2 foreign keys for the user id and recipe id. The final data table (which has a column count of user id amount + recipe id amount), would be put through Principal Component Analysis (PCA), before being put into Logistic Regression. The model was not good at all (50% accuracy, no better than just random prediction).
- Bayes Personalized Ranking (BRP): This model is used to predict the binary rating the user gave to the recipe that they have not interacted with before. There are two versions of binary rating setup:
    - Using the value of 3 as the rating threshold. That means we treat the user/recipe interaction as positive when it has a rating larger than 3 and as negative otherwise.
    - Replacing each user's rating with value from subtracting the rating by that user's mean rating, then using a value of 0 as the threshold.

  Doing so had expectations of a better performance by making a user's ratings subjective and based on a personalized rescaling. However, the accuracy achieved from both setups above are approximately 0.52. This is because BPR mostly deals with implicit data, where the data only tells us what users have been interacting with instead of what users dislike. Therefore, our binary setup does not quite fit into this model. Specifically, our definition of a negative sample is completely different from the negative sample treated in the BPR model.
- Latent Factor Model: This model is used to work with the User-Recipe matrix of ratings (says R) that contains many incomplete data entries. To get rid of this, we use matrix factorization:

  $$R = X \cdot Y^T$$

  Where R is the rating matrix, X is user by factors matrix, and Y is the recipe by factors matrix.

Hence, the matrix multiplication satisfies R is the users by recipe matrix. The dimension of the factors, says k, is one of the hyperparameters that represent the latent factors to predict ratings. Let row i^{th} in X be γ_i and column j^{th} in Y^T be γ_j. Then we have the following:

$$R_{ui} = \gamma_i \cdot \gamma_j$$

Add this to the model with bias term $f(u, i) = \alpha + \beta_u + \beta_i$ (where β_u tells how much this user tends to rate thing above mean, and β_i tells how much this item tends to receive higher ratings than others), then we have:

$$f(u, i) = \alpha + \beta_u + \beta_i + \gamma_i \cdot \gamma_j$$

And the optimization problem becomes:

$$argmin_{\alpha,\beta,\gamma} \sum_{u,i} (\alpha + \beta_u + \beta_i + \gamma_u \cdot \gamma_i - R_{ui})^2 + \lambda(\sum_u \beta_u^2 + \sum_i \beta_i^2 + \sum_i ||\gamma_i||^2 + \sum_i ||\gamma_i||^2$$

The MSE and accuracy after running the model on the test dataset is 1.73 and 0.94 respectively. However, the confusion matrix (Fig. 2) is not reasonable since the model fails to predict true negative, which is our goal.

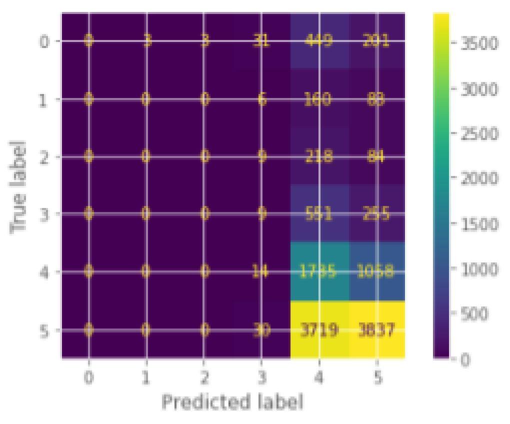

Fig. 2. Confusion Matrix

V. LITERATURE

Looking at other literature, it is important to understand the specificity of the dataset in question first. The dataset came from Food.com, and was on Kaggle. It consists of 231,637 recipes and 1,132,367 recipe reviews covering 18 years of user interactions and uploads on Food.com.

According to the code and discussion section, some people have used it as a dataset to learn machine learning or make a recommendation. Additionally, it was created to be used in a paper titled "Generating Personalized Recipes from Historical User Preferences" by Bodhisattwa Prasad Majumder, Shuyang Li, Jianmo Ni, and Julian McAuley. Similar datasets include reviews from restaurants in Seoul that were used in "Rating and Comments Mining Using TF-IDF and SO-PMI for Improved Priority Ratings"[1]. As the title suggests, they tried to make a predictor with TF-IDF and SO-PMI regarding user intentions.

Another similar dataset is the one used in "Recurrent Recommender Networks"[2]. They used Netflix contents to make a recommender with a Long Short-Term Memory (LSTM) autoregressive model in addition to a lower level factorization. With this in mind, the high level techniques used to study data like this include TF-IDF has been used in BigTech. For instance, Netflix[3] and Microsoft[4] have used it. Additionally, LSTM is now used for dynamic data as well as Word2Vec and NLTK stopwords. After comparing our results with those of existing work, we find that our findings are similar. As you might know, even though TF-IDF, standing for term frequency–inverse document frequency, is simple, it is an efficient way to reflect how important a word is to a document in a collection or corpus.

For our models corresponding to our predictive task, we use the following methodologies:

- Count Vectorization/"Bag of Words"
- TF-IDF Vectorization:

$$TF = \frac{Number\ of\ times\ the\ term\ appears\ in\ the\ document}{total\ number\ of\ terms\ in\ the\ document}$$

$$IDF = log(\frac{number\ of\ the\ documents\ in\ the\ corpus}{number\ of\ documents\ in\ the\ corpus\ contain\ the\ term})$$

$$TF\text{-}IDF = TF * IDF$$

- Logistic Regression

  $F(x) = S(X^T w + b)$, where

  $S = \frac{1}{1+e^{-x}}, X \subseteq R^{N \, x \, 100}, W \subseteq R^{100}$
- NLTK Stopwords
- Decision Tree Classification
- Principal Component Analysis (PCA)
- TSNE and Spectral Embedding
- K means Clustering and DBSCAN Clustering
- Word2Vec by Gensim
- Doc2Vec by Gensim
- Cosine Similarity
  - $Cos(x, y) = \frac{x \cdot y}{||x|| * ||y||}$
- Bayesian Personalized Ranking
- Latent Factor Model

VI. RESULTS AND CONCLUSION

The model that we chose to build for the predictive task performed at a decent accuracy of 78-79%. Its original accuracy was good, but its confusion matrix (Fig. 2.) is poor due to the class imbalance, resulting in a high false negative.

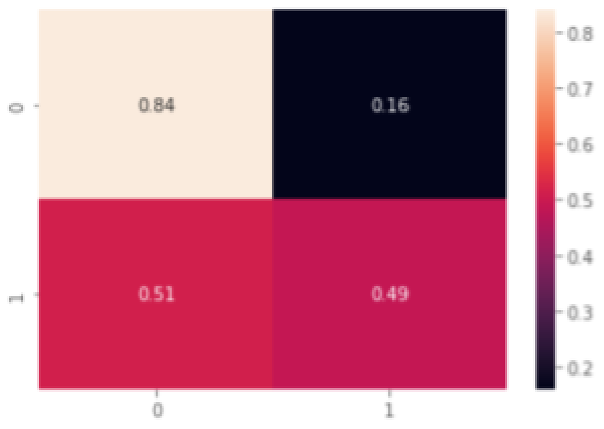

Fig. 3. Confusion Matrix

However, changing the sampling procedure with repeated undersampling, the confusion matrix improved by a +10% discrepancy between the correct vs wrong category, with the precision for category 1 being improved, as shown below:

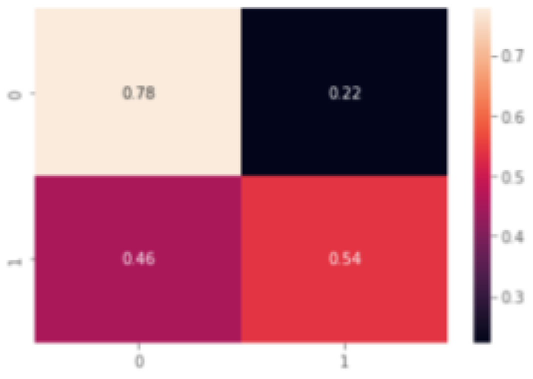

Fig. 4. Confusion Matrix

Next, we show a bar plot of the top 20 highest coefficients and their corresponding feature column words:

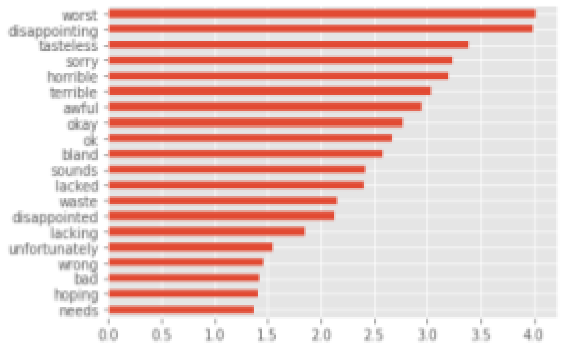

Fig. 5. Coefficients and their corresponding feature column words

The TF-IDF procedure during feature engineering proved to be successful in retrieving “unpleasant words”. Most of them are negative words, but some words that could be arguably not negative, meaning positive or just neutral, have nonetheless appeared (“okay”, “ok”, “watery”, “sounds”, “hoping”). Meanwhile, below are the lowest 20 coefficients and their corresponding words:

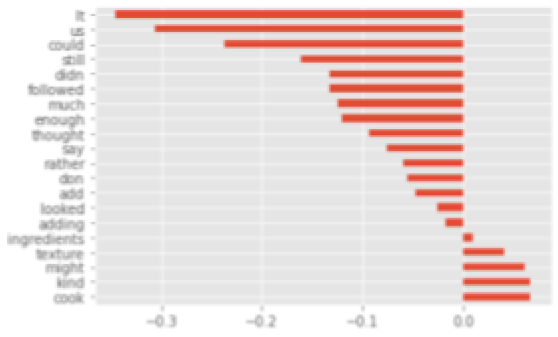

Fig. 6. Lowest 20 coefficients and their corresponding words

In conclusion, the precision still needs improvement: for category 1, it could be that the reviews that are negative lack any of the 100 words, meaning it is filtered out and deemed as part of the positive reviews. On the other hand, the positive reviews contain words that are in the 100 word set and are thus included in. N-grams are an option to improve the model without radically changing things, but at this point, there is likely diminishing returns in terms of bias/variance for which using word count frequency is no longer sustainable, and new ways must be done to improve precision beyond just the review text column. For example, perhaps the user who posted the recipes may themselves may overall have a good/bad quality.

VII. REFERENCES

[1] Kim, Jinah, and Nammee Moon. "Rating and comments mining using TF-IDF and SO-PMI for improved priority ratings." *KSII Transactions on Internet and Information Systems (TIIS)* 13.11 (2019): 5321-5334.

[2] Wu, Chao-Yuan, et al. "Recurrent recommender networks." *Proceedings of the tenth ACM international conference on web search and data mining.* 2017.

[3] Chiny, Mohamed, et al. "Netflix Recommendation System based on TF-IDF and Cosine Similarity Algorithms." *no. Bml* (2022): 15-20.

[4] Yunanda, Gisela, Dade Nurjanah, and Selly Meliana. "Recommendation system from microsoft news data using tf-idf and cosine similarity methods." *Building of Informatics, Technology and Science (BITS)* 4.1 (2022): 277-284.

[5] Generating Personalized Recipes from Historical User Preferences
Bodhisattwa Prasad Majumder*, Shuyang Li*, Jianmo Ni, Julian McAuley *EMNLP*, 2019