Sentiment, Rating and Similarity Prediction using Recipe and Interaction Data

1. INTRODUCTION

Taking the Food.com Recipe and Review dataset, we experiment with multiple methods to predict user ratings for various food items. Additionally, we explore ways to predict recipe similarity and perform sentiment analysis. We use a user-item interaction dataset to analyze the kinds of the recipes a user has tried/rated and then recommend recipes they would like to try. Due to the extensive nature of the dataset, we explore various features and methods to perform above tasks. Section 2 details the dataset analysis. Section 3 talks about the predictive tasks, while Section 4 explains the models we've used for our tasks. Section 5 gives a brief overview of the literature survey and Section 6 mentions our evaluation and results.

2. DATA ANALYSIS

2.1 Dataset Understanding and Statistics

The dataset primarily consists of two main csv files :

1. **RAW_interactions.csv**: this file consists of 1.1M interactions between users and items. Each interaction contains 5 features [user_id, recipe_id, date, rating, review]. Base statistics for this dataset are as follows
   - Number of Users: 226570
   - Number of Recipes: 231637
   - Number of Interactions: 1132367
   - Range of Rating: 1 to 5
2. **RAW_recipes.csv**: this file consists of 230K recipes, each having 12 features [name, recipe id, minutes, contributor_id, date, tags, nutrition, n_steps, steps, description, ingredients, n_ingredients]. Base statistics for this dataset are as follows
   - Number of Recipes: 231637
   - Number of Tags: 572
   - Number of Nutrition Types: 7
   - Range of N_steps: 0 to 145
   - Range of N_ingredients: 1 to 43

   For the nutrition features, we have 7 different types [calories, sugar, protein, total fat, carbohydrates, sodium and saturated fat]. To get their statistics, we plot boxplots as shown in Figures 1 and 2. Theses plots are obtained after filtering out the extreme outliers. The horizontal line in each plot denotes the mean of the feature value, the bounds of upper and lower boxes signify the interquartile bounds

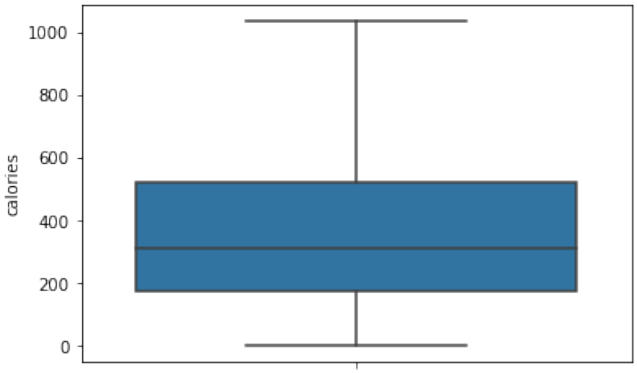

Figure 1. Distribution of Calories in the recipes

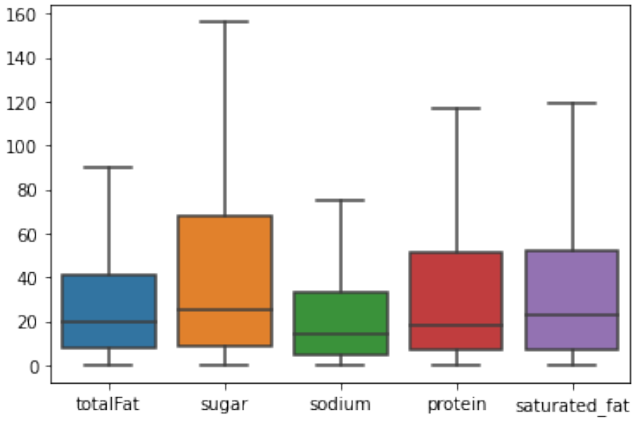

Figure 2. Distribution of Nutritions in the recipes

This dataset is large and comprehensive. We sift through it to get an appropriate subset of features required for our tasks. The EDA and preprocessing required for the same are mentioned in the next subsection

2.2 Data Preprocessing

2.2.1 Exploratory Data Analysis

In the RAW_interactions data, we primarily analyze how ratings are distributed. To that end, we look at its temporal dependency and popularity.

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

- **Number of Occurrences of Ratings vs Rating value**: We begin with a simple analysis of how are the rating values distributed i.e what is the total number of each rating (5 to 0). This helps us in understanding which are the more popular ratings. Figure 3 shows a plot for the same.

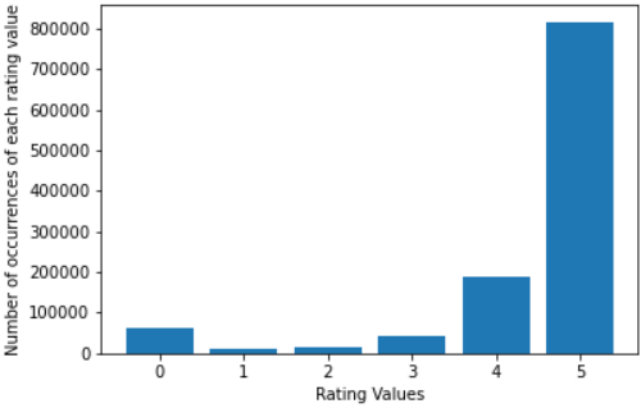

Figure 3. Distribution of Number of Occurrences of each Rating Value

- **Interactions vs Year**: Here we looked at how the number of interactions differed depending on years. It is observed most of the interactions were done in the years from 2007 to 2009. This possibly tells us that more users were actively rating food items in those years, whereas the number has decreased drastically in recent years (2016 onwards). Figure 4 shows the plot for this analysis

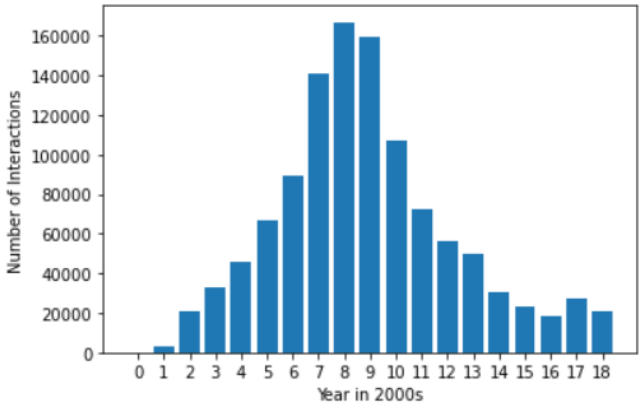

Figure 4. Distribution of Number of Interactions per Year

- **Number of 0 and 5 Ratings vs Year**: After analysing number of interactions per year, we move on to analyze number of 0 and 5 ratings per year. This is done to see if there is a year dependent pattern in user ratings. We see that the plot for rating 5 follows a trend similar to Figure 4 and avoid pasting its snippet here. Figure 5 shows the plot for rating 0 and it is slightly different trend.

In the RAW_recipes data, we first explore the relation between interactions and feature values. One particular feature value of importance is number of ingredients. More the number

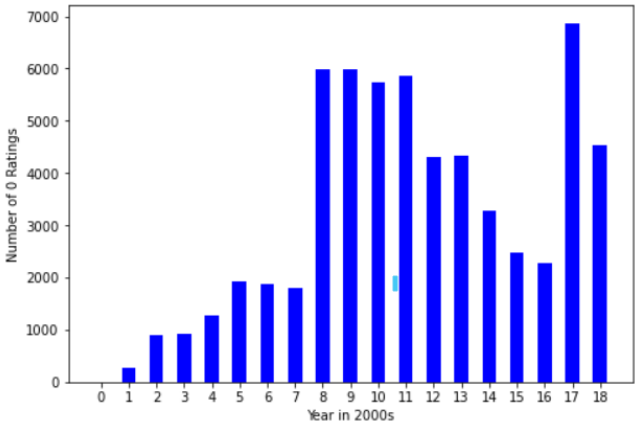

Figure 5. Distribution of Number of 0 ratings per Year

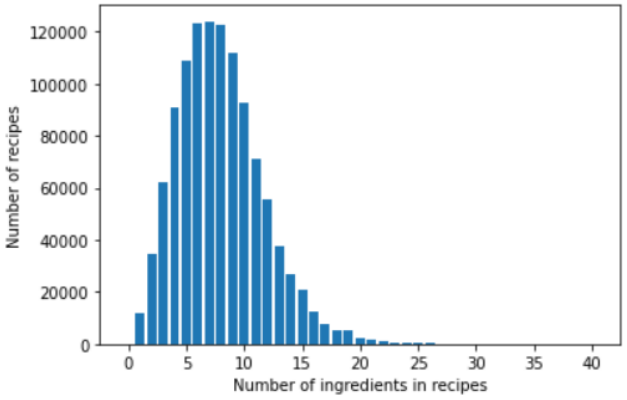

Figure 6. Number of Interactions vs Number of Ingredients in Recipes

of ingredients, more the "complexity" of the recipe. It could be possible that people interact with recipes of some average complexity. The plot in Figure 6 shows that most people prefer rating (probably highly) recipes with 5-10 ingredients. This makes a valuable metric to prune out the recipes with higher or lower ingredients.

We also study the features in the RAW_recipes dataset to find a good measure of similarity among recipes. This dataset has many features. Picking the appropriate ones is a task in itself. To explain how we go about it, we show an example of finding n_step similarity and calorie content similarity.

- **N_step similarity measure**: For each user, we calculate the standard deviation of the N_steps for all the recipes the user has interacted with. This parameter represents the number of steps required to prepar the recipe. If the standard deviation is high then one could *possibly* reason that the user doesn't interact with recipes with similar N_step values. On the other hand if the standard deviation is low then one could reason that user interacts with recipes with similar N_step values. In the latter case, N_steps could be a good similarity measure between recipes to recommend to that

user. A plot of N_step standard deviation can be found in Figure 7.

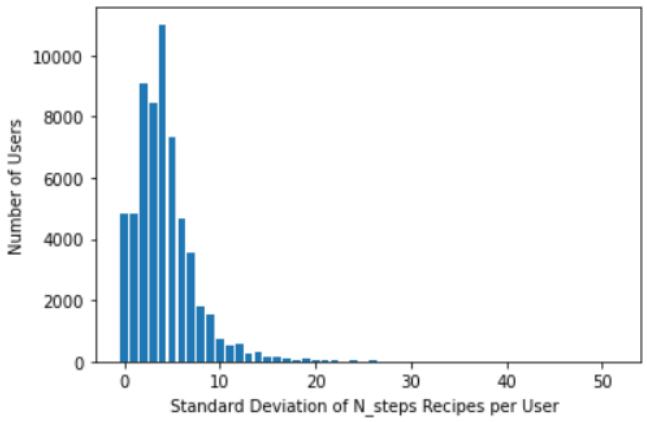

Figure 7. Number of Users vs Standard deviation of N_steps in Recipes

- **Calorie similarity measure**: For each user, we calculate the standard deviation of the calories for all the recipes the user has interacted with. If the standard deviation is high then one could *possibly* reason that the user doesn't interact with recipes with similar calorie values. On the other hand if the standard deviation is low then one could reason that user interacts with recipes with similar calorie values. In the latter case, calorie number could be a good similarity measure between recipes to recommend to that user. A plot of calories' standard deviation can be found in Figure 8. As observed, the standard deviation in calories is high and hence it might not be entirely justified to take calorie as a similarity feature.

Please note that the EDA examples explained above are just a part of all the analyses we performed. We performed similar analyses on other features, and are not representing them here for lack of space.

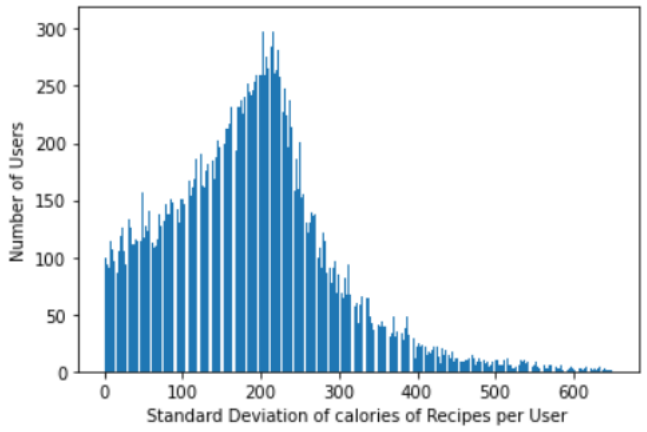

Figure 8. Number of Users vs Standard deviation of Calories in Recipes

2.2.2 Sentiment Analysis Preprocessing

The raw dataset is extremely large, consisting a total of 1132367 records, due to its large size, we sampled 50k records randomly to perform our data explorations and further analysis. This dataset does not have sentiment labels so we have created our own categories, where we categorize ratings of 4 & 5 as positive (1), ratings of 1 & 2 as negative (-1) and rating of 3 as neutral (0). We removed ratings of 0 as it was corresponding to the situation where the user decided not to rate but just leave a review. These reviews were of varying sentiments, so labelling them as positive or negative was not possible. We checked the distribution of sentiments in the dataset. This helps to check if the resulting dataset is balanced or not, as classifiers like logistic regression, decision trees etc would expect the data to be balanced. The distribution is present in Figure 1. We also explored the top words present most frequently in positive vs negative reviews Figure 10 and Figure 11 respectively. As the review data is raw text, preprocessing was done to make it robust when feature representations are generated from it. The steps undertaken were

- Removing punctuation - This involves removing all punctuations from the reviews
- Removing stopwords - Words like I, am, the etc which while present frequently, do not add any information.
- Tokenizing - Breaking up the sentences into unigram, or n-gram tokens.
- Lemmatizing - Reducing the word to its root form
- Embedding using different representations - To make text data compatible with ML models, we need to embed them as numeric vectors

2.2.3 Recipe Recommendation Preprocessing

As the raw dataset of recipes is very large with 231,637 records, 50k records were sampled randomly. The preprocessing steps were done to remove punctuation, stopwords and lemmatize the corpus of text. 2 kinds of recommendations were measured

1. Given a name, recommend similar recipes by name.
2. Given a combination of tags, ingredients and description, recommended similar recipes.

The feature for task 2 was derived by concatenating the rows of ingredients, description and tags. This was done to create a robust text document of the recipe. The final preprocessing step was done to create TaggedDocuments tagging which creates documents with document tags, which can then be vectorized using Doc2Vec.

2.2.4 Rating Prediction and Recommendation Preprocessing

For this task, we used the interaction dataset given in separate files for training, validation, and testing processed and stored separately. Each of these data consists of 6 entries: *{user_id, recipe_id, date, rating, u, i}*, where *u* and *i* are user and recipe ids mapped to a continuous integers starting from 0. We only use the entries from the columns *user_id, recipe_id* and *rating* to train our model, and compute validation accuracy and prediction on the test dataset.

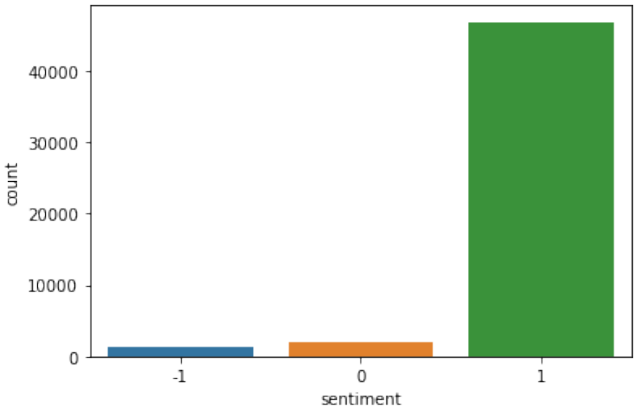

Figure 9. Distribution of sentiment labels - 1 for positive, -1 for negative and 0 for neutral

Figure 10. Positive sentiments have review words like perfect, sharing,love and wonderful

3. PREDICTIVE TASKS

3.1 Sentiment Analysis

The first predictive task that we will tackle is sentiment analysis from the reviews present in *raw_interactions*. In this task we aim to understand the socio-linguistic dimensions of sentiment in reviews, rather than the immediate task of predicting the rating. We chose this task because even though ratings give a good idea of user satisfaction, it does not account for user tendencies. A user might love the recipe, but be prone to give ratings of 3/4. This might be solved by analyzing their sentiment from reviews.

3.2 Rating Prediction using Interactions

As we saw earlier in the EDA, the data trends show that most users give ratings that are positive, i.e, a rating of 3 or higher. For this task, we want to predict, given the recipes the user has tried and the rating they have given to each recipe, if we were to recommend them a new recipe, what rating would they give to this new recipe. We will use the data from the following files *interaction_train, interaction_validation, and interaction_test*.

3.3 Recommending similar recipes

In this predictive task, instead of predicting rating or sentiment, we will try getting similar recipes to a given recipe using the

Figure 11. Negative sentiments have review words tough, awful, and disappointing

raw_recipes.csv dataset. This can be used in combination with the user interaction data. If a user has interacted with a recipe recently, they can be shown similar recipes based on the name or ingredients.

4. MODELS

4.1 Sentiment Analysis

We will evaluate 4 feature representations for sentiment analysis

1. Bag of words using TextBlob
2. TF-IDF
3. VADER (Valence Aware Dictionary for sEntiment Reasoning)
4. Word2Vec

4.1.1 TextBlob

TextBlob is a NLP library which is built upon NLTK. It is used for various preprocessing tasks like Part-of-Speech Tagging, stemming etc. It also has advanced capabilities like sentiment analysis. Textblob is powerful because it can give polarity of the sentiment (-1 to 0) in a text as well as the subjectivity (0 to 1). It also has a feature *show_informative_features* which can be used to indentify what words gave that sentiment. We used the PatternAnalyzer (based on pattern library) to do sentiment analysis.

Why this model - Textblob allows sentiment analysis using a Bag of words approach. If number of positive words are greater than negative words, then the sentiment is deemed positive. It is simple to understand, but fails to capture sentence meaning. It is not scalable if the text corpus is very large.

4.1.2 VADER

VADER (Valence Aware Dictionary and sEntiment Reasoner) [3] is a lexicon and rule-based sentiment analysis tool that is specifically attuned to sentiments expressed in social media. A sentiment lexicon is a list of lexical features (e.g., words) which are generally labeled according to their semantic orientation as either positive or negative. VADER not only tells about the Positivity and Negativity score but also tells us about how positive or negative a sentiment is. VADER produces four sentiment measurements from these word gradings. The initial

three, +ve, neutral, and -ve, address the % extent of the content that falls into those classifications. The last measurement, the compound score, is the total amount of the lexicon grades, which have been normalized to run between – 1 and 1. This matches with our labels of 1,-1 and 0.
Why this model - VADER allows sentiment analysis using lexicon and rule based approaches. The model has been carefully designed using social media texts, and can perform very well if the test data is similar. It is not generalizable if the text corpus is unique or different from what the model is trained on.

4.1.3 TF-IDF

TF-IDF is a popular way to embed words such that words which are infrequent across documents are given higher weights as they convey more information. Frequently occurring words have little impact on the similarity The similarity is now determined by the words that are most "characteristic" of the document. The formula for calculating TF-IDF embeddings for a word in a document in (1) . Due to computational costs we performed this embedding for the 5000 most frequent words in the corpus. (All stopwords have been excluded and words have been lemmatized) This embedding was fed into a decision tree classifier which is trained on 66% of the review data. 33% of the review data is used for testing. The data is stratified while splitting so the label distribution is maintained.
Why this model - TF-IDF is one of the most popular word embeddings which help analyse words based on their importance in the document. It can however not capture position in text, and semantic meaning of words. It also creates a sparse matrix which can be computationally inefficient. A decision tree is used for multi-class classification, as it is easy to interpret and can scale logarithmic with data. As with all decision trees, it is essential to keep the depths low or else the tree can overfit.

$$tfidf(t,d,D) = tf(t,d) * idf(t,d,D) \quad (1)$$

where tf(t,d): = number of times the term t appears in the document d

$$idf(t,D) = log(N/(d \in D | t \in d)) \quad (2)$$

4.1.4 Word2Vec

Word2Vec [5] is a very popular word embedding framework which tries to maximise the probability of co-occuring words and minimize probability of words which don't occur together. It uses neural networks to find a latent space of words. We used a Word2Vec vector size of 1000 with a window of size 5. This was used with a Skip-Gram model which predicts the context of a word given a target word. E.g predict context [quick, fox] given target word 'brown' . This was used as we were trying to group reviews when a target (positive or negative) word is given. We got a single 1000 dim vector for a review by considering the mean of word2vec tokens in a review containing different words. The train and test splits of preprocessed data were again similar to the TF-IDF models. Decision Trees were used to classify the sentiment of the model.
Why this model - Word2Vec retains the semantic meaning of different words in a document. The context information is not lost. Another great advantage of Word2Vec approach is that the size of the embedding vector is very small. Each dimension in the embedding vector contains information about one aspect of the word. We do not need huge sparse vectors, unlike the bag of words and TF-IDF approaches. Again, decision trees had the same pros and cons as listed in the TF-IDF section.

4.2 Rating Prediction using interactions

Knowing how would a given user rate a new recipe can help with recipe recommendations. This doesn't just help with the relevant recommendations to the user, but providing the recipes that users will like can increase their interaction numbers and bring in more revenue for the website. It is also important to know the current food or diet trend among the user so one can come up with more such recipes to increase audience interactions.

Since rating prediction is one of the simple tasks, fully based on past interactions and the ratings for those recipes, it is pretty simple to use a simple (bias only) latent factor-based model as our baseline model.

4.2.1 Baselines

To predict the rating for a given user and recipe pair, we used a simple bias-only latent factor-based recommender. Here we use gradient descent to implement a machine-learning-based recommender (a latent-factor model). First, we build some utility data structures to store the variables of our model (alpha, userBiases, and itemBiases). The actual prediction function of our model is simple, we just predict using a global offset (alpha), a user offset (beta_u), and an item offset (beta_i). The "cost" function is the function we are trying to optimize. Again this is a requirement of the gradient descent library we'll use. In this case, we're just computing the (regularized) MSE of a particular solution to our prediction parameters, and we iterate until we find the minimum MSE for the given parameters. We compute the MSE of a trivial baseline (predict average rating of the user) for comparison. Finally, we now run gradient descent. This particular gradient descent function takes arguments like our cost function (implemented above), initial parameter values, labels, and the regularization strength and computes the derivative function, returning the optimal bias terms. We late use these bias terms to predict the rating for the test dataset.

We also use the complete latent factor model as one of our baselines. Here we extend onto the bias-only baseline above to include a low-dimensional user and item terms. We initialize the parameters the same way as before, the cost and derivative computation also stay the same in terms of the role it plays in the computation, however, the computation becomes more complex due to the addition of more parameters. The actual prediction function of this model would now predict using a global offset (alpha), a user offset (beta_u), an item offset (beta_i), a user's preference (gamma_u), and item's properties (gamma_i).

4.2.2 Using Surprise (Latent Factor Model)

To improve our baseline model, we decided on using a fairly simple interface that implements the type of rating prediction model that we have described above. We are using Surprise Library which has methods that are easy to use and help with

training our model. So We first create a model instace (we use SVD() in this case), load our train and test dataset, and fit the model on the train set. We then make a prediction of the test set and compute MSE from the fitted model. Surprise gave us lot of trouble, due the format of our dataset and the compatibility issues with it's requirements of specific type of input set. We spend far too long on this to try something else that could have worked better.

4.3 Recommending similar recipes

In this section, we will explain the embeddings we use to calculate similarities between names, recipe tags, recipe ingredients and recipe descriptions. This is done so that given a target query, we can recommend recipes to the user which have the highest similarity.

4.3.1 TF-IDF embeddings

As explained during sentiment analysis, we use TF-IDF to get recipe embeddings. This is done for the 2 features (name, descriptions + ingredients + tags). A max_feature size of 1000 was used which would restrict the embeddings to the most frequent 1000 words. (excluding stop words)

4.3.2 Doc2Vec embeddings

We explore Doc2Vec embeddings which are a modification of Word2Vec. Doc2Vec creates a numeric representation of a document, regardless of its length. The method is similar to Word2Vec but learns another vector to learn the representation of the document. We use the skip-gram version of the model, Distributed Bag of Words Paragraph Vector (PV-DBOW). This model again using unsupervised learning to find latent spaces where similar documents (recipes) are together. The similarity metric we used was cosine similarity which can measure how similar two sequences of numbers are (in our case, the numbers are word embeddings)

5. LITERATURE SURVEY

The main paper we studied for this assignment is [4] by Majumder et. al. This paper describes a recipe "generation" algorithm using attention mechanism on encoded inputs (ingredients). They also use prior recipes a user has interacted with. Although we do not delve into recipe generation, this paper forms the basis of our assignment because the dataset we use was generated as a part of it. Some state-of-the-art techniques in this field of data to language generation use large scale transformer models.

A dataset similar to the one we used is *RecipeNLG* [1] by Bien et.al. It comprises of cooking recipes and is intended to be used while generating recipes in a semi-structured text format. Since recipe names don't follow particular naming or structure, it becomes difficult estimate generated recipe quality. The authors of this dataset explore various metrics to evaluate generated recipes.

An interesting take on recipe personalization and recommendation can be seen in [6], a work by Twomey et. al. Their main aim to develop an initial understanding and exploration of recipe recommendation in multiple problems. Their task becomes a good mix of interpreting ingredients in multiple languages, finding similarities based on interactions and recommending appropriate recipes. They use SOTA Python NLP techniques to interpret recipe ingredients.

Finally, we should mention [2] by Freyne, et. al. This is a formative work in recipe recommendation. It builds ingredient preferences for a user based on historic recipe interaction. It represents recipes by its ingredients. Our method in this assignment somewhat follows this approach too. They however use N nearest neighbour approach based on Pearson correlation, unlike us.

6. EVALUATION AND RESULTS

6.1 Sentiment Analysis

The evaluation of different models is present in Table 1. The performance of the models is similar in the dataset, however it is important to note that sentiment analysis was done for 50k reviews using TextBlob and VADER, but only 33% of the reviews for TF-IDF and Word2Vec. This was because the former were pretrained, and the latter were trained on a set of reviews and evaluated on an unseen test set. As we can see from the table, TF-IDF had the highest F1 score (90.9%) which the highest recall (along with VADER) Word2Vec did not perform well on recall as it was affected by the imbalance in the classes, and because we restricted the vector size to 1000. We can see from the confusion matrices that while most models can classify the positive sentiment (which is prevalent in the data) it struggles to capture negative and neutral sentiments. This is particularly evident for VADER which has no predictions as negative. (This is because the semantics of social media vs review text might be different). Even Bag of Words did not fare well in detecting negative sentiments, but it was decent in detecting neutral sentiments.

Possible Improvements Some of the improvements we can use in our model would be to use tools like SMOTE (Synthetic Minority Over-Sampling Technique) or ADASYN (Adaptive synthetic sampling) to upsample our dataset. Another would be to use longer feature vectors and tune the models more (and experiment with other classification models like random forest, XGBoost etc.)

6.2 Rating Prediction using Interactions

The evaluation of the different models we tried is given in Table 2. The performance of the models is similar to each other. However, it is important to note that surprise improves the performance a little more than the two baselines, and complete Latent Factor Model does a lot better for the test dataset compared to the bias-only model. This is expected, due to it's nature of treating the user and items independently from each other. The complete Latent Factor Model where we also consider the user's preferences and the item properties.

Possible Improvements: Given more time and computation power, we could find better regularization and learning parameters that give better results for the above model. Another model that we think would give us better results is a similarity-based rating prediction model, where we do not have to use feature vectors. Also, Latent-factor models are not as useful when it comes to looking at a new user who has not rated any recipes before or if the new recipe has never been rated before.

Sentiment Analysis Model	*Precision*	*Recall*	*F1 score*
BoW with TextBlob (rule based)	0.914	0.90	0.908
VADER (rule based)	0.897	0.913	0.905
Tf-idf with Decision Tree	0.905	0.913	0.909
Word2Vec with Decision Tree	0.936	0.876	0.905

Table 1. Evaluation of Sentiment Analysis Models

Rating Prediction Model	*Validation MSE*	*Test MSE*
Simple (Bias-only) LFM	1.6400	1.7961
Complete Latent Factor Model	1.6457	1.7548
Surprise Latent Factor Model	1.6233	1.7332

Table 2. Evaluation of Rating Prediction Models

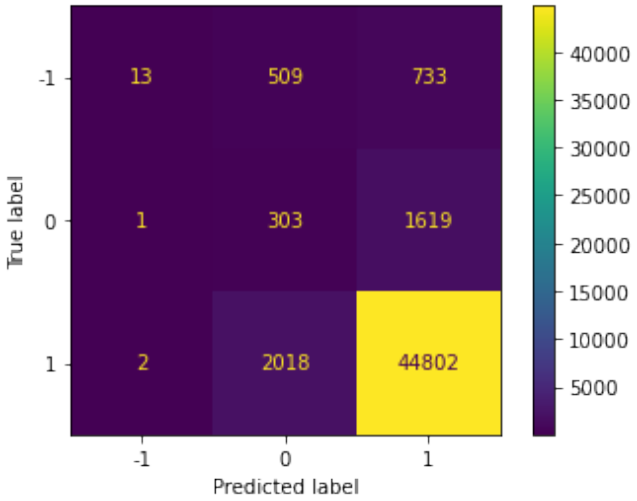

Figure 12. Confusion Matrix using TextBlob rule based analysis

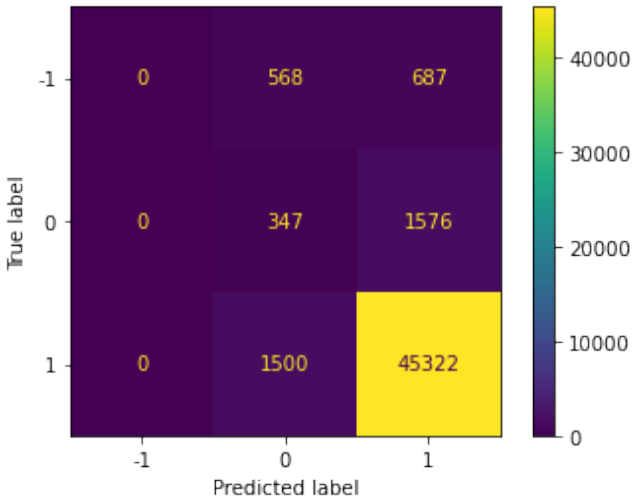

Figure 13. Confusion Matrix using VADER rule based analysis

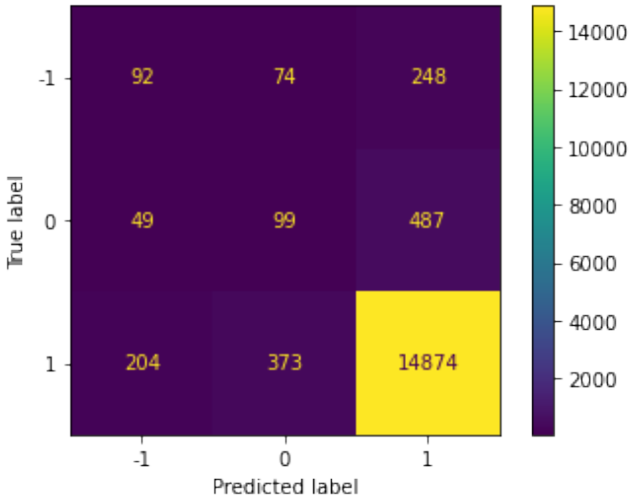

Figure 14. Confusion Matrix using TF-IDF embeddings with Decision Trees

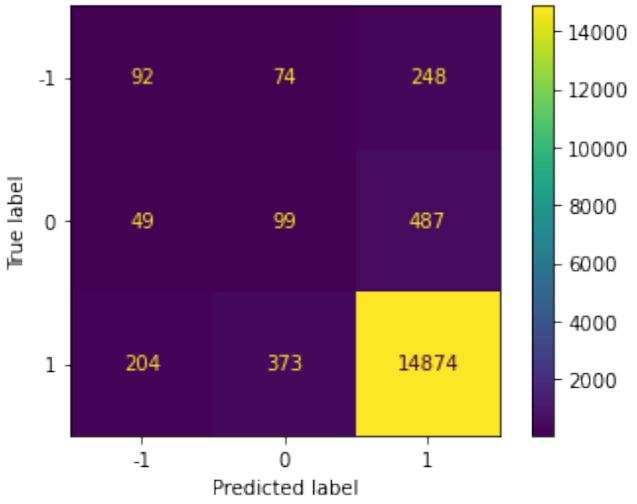

Figure 15. Confusion Matrix using Word2Vec embeddings with Decision Trees

```
black coffee barbecue sauce

['grilled chicken with coffee barbecue sauce',
 'black forest coffee',
 'coffee barbecue sauce from texas highways',
 'barbecue sauce',
 'barbecue coca cola sauce',
 'drake bulldogs superb barbecue sauce',
 'running creek barbecue sauce',
 'sexy barbecue sauce',
 'master class barbecue sauce',
 'the shack barbecue sauce']
```

Figure 16. Using TF-IDF Given a recipe name, recommendations for similar recipes

```
Similar to --- ['dry yeast', 'water', 'honey', 'wheat', 'oil', 'garlic clove', 'tomatoes', 'tomato sauce',
'mozzarella cheese', 'parmesan cheese', 'olive oil', 'salt', 'pepper', 'oregano', 'basil']

["['flour', 'active dry yeast', 'salt', 'warm water', 'vegetable oil', 'diced tomatoes', 'tomato paste',
'dried basil', 'oregano', 'marjoram', 'thyme', 'garlic powder', 'pepper', 'pepperoni', 'mozzarella cheese',
'parmesan cheese', 'romano cheese']",
 "['pizza dough', 'dried tomatoe', 'olive oil', 'sweet onions', 'salt & pepper', 'garlic cloves', 'anchovy
paste', 'dried oregano', 'mozzarella cheese', 'parmesan cheese']",
 "['lemon juice', 'olive oil', 'eggplants', 'tomato puree', 'tomatoes', 'oregano', 'basil', 'cayenne pepper',
'garlic cloves', 'dry breadcrumbs', 'parmesan cheese', 'ricotta cheese', 'mozzarella cheese']",
 "['olive oil', 'garlic cloves', 'san marzano tomatoes', 'salt', 'fresh basil']",
 "['egg', 'milk', 'vegetable oil', 'all-purpose flour', 'eggplants', 'tomato sauce', 'tomato paste',
'tomatoes', 'burgundy wine', 'dried oregano', 'dried basil', 'dried thyme', 'garlic salt', 'salt', 'mozzarella
cheese', 'parmesan cheese']",
 "['penne pasta', 'mozzarella cheese', 'basil leaves', 'parmesan cheese', 'tomatoes', 'olive oil', 'onion',
'garlic clove', 'fresh basil leaves', 'salt & freshly ground black pepper']",
 "['olive oil', 'semolina', 'warm water', 'instant yeast', 'salt', 'sugar', 'all-purpose flour', 'bread
flour', 'plum tomatoes', 'pizza sauce', 'fresh ground black pepper', 'garlic clove', 'fresh basil',
'mozzarella cheese', 'pepperoni']",
```

Figure 17. Using TF-IDF Given a recipe ingredients, recommendations for similar ingredients

```
Similar to -- ['dry yeast', 'water', 'honey', 'wheat', 'oil', 'garlic clove', 'tomatoes', 'tomato sauce',
'mozzarella cheese', 'parmesan cheese', 'olive oil', 'salt', 'pepper', 'oregano', 'basil']

["['farfalle pasta', 'onion', 'garlic cloves', 'olive oil', 'tomato sauce', 'diced tomatoes', 'salt & pepper',
'dried basil', 'dried oregano', 'parmesan cheese', 'romano cheese']",
 "['stewed tomatoes', 'water', 'ground beef', 'onion', 'minced garlic cloves', 'elbow macaroni', 'salt',
'pepper', 'olive oil', 'parmesan cheese']",
 "['spaghetti', 'diced tomatoes', 'mozzarella cheese', 'basil', 'parsley', 'salt', 'pepper', 'olive oil']",
 "['olive oil', 'garlic cloves', 'tomatoes', 'parsley', 'dried basil', 'dried oregano', 'cannellini beans',
'salt', 'pepper', 'pasta']",
 "['penne pasta', 'broccoli', 'olive oil', 'onion', 'canadian bacon', 'vodka', 'whole tomatoes', 'tomato
paste', 'salt', 'red pepper flakes', 'fat-free half-and-half', 'parmesan cheese', 'basil leaves']",
 "['pasta', 'olive oil', 'garlic clove', 'tomato sauce', 'fresh parsley', 'mozzarella cheese', 'salt',
'pepper', 'oregano']",
 "['zucchini', 'onion', 'green pepper', 'garlic', 'margarine', 'frozen spinach', 'mushroom stems and pieces',
'eggs', 'italian seasoned breadcrumbs', 'pasta sauce', 'salt', 'garlic powder', 'oregano', 'burgundy wine',
'swiss cheese']",
```

Figure 18. Using Doc2Vec - Given a recipe ingredients, recommendations for similar ingredients

6.3 Recipe Recommendation

The challenge in evaluating recipe recommendations was that there were no ground truths available. The only way to check if the recommended recipe was relevant to the user would be by integrating interaction data with it. While we tried that, it did not have robust enough test data as the user might not have interacted with the recipe, but it was very similar to the query recipe. The evaluation was done by qualitative evaluations of the recommendations. While the recipe recommendations were 95% accurate for names and ingredients, they were 85% accurate for tags as there is a wide variability in tags. Some qualitative results are present in Figure16 , Figure 17 and Figure 18.

Possible Improvements Possible improvements to these embeddings would be to consider n-grams of the ingredients and tags. Because certain ingredients need to be used in conjunction with each other.

CONCLUSION

For rating prediction, we used 3 different models and compared their performances based on the MSE of the prediction on the test dataset. We learned that the surprise latent factor model works better than the baseline, however, it gave us a lot of issues that could be resolved by using different similarity-based models. For sentiment analysis, we presented 4 approaches, which worked with varying accuracies. The TF-IDF method performed the best, and upsampling the data to make it more balanced would increase those metrics.

The rating prediction could be used as part of our overall recipe recommendation task, where we only recommend the recipes that the user will give a positive rating for. For example, we can make sure as one of our filtering mechanisms is that only use the recipe that the user is likely to give more than a rating of 3 stars.

Lastly, we formulated 2 embeddings to find similar recipes based on their names and the ingredients and tags. We presented qualitative results for them, and human evaluation gave it a 90% accuracy,

REFERENCES

[1] Michał Bień, Michał Gilski, Martyna Maciejewska, Wojciech Taisner, Dawid Wisniewski, and Agnieszka Lawrynowicz. RecipeNLG: A cooking recipes dataset for semi-structured text generation. In *Proceedings of the 13th International Conference on Natural Language Generation*, pages 22–28, Dublin, Ireland, December 2020. Association for Computational Linguistics.

[2] Jill Freyne and Shlomo Berkovsky. Intelligent food planning: personalized recipe recommendation. In *International Conference on Intelligent User Interfaces*, 2010.

[3] Clayton Hutto and Eric Gilbert. Vader: A parsimonious rule-based model for sentiment analysis of social media text. In *Proceedings of the international AAAI conference on web and social media*, volume 8, pages 216–225, 2014.

[4] Bodhisattwa Prasad Majumder, Shuyang Li, Jianmo Ni, and Julian McAuley. Generating personalized recipes from historical user preferences, 2019.

[5] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.

[6] Niall Twomey, Mikhail Fain, Andrey Ponikar, and Nadine Sarraf. Towards multi-language recipe personalisation and recommendation. In *Proceedings of the 14th ACM Conference on Recommender Systems*, RecSys '20, page 708–713, New York, NY, USA, 2020. Association for Computing Machinery.