Food.com Recipe Rating Prediction

Github

ABSTRACT

In this paper, we describe three different approaches for predicting user ratings on cooking recipes. We utilized Singular Value Decomposition (SVD), Support Vector Regression (SVR), a Neural Network, and a hybrid Light Gradient Boost Model (LGBM). We found the most successful strategy was using the Neural Network, achieving our lowest root mean squared error (RMSE) of 1.115. This means the average difference in rating was 1.115 stars off. This paper will discuss the background of each model as well as the advantages and disadvantages each model brings.

Keywords

Singular Value Decomposition (SVD);

Support Vector Regression (SVR);

Light Gradient Boost Model (LGBM);

Root Mean Squared Error (RMSE);

Recommender Systems;

1. INTRODUCTION

The goal of this research project was to develop a model that can effectively predict a rating for a recipe given certain information (user, recipe, ingredients, etc.). To that end, we focused on a few different attributes: input features, model type, and hyperparameters.

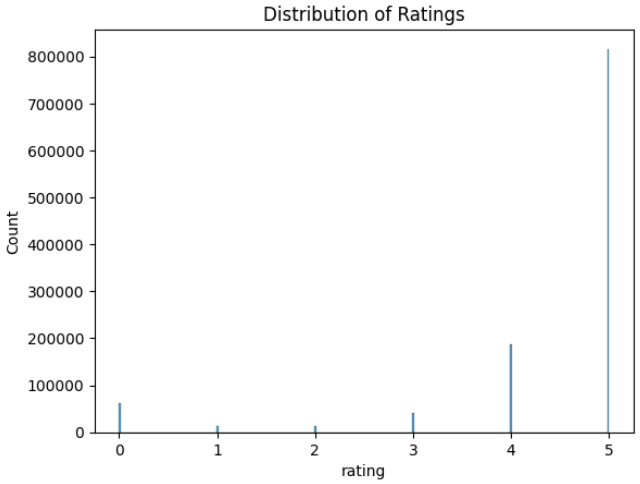
As an example, one model that was utilized was the Support Vector Regression model. It utilizes sentiment analysis of each user's review as its input feature to predict how the user would rate a recipe. If a user has a positive sounding review for a recipe, this can impact how the model may predict their rating.

Each of these attributes altered the root mean squared error. The RMSE explains how the average difference in rating the model predicted compares with the true rating value. To assess each model, the RMSE was minimized.

The models in this paper can be used in recommender systems. Recommender systems are used in a wide variety of applications. Their basic function is to recommend items to users. In this case, one application of the models found here would be to recommend recipes to users based on their previous reviews of other recipes. Using the predicted ratings that these models generate, recipe websites, like Food.com, could only recommend recipes that the model believes would rate highly.

2. DATASET

We chose to work on the Food.com Recipes and Interactions dataset, which contains over 180k recipes and over 700k recipe reviews over the years 2000-2018[15]. This is a comprehensive database that anonymizes users and recipes while also providing extensive information related to how the user interacts with the site. Information exists pertaining to which recipes the user reviewed along with the rating they provided to any given recipe. This serves to be extremely useful for our task, since we can use these features to potentially glean insight into the rating a user would give to a recipe given knowledge of their past rating history.



The dataset itself is clean, so minimal preprocessing was necessary for our task. It also offers a tokenized version of review and ingredient text that can be utilized directly with a model. We were most interested in user rating prediction on recipes, so default organization of the data was most effective for our task in most cases. However, for our SVR model, we tokenize the revie text of a user with a TfidfVectorizer from the scikit-learn library before passing it our model. This is due to the model's input being only the user's review. We also ensure that there are no null entries in our dataset before utilizing it.

3. RELATED WORKS

The Kaggle dataset we used was based on a paper that proposes an natural language processing (NLP) task of personalizing recipe generation based on listed ingredients[12]. The model expands incomplete ingredient details into complete generated instructions that follow the user's recommended preferences. Features they used that are included in the dataset include techniques in a BPE tokenized format and recipes representations of a user's previously consumed recipes in an attention fusion layer to control recipe text generation.

Similar papers that utilize the same dataset specifically for food review prediction include one paper comparing pure vector embedding models, similarity-based models, latent factor models, SVD++, and bias-only models, with a bag-of-words vectorizer being the best at finding user preferences[10]. A similar paper also utilizes a linear regression model with a bag-of-words approach for the ingredients list to achieve the best results. Our work delves deeper into text-based NLP techniques that can outperform pure SVD models and highlights the importance of features engineering in recommender systems, which is similarly portrayed on paper on a hybrid SVD model that includes more external factors when collaborative information is too sparse and inaccurate[7].

4. MODELS

4.1 SVR[1]

Support Vector Regressions (SVR) attempt to fit a line (or curve) that best predicts the rating from the given data. It is different from regular regressions in the sense that it is focused on generating a line that is good enough, not one that completely minimizes every error.

To achieve this, SVR models do not use just one line, but instead a sort of range around this line. If a data point falls within this range, it is not seen as an error. This range is the margin of error that is allowed. The model thus attempts to minimize all errors that occur outside of this margin (these points are called Support Vectors).

This allows SVR models to be robust and tolerant of variations in the data since it only focuses on outliers to affect the line.

4.1.1 TF-IDF Vectorization

To preprocess our input for the SVR model, as mentioned earlier, we use a TfidfVectorizer. This tool converts our input words into a matrix of TF-IDF features. TF means term frequency, which is how often a word appears in the text. IDF means inverse document frequency, generally referring to how often a word appears across an entire set of documents. These values are extremely useful for determining how the words relate to one another in a text as well as determine how important they are. We also ensure to ignore any stop words (a, the, etc.) which have little influence on the meaning of a sentence.

$$TF(word, doc) = \frac{Frequency of word \in the doc}{No. of words \in the doc}$$

$$IDF(word) = log_e\left(1 + \frac{No. of docs}{No. of docs with word}\right) \quad [2]$$

This vectorizer computes this for all words in a review for each training sample. These vectors can be converted into a list for sentiment analysis through SVR now that the terms are in a more processible numerical format.

4.2 LGBM

We developed a hybrid recommendation system combining LightGBM and SVD++ to predict user ratings for recipes. LightGBM, a gradient boosting model, was used to leverage structured features, while SVD++ provided latent factors capturing implicit user-item interactions. The model incorporated the following features: user ID, recipe ID, number of recipes reviewed by the user, average user rating, SVD++ predicted ratings, latent SVD features and biases (e.g., p_u, q_i, b_u, b_i), global rating mean, and recipe steps length.

The SVD++ predictions and latent factors were used as inputs to LightGBM to capture both explicit (e.g., user behavior and recipe metadata) and implicit (e.g., user preferences and item characteristics) patterns. The RMSE for the hybrid model was 1.29, demonstrating that combining the collaborative filtering method of SVD++ with the flexibility of LightGBM's feature-based learning does not necessarily lead to improved accuracy in predicting user ratings. This hybrid approach utilizes both the latent preferences learned by SVD++ and the structured features that LightGBM can model. The personalized recipe recommendations is therefore highly dependent on the quality of input features in the LightGBM model and would benefit from vectorized embeddings for review text, recipe names, steps, or ingredients.

LightGBM is a gradient boosting framework designed for efficient handling of large datasets and complex features, particularly in supervised learning tasks. It is a tree-based model that leverages decision trees for both regression and classification, building an ensemble of trees to minimize a loss function iteratively. LightGBM is particularly well-suited for datasets with diverse feature types, including categorical and numerical data, and it handles sparse data effectively.

In contrast, SVD++ is a matrix factorization-based collaborative filtering method used primarily for recommendation systems. LightGBM excels in supervised learning tasks with complex features, while SVD++ is tailored for recommendation systems, focusing on learning from user-item interactions. LightGBM is more general-purpose and can handle a broad range of structured data types, while SVD++ is specialized for collaborative filtering and works best when the goal is to predict user-item interactions based on historical data. LightGBM's ability to model non-linear relationships and handle large-scale datasets makes it highly flexible and efficient, whereas SVD++ is more suited for tasks where explicit and implicit user preferences need to be modeled, typically in a sparse matrix setting. This is why we decided to combine both models in a hybrid approach to leverage their complementary strengths, with SVD++ capturing deeper latent patterns in user-item interactions that can be inputted into the more generalized LightGBM predictive framework. Better feature engineering and hyperparameter tuning may allow this hybrid model pipeline to better outperform pure SVD++.

4.3 Latent Factor Models

Latent factor models are widely used in recommender systems for their simplicity and ability to handle unseen data points. In simple terms, latent factor models extract the hidden (latent) features or characteristics of the users and items. For example, a user may really like spicy food, but it is not explicitly shown in the data. This is a latent characteristic that latent factor models attempt to discern.

Then given a big matrix of users as the rows and items as the columns, where each entry is the rating, the latent factor model breaks this big matrix into two smaller ones. One which contains the user's preferences stored as a vector and one that contains the item features stored as a vector. These two matrices are then multiplied together in an attempt to predict the rating (or whatever is specified), including any unknown values.

This model is simple, easy to use, and allows for predictions on items that are not explicitly known.

4.3.1 Surprise SVD[9]

$$\hat{r}_{ui} = \mu + b_u + b_i + q_i^T p_u$$

In the SVD (Singular Value Decomposition) model, we predict the rating given a user-item pair, where the item is the recipe in our case, based on parameters found through minimizing regularized square error.

$$\sum_{r_{ui} \in R_{train}} \left(r_{ui} - \hat{r}_{ui}\right)^2 + \lambda \left(b_i^2 + b_u^2 + ||q_i||^2 + ||p_u||^2\right)$$

The decomposition allows us to uncover latent factors in the data, enabling more efficient representation and improved model performance as it's able to discover underlying features that capture hidden patterns, correlations, and relationships within the data, which may not be immediately obvious. This makes SVD particularly well-suited for recommendation tasks. In our case with our dataset being perfectly tailored for user/recipe focused predictions, we can directly train our model with inputs being a user and recipe pair and our output as the rating.

We use the SVD model part of the Surprise library, with the following parameters: n_factors = 100, n_epochs = 200, biased = True, lr_all = 0.005, reg_all = 0.1. As mentioned above, we trained the model using user ID, recipe ID, and rating with data from the training set provided as part of the Food.com dataset. Our best model had an RMSE of 1.27 on the test set, meaning that generally our model will predict a rating within 1.27 stars from the actual rating.

4.3.2 Surprise SVD++[9]

SVD++ is similar in structure to SVD, however it also takes advantage of what it calls "implicit" ratings, unlike standard SVD which only uses "explicit" ratings. Explicit ratings are what the user directly provides as a rating for a given recipe in this case. In comparison, implicit ratings are more subtle actions or observations that can help determine how a user would interpret a recipe. SVD++ treats the idea that a user rated an item as an implicit rating, without necessarily caring about the actual rating itself. This helps improve its performance over standard SVD at the cost of increased training time due to having to fit a greater number of parameters to the dataset. Similar to SVD, this model will take a user and a recipe as input and provides a rating as output.

We use the SVD++ model part of the Surprise library, with the same parameters as our SVD model. The training process was effectively the same as well. Our best SVD++ model had an RMSE of around 1.27.

4.4 Neural Networks

4.4.1. Neural Network

We also explored models beyond traditional recommender systems. Neural Network can be helpful with modeling non-linear, complex relationships on larger datasets, which is suitable for our dataset and task.

We developed an architecture that predicts user rating on a recipe based on user reviews. We first transform the raw review text into sentence embeddings using Word2Vec from the Gensim library. The user ID and recipe ID are also mapped to continuous indices. The model, implemented with the PyTorch library, is a fully connected neural network that concatenates user embeddings, recipe embeddings, and review embeddings and passes it to subsequent layers. The dimension is set to 50 for user and recipe embeddings and 100 for pre-computed review embeddings. There are two hidden layers, each with ReLU activation, followed by an output layer for regression. The dimension of the first hidden layer is (embedding_dim * 2 + review_embedding_dim, 128), and for the second hidden layer is (128, 64). We also add weight decay to mitigate overfitting, with experiments performed on varying decay values. The model is trained using the Mean Squared Error (MSE) loss function. The training procedure also uses the Adam optimizer with a learning rate of 0.001.

5. RESULTS

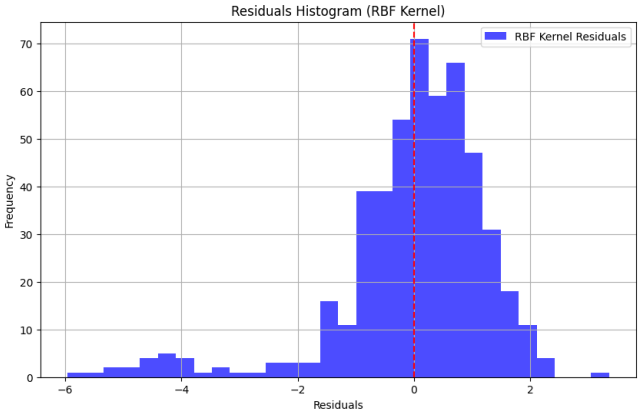
5.1 Sentiment Analysis and SVR[4]

As the data was so large, in order to run and analyze this model in a reasonable time, we randomly split the data into a training and test set using a 80:20 split. We then took the first 10,000 entries from the training set and ran it on the full test set that had been created.

Our Sentiment Analysis and SVR models performed admirably. We tested four different kernels, linear, polynomial, radial basis function (RBF), and sigmoid. Of the four, RBF performed the best.

Table 1. (MSE, RMSE) of Different Kernels

	MSE	RMSE
Linear	1.486	1.219
Polynomial	1.442	1.201
Radial Basis Function	1.390	1.179
Sigmoid	1.584	1.259



5.2 LGBM

Table 2. (MSE, RMSE) of Different Features for LGBM

Features	MSE	RMSE
users, recipes, steps length, average rating per user	1.748	1.322
users, recipes, steps length, average rating per user, SVD++ predictions	1.695	1.302
users, recipes	1.731	1.316

The LGBM model that utilizes our SVD++ predictions performs best. The model uses hyperparameters with learning rate 0.01, number of leaves is 63, feature fraction is 0.9, bagging fraction is 0.8, and bagging frequency is 5.

5.3 Latent Factor Models

5.3.1 Surprise SVD

The Surprise SVD model only utilized two inputs, user ID and item ID. No preprocessing was needed. Thus, we used the base Surprise SVD model to predict rating.

We were able to utilize the preprocessed training, validation, and test sets that were given in the dataset.

Table 3. (MSE, RMSE) of Different Number of Factors

	MSE	RMSE
10	1.745	1.321
20	1.745	1.321
50	1.747	1.322
100	1.750	1.323

Table 4. (MSE, RMSE) of Different Number of Epochs

	MSE	RMSE
10	1.788	1.337
20	1.750	1.322
50	1.727	1.314
100	1.724	1.313

Table 5. (MSE, RMSE) of Different Regularization

	MSE	RMSE
0.001	1.750	1.323
0.01	1.751	1.323
0.1	1.752	1.324
0.2	1.755	1.325

5.3.2 Surprise SVD++

Table 6. (MSE, RMSE) of Different Epochs

	MSE	RMSE
10	1.693	1.301
50	1.695	1.302
100	1.667	1.291
150	1.638	1.280

The SVD++ model has hyperparameters of learning rate 0.005, regularization 0.1, and n factors 150.

5.4 Neural Network

Table 7. (MSE, RMSE) of Different Regularization

	MSE	RMSE
0.00	1.387	1.178
0.1	1.391	1.179
0.01	1.244	1.115
0.001	1.260	1.122
0.0001	1.310	1.145

5.5 Best Results

Finally, we compile the best results from each model, reported in RMSE. The best performing model in terms of RMSE is the Neural Network, with a RMSE of 1.115. The worst performing model is LGBM with an RMSE of 1.3. We also note that the results from SVD and SVD++ are very close, with only a 0.01 difference.

Table 8. Best RMSE Results from Each Model

SVD	SVD++	SVR	LGBM	Neural Network
1.313	1.280	1.179	1.302	1.115

6. CONCLUSION

As shown by the results, the best performing model was our Neural Network model. It had the lowest RMSE of all the models. A major disadvantage to using this model however is the training time. As mentioned, we were only able to train the model on a small part of the dataset because of time constraints. This similarly applies to our SVR model. It achieved similar results to the Neural Network model but also took a significant amount of time to train. Thus, while these models did perform the best, they may not be practical with large datasets like this one. Further research and testing should be done to train them on the entire dataset, as this may improve performance a lot. Ideally, using neural network tools like PyTorch and TensorFlow which offer parallelization through GPUs could be advantageous here to train more complex models more quickly. Even still, the performance it achieves with a smaller sample is impressive, considering how much of the dataset we did not use for this model.

The SVD model shows acceptable performance given how much faster it is at training and predicting compared to the other models. It is also significantly simpler and easier to implement as it only requires the user and item interactions. SVD++ offers similar performance and may be even better through further hyperparameter tuning, however it takes substantially longer to train. The main limitation with these models is with scikit-learn inherently, which does not offer GPU support[6]. This would affect the ability to scale these models up, but otherwise it is effective on this dataset.

The LGBM model shows worse performance than pure SVD++ when we include both the hybrid model and the pure LGBM model. However, the process is much quicker than SVD++ and more efficient.

7. ACKNOWLEDGMENTS

Our thanks to Julian McAuley, our CSE158 instructor, for all his guidance throughout the course. Additional thanks to all of our teaching assistants who also provided invaluable help when we were struggling. We would also like to thank the class for being such a welcoming environment for learning all about recommender systems.

8. REFERENCES

[1] Awad, M., Khanna, R. (2015). Support Vector Regression. In: Efficient Learning Machines. Apress, Berkeley, CA. https://doi.org/10.1007/978-1-4302-5990-9_4

[2] Das, B., & Chakraborty, S. (2018). An improved text sentiment classification model using TF-IDF and next word negation. *arXiv preprint arXiv:1806.06407*.

[3] Devlin, J., Chang, M., Lee, K., & Toutanova, K. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. ArXiv.https://arxiv.org/abs/1810.04805

[4] Chang, C.-C., & Lin, C.-J. (2001). LIBSVM: A Library for Support Vector Machines. https://www.csie.ntu.edu.tw/~cjlin/papers/libsvm.pdf

[5] Forman, G. 2003. An extensive empirical study of features selection metrics for text classification. J. Mach. Learn. Res.3 (Mar. 2003), 1289-1305.

[6] *Frequently Asked Questions*. Scikit-Learn. (n.d.). https://scikit-learn.org/stable/faq.html#will-you-add-gpu-support

[7] Frolov, E., & Oseledets, I. (2018). HybridSVD: WhenCollaborative Information is Not Enough. ArXiv.https://doi.org/10.1145/3298689.3347055

[8] Hensley, S. (2019). CS 229 Final Report: Recipe Rating Prediction (Natural Language). 33rd Conference on Neural Information Processing Systems (NeurIPS 2019),Vancouver, Canada.

[9] Hug, N., (2020). Surprise: A Python library for recommender systems. Journal of Open Source Software, 5(52), 2174, https://doi.org/10.21105/joss.02174

[10] Jain, M., Woenardi, J., & Makhija, J. Recipe for Success: Accurately Predicting Recipe Rating.

[11] Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., ... & Liu, T. Y. (2017). Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, *30*.

[12] Majumder, B. P., Li, S., Ni, J., & McAuley, J. (2019). Generating Personalized Recipes from Historical User Preferences. Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP). https://doi.org/10.18653/v1/d19-1613

[13] Platt, John (1999). "Probabilistic Outputs for Support Vector Machines and Comparisons to Regularized Likelihood Methods"

[14] Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.

[15] Shuyang Li. (2019). Food.com Recipes and Interactions [Data set]. Kaggle. https://doi.org/10.34740/KAGGLE/DSV/783630