CS 229 Final Report: Recipe Rating Prediction (Natural Language)

Sarah Hensley (hensleys)

Abstract

This work develops a model for predicting the star rating of an online recipe from the ingredients used in the recipe. We compare two text embedding strategies for encoding the ingredients, and compare linear regression and kernel methods, using a square kernel and a Gaussian kernel. The linear regression model with one-hot encoding achieves the best performance, with a mean squared error of 0.172, corresponding to the average prediction being within half of a star of the true recipe rating.

1 Introduction

This project aims to predict the rating of a recipe, given the ingredients used in the recipe. The motivation for this is straightforward; given the ingredients in my refrigerator, how good of a meal can I cook? More technically, this project is interested in whether the ingredients alone are an accurate predictor of the recipe rating.

To accomplish this, we develop a model that takes as input a list of ingredients and outputs a real value. The ingredient list is the ingredients necessary to cook an online recipe. The real valued output should be between one and five, representing the average star rating (between one and five) assigned to the recipe by recipe website users. As an example, the model should take in the ingredient list [“`canned chicken`”, “`green chilies`”, “`cream cheese`”, “`green onions`”, “`flour tortillas`”, “`cooking oil`”] and output 5.0. Throughout this paper, we use “ingredient list” to refer to the English word representation of the list of ingredients used to make a certain food dish, “rating” to refer to the average star rating assigned by online users, and “recipe” to refer to an (ingredient list, rating) pair.

To predict the rating of a recipe, we train several different models. To embed the ingredient list as a feature vector, we experiment with both a one-hot encoding strategy and with BERT, a pre-trained natural language processing encoder. With one of these embeddings as input, we examine linear regression and kernel methods, using both a square kernel and a Gaussian kernel.

As a solo project, all contributions are from the sole author. The code developed for this project can be found at https://github.com/j-s-hensley/CS-229.

2 Related Work

There is very little work predicting the online ratings of recipes; instead, research focuses on developing a recommendation system for recipes based on a particular user’s ratings of recipes. Majumder et al. propose a personalized recipe generator, which accepts a user’s ratings and reviews of recipes, key ingredients, a recipe title, and calorie level, and returns a set of instructions for a dish to create (1). The dataset used in this project was originally developed by Majumder et al. for this work. While this work has interesting results, it only implicitly predicts whether a user will like the result, instead of explicitly rating the recipe. The work done by Harvey et al. builds a recipe recommendation model tailored towards recommending healthy recipes based on users past preferences (2).

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

Similarly, Freyne and Berkovsky develop a healthy recipe recommendation system, with a focus on how to relate user preferences on food preparation to possible recipes (3). Teng et al. develop a recipe recommendation model that attempts to capture ingredient similarities and possible substitutions (4). Finally, work by Mao et al. attempts to predict whether a recipe will be popular by doing sentiment analysis of reviews (5).

3 Dataset and Features

We used recipe information from Food.com, as taken from the Kaggle dataset "Food.com Recipes and Interactions". Before using the dataset, we did preparatory work and pre-processing. The dataset was originally associated with (1), which instead focused on developing a personal recommendation system using the ratings that individual users assigned to recipes. As a result, ratings were organized by user, and not directly associated with the ingredients for the recipe. To prepare the data, we first removed any ingredients that appeared in fewer than five recipes, computed the average rating for each recipe, and made the pair-wise associations between an ingredient list and an average user rating. After this first set of pre-processing steps, we removed any recipes that did not have any remaining ingredients or did not have any user ratings. The resulting dataset contained $226,590$ recipes, which could not fit in memory, so we instead used $5,000$ recipes as a training set, $1,000$ as a validation set, and $1,000$ as a test set. With this preprocessing, we had $3,866$ unique ingredients. Each ingredient list was composed of between 1 and 44 ingredients.

We evaluated two methods for encoding the ingredient list for each recipe as a feature vector. Each of the embedding strategies satisfies different desiderata. Ideally, we would like the embedding to not depend on order of the ingredient list; shuffling the ordering of the list should not affect the output rating. In addition, an ideal embedding should capture information about related ingredients; for example, "granulated sugar" and "powdered sugar" should be more similar than "granulated sugar" and "chicken". To achieve this, we considered doing further pre-processing on the dataset to group ingredients. However, three issues emerged. First, $3,866$ ingredients is far too many for one person to hand-tune. Second, the correct level of granularity in grouping ingredients would be an additional qualitative hyperparameter, whcih would be difficult to keep consistent and tune. Third, automating to select for nouns loses essential ingredient information; "chocolate chips" and "potato chips" would both become "chips", despite being used in vastly different contexts. Thus, we must rely on our embedding strategy, instead of pre-processing, to capture relational information.

First, we considered a one-hot vector encoding. Each ingredient list was represented a vector of length $3,866$, where each index corresponded with one of the $3,866$ unique ingredients. The feature vector was then a sum of one-hot vectors for each ingredient present in the ingredient list. Alternatively, this could be viewed as a "bag of ingredients" approach, in which a feature vector index corresponds to the number of times an ingredient appears in an ingredient list. Using domain knowledge, we know that each ingredient appears at most once, and so these strategies result in identical feature vectors. Note that this is slightly different than a standard bag-of-words approach, which would exhibit the undesirable behavior of, for example, splitting "olive oil" into two separate features "olive" and "oil". This embedding does not capture relational information between ingredients; however, it does satisfy our criterion that it does not depend on the ingredient list ordering.

Second, we embedded ingredient lists using BERT, a pre-trained deep natural language processing model, as presented in (6). BERT accepts a sentence as input and returns a feature vector of length 768. To create the feature vector, we concatenated the full ingredient list as one string, with ingredients separated by semi-colons, and embedded this string using BERT. As a deep NLP model trained on a large corpus, BERT captures contextual meanings of words using other words in the sentence. Thus, BERT should satisfy that closely related ingredients correspond to similar feature vectors. However, BERT uses syntactical information when encoding a sentence; thus, the ingredient list ordering affects the output feature vector. Using individual ingredients as inputs to BERT would not solve this problem. The feature vector for an ingredient list could not be a concatenation of its ingredients, because the recipes are variable length; in addition, concatenation would introduce dependence on ingredient list ordering. Summing the feature vector for each individual ingredient would lose embedding information. Thus, this is an issue inherent in deep NLP embeddings.

4 Methods

Given an ingredient list embedding as our feature vector, we want to predict a real value in the range $[1, 5]$ as the rating of the associated recipe. We applied linear regression and kernel methods, as these are the two methods discussed in class for regression (as opposed to classification). Note that, while we could have strictly enforced the output range by, for example, applying a scaled sigmoid function to the output of the model and shifting it, we chose not to do this, as it would encourage extremely large parameter values to reach the extreme ends of the range. Instead, the model must learn that the acceptable output range is $[1, 5]$.

For both methods, we must define a loss function, so that we may update our parameters to minimize this loss. The goal is to predict a value close to the true rating, and so we want some distance metric. We choose the mean squared error loss, as this is convex and therefore guaranteed to find globally optimal parameters, unlike the ℓ_1 distance. Letting the feature vector be x, the true rating be y, the predicted rating be $h(x)$, and the number of examples be n, we defined our loss function as

$$J(\theta) = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{2} \big(h(x^{(i)}) - y^{(i)}\big)^2$$

For linear regression, we learn the vector θ to minimize the loss, where predictions are defined as

$$h_{\text{lin reg}}(x) = \theta^T x$$

With this prediction function and loss function, we update θ iteratively using full-batch gradient descent. Letting X be the design matrix (i.e. each row corresponds to one feature vector) and letting α our learning rate hyperparameter, we update as

$$\theta \leftarrow \theta + \alpha \big(Y - X\theta\big) X$$

To understand linear regression intuitively, consider the bag-of-ingredients embedding. Our prediction for a new embedding is the inner product with θ, and so each present ingredient contributes some weight (defined by θ) to the overall ingredient prediction. Thus, at a high level, we may understand of linear regression as thinking of each recipe to be the sum of its parts. With the BERT embedding, each rating is a linear combination of the feature vector elements, although there is no clear intuitive meaning of the feature vector elements.

Next, we applied kernel methods. With this, we aim to learn β to predict as

$$h(x) = \sum_{i=1}^{n} \beta_i K\big(x^{(i)}, x\big) \tag{1}$$

where K is a kernel function. A kernel function $K(x_1, x_2)$ acts as an inner product between virtual feature vectors derived from x_1 and x_2. We have previously shown that a linear kernel is equivalent to linear regression.[1] Given a kernel function, to learn the parameter β, we update as

$$\beta \leftarrow \beta + \alpha \big(Y - K\beta\big)$$

where we have defined K as a kernel matrix, such that each entry satisfies $K_{ij} = K\big(x^{(i)}, x^{(j)}\big)$ for every pair $(x^{(i)}, x^{(j)})$ in the training set. We considered two kernel functions.

We first considered a square kernel, as defined by

$$K_{\text{square}}(x, z) = (x^T z)^2$$

Again, consider the bag-of-ingredients embedding. With this kernel, the virtual feature vector is the product of every possible pairwise combination of ingredients. Thus, this kernel can capture information about nonlinear combinations of ingredients. This reflects our human understanding of recipes; seeing "dough" and "mozzarella" may indicate to us that the recipe is for pizza, and therefore should obtain a high rating. This kernel may now capture this pairwise information that the ingredients separately cannot convey.

[1] We implemented a linear kernel to check that our overall kernel implementation was correct, and recovered the exact same results as from linear regression, which is surprising given the space of hyperparameters.

Next, we consider a Gaussian kernel. Letting σ be a spread hyperparameter, we define

$$K_{\text{Gaussian}}(x, z) = \exp\left(-\frac{||x - z||_2^2}{2\sigma^2}\right)$$

Once more, consider the bag-of-ingredients embedding. We may expect that recipes with nearly identical sets of ingredients should obtain nearly identical ratings, and that recipes with entirely different ingredients should not influence the ratings of each other. The Gaussian kernel satisfies exactly that, with large values of $K_{\text{Gaussian}}(x, z)$ when the ℓ_2 distance between x and z is small, and small values of $K_{\text{Gaussian}}(x, z)$ otherwise. The value of σ controls the rate at which the weight contribution from increasingly different feature vectors falls off.

5 Results

As noted in the previous section, we split our dataset into $5,000$ training examples, $1,000$ validation examples, and $1,000$ test examples. We use the training data to learn the parameters for a given model. We use the validation dataset to do a hyperparameter sweep over the number of iterations and the learning rate. More specifically, we evaluate the loss of each model on the validation dataset with twenty-five possible learning rates, making plots such as Figure 1 for each learning rate to determine the optimal number of iterations to stop training. Using the number of iterations and learning rate that achieved the best performance over the validation dataset, we report the loss on the test dataset. Note that, as a regression problem, the loss is equivalent to the average error and summarizes the performance of our model; no confusion matrix, area-under-curve, or precision results were reported, as these are all only applicable to classification problems. [2] Note that this is not cross-validation, because the validation dataset is a holdout set, and so it does not overlap with the training dataset. We choose not to use cross-validation because we are in a data-rich environment, and can afford to keep a holdout set. The results for the different models are reported in Table 1.

model kernel embedding	linear reg. one-hot	kernel square one-hot	kernel Gaussian one-hot	linear reg. BERT	kernel square BERT	kernel Gaussian BERT
training loss	0.1958	0.1181	0.2660	0.2114	0.2338	0.0002
validation loss	0.1893	0.2028	5.843	0.1998	0.2262	0.3886
test loss	0.1719	0.1866	6.962	0.1866	0.2125	0.3989

Table 1: The mean squared error achieved by different models on each dataset.

Surprisingly, linear regression with a one-hot embedding of the ingredient list shows the best performance. This suggests that the BERT embedding is not successfully capturing more information about the ingredient list than the one-hot encoding captures. In addition, it may be indicative that the training dataset used was too small to capture nuance between ingredient interactions, thus making kernel methods underfit and underperform.

A particular interesting result is the behavior of the Gaussian kernel. With both embeddings, the training set achieves loss of at least an order of magnitude better than the validation or test datasets. While the initial suspicion may be that the model overfits to the training set, this is not the case. As can be seen in Figure 2, the validation loss does not increase as the training loss decreases, as would be expected in overfitting. Instead, the validation loss plateaus far earlier than the training loss, but is non-increasing.

We propose a possible reason for this behavior. Recall from Equation 1 that the Gaussian kernel predicts a rating from an ingredient list by using summing the parameters β, with each index β_i weighted by how "close" the new ingredient list is to those seen in the training set. If no recipes in the training set are "close" to the new ingredient list, the model will assign very little weight to the kernel function, and so the predicted rating will be close to zero. Because our feature vector length is either 768 or $3,866$, and we only have $5,000$ training examples, we can see by a dimensionality argument most recipes will not be "close" to each other. This is especially obvious in the bag-of-ingredients setting; most recipes will overlap by only a few ingredients, if at all. Thus, a possible

[2] I include this clarification because I was asked where my AUC statistics were by the teaching assistant grading me at the poster session, despite this being a regression problem.

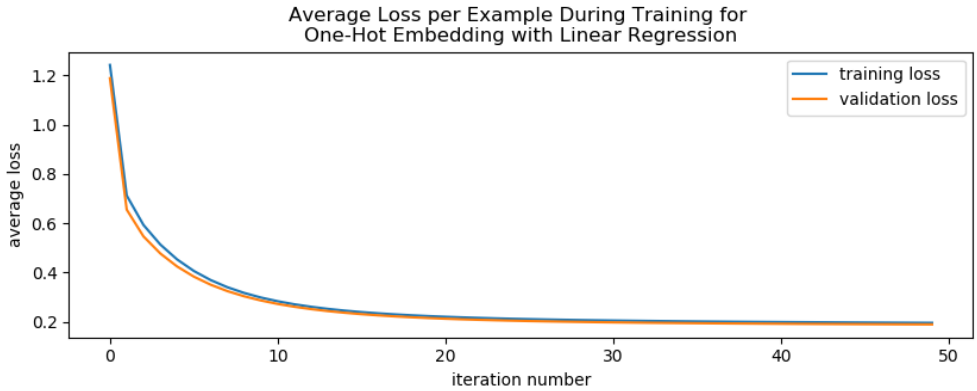

Figure 1: The loss over the training and validation sets as a function of number of iterations for linear regression. This shows good convergence and no overfitting.

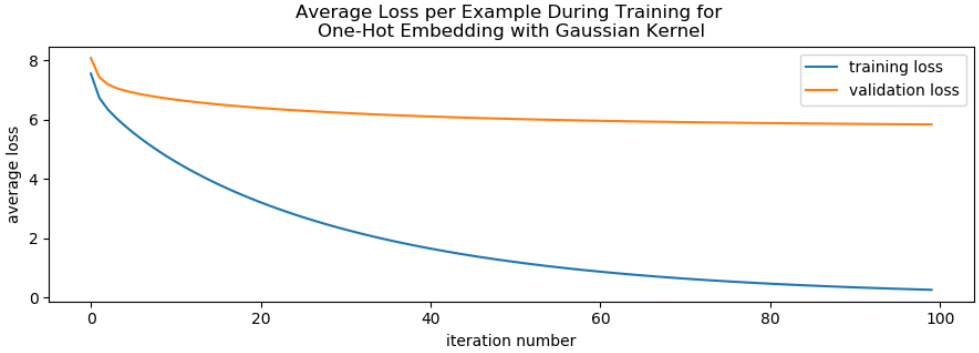

Figure 2: The loss over the training and validation sets as a function of number of iterations with a Gaussian kernel. This does not show signs of overfitting to the training dataset, yet still achieves poor poerformance on the validation set.

explanation for this behavior is that the recipes seen in the training set ovelap very little with those in the validation and test sets, and so the model cannot predict the rating for these recipes. Future work would explore this more fully.

6 Conclusion

This project aimed to predict the star rating of a recipe given the list of its ingredients. A linear regression model with a bag-of-ingredients approach achieved the best results, with a mean squared error of 0.1719, indicating that predictions were within 0.436 stars of the true prediction. The fact that this model achieved the best performance was somewhat surprising, and leads to questions for future work.

Future studies could examine increase the size of the training dataset. This may allow kernel methods to outperform linear regression. Exploring different embedding strategies, such as word2vec, ELMo, and GPT-2 could offer interesting results. Finally, implementing different kernel functions could improve performance.

References

[1] B. Prasad Majumder, S. Li, J. Ni, and J. McAuley, "Generating personalized recipes from historical user preferences," *Empirical Methods in Natural Language Processing (EMNLP)*, 2019.

[2] M. Harvey, B. Ludwig, and D. Elswiler, "You are what you eat: Learning user tastes for rating prediction," *Lecture Notes in Computer Science*, 2013.

[3] J. Freyne and S. Berkovsky, "Intelligent food planning: personalized recipe recommendation," 2010.

[4] C.-Y. Teng, Y.-R. Lin, and L. Adamic, "Recipe recommendation using ingredient networks," 2012.

[5] X. Mao, R. Yanghui, and Q. Li, "Recipe popularity prediction based on the analysis of social reviews," 2013.

[6] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *North American Chapter of the Association for Computational Linguistics*, 2019.