How to Train Your Palate

Victor Treaba, Herbert Li

Abstract

In this project, we will try to see whether we can predict recipe ratings based on features like ingredients used, cooking directions, and flavor composition by analyzing a dataset of over 20,000 cooking recipes. Our methods will be tested by applying several algorithms, including SVMs, Adaboost, and Neural Networks. We will also explore the question of whether or not we can learn to rank recipes—if we can determine if recipe A will be better than recipe B by using similar methods as in the classification case.

1 Introduction

Finding cooking recipes has never been easier. We all have almost instant access to hundreds of thousands of recipes, whether they be online or in print. However, finding *good* recipes is bit more difficult. After all, the only guaranteed way to tell whether a recipe is good is to make it yourself. As a result, we would like to tackle the problem of telling how good a recipe is without cooking it.

Broadly speaking, we will try to determine the quality of a recipe by predicting its rating. Due to the constraints imposed by our dataset, we will run experiments in a two, three, and four class setup by combining lower rated recipes. For each of these scenarios, we'll also compare the effectiveness of various methods of featurizing recipes. We will first consider basic methods for featurization (e.g. based on the presence or absence of certain ingredients). Gradually, we'll work our way to the other side of the spectrum, trying more complex methods that take into account the flavor profile, preparation, amount, and overarching category of each ingredient. To determine which methods and features work best, our experimental setups will be tested using several classifiers we covered in class, namely SVMs and Adaboost, as well as with Neural Networks.

In addition to the multi-class classification setting, predicting recipe ratings naturally leads us towards the question of ranking recipes. In this case, we will try to see whether it is possible to learn if recipe A is better than or equally as good as recipe B. Again, we'll consider the same

featurization methods as in the classification case, however, our ranking setup will use Adaboost and SVMs to test our various features and models.

This report is structured as follows: we will first take look at our dataset, break down our various methods of featurization, outline our goals, present our data and findings, analyze our results, and finally suggest additional work that could be done with regard to recipe classification and ranking.

2 How to Predict Recipe Ratings

2.1 Dataset and Featurization

Currently, most recipe APIs don't contain the full spectrum of information needed for our purposes. Some datasets don't contain ratings, while others don't include cooking directions. After a few failed attempts at web scraping, we found a packaged JSON of over 20,000 recipes online from Kaggle[1]. We're unsure of who compiled this dataset, as it is available from other sites as well, but the source of the original recipes is Epicurious.com[2]. With the dataset in hand, we then excluded recipes that we could not gather pertinent information from (e.g., some recipes were comprised solely of non-food ingredients, giving us a feature vector with all zeros, and some recipes had a "null" rating). This had little effect on the size of the dataset, reducing it from 20110 to 20035 workable data points.

Rounding ratings to the nearest whole number, the breakdown of ratings in our original dataset is as follows:

- 0 ratings: 1834 (9.43%)
- 1 ratings: 162 (0.85%)
- 2 ratings: 124 (0.65%)
- 3 ratings: 2011 (9.84%)
- 4 ratings: 13194 (65.50%)
- 5 ratings: 2710 (13.73%)
- Total recipes: 20035

This distribution is quite problematic for machine learning for two reasons:

1. It's *highly* skewed towards the upper echelon of ratings, which preliminary tests with SVM showed that this would cause most "predictions" to be merely guessing the same class (in this case, 4 ratings).
2. It's almost completely absent of 1 and 2 ratings, making it more difficult to accurately fit a model that would correctly classify these points.

In order to fix these issues, we decided on a broader 4-class classification model where we would push 0 and 1 ratings up to 2, based on the assumption that recipes rated 0, 1, and 2, are all similarly distasteful. Doing this gives us the following data distribution:

- 0, 1, 2 ratings: 2120 (10.93%)
- 3 ratings: 2011 (9.84%)
- 4 ratings: 13194 (65.50%)
- 5 ratings: 2710 (13.73%)
- Total recipes: 20035

Each recipe in the dataset includes fields like:

1. Cooking Instructions
2. Ingredients
3. Rating/5 stars (note that Epicurious has since switched to a 4 star rating system)
4. Title
5. Nutritional Information
6. Categories/Keywords

For this project, we primarily concerned ourselves with cooking instructions, ingredients, and the rating of each recipe. In order to create our feature vectors, we experimented with a couple methods of featurization, summarized below. Generally, our feature vectors varied based on how we decided to split up ingredients, our methods of counting them, and whether or not we decided to include cooking directions.

From the ingredients list of a recipe, we first wrote scripts that isolated the main ingredient and reduced them to their main nouns. For example, phrases like “extra virgin olive oil” would be reduced to just “olive oil”. This gave us a list of all ingredients, from which we would create a broad featurization method: `ALL`. To make sure this list is complete, we created a script that would scan through every ingredient, and see if it could be reduced to one of the main ingredients previously seen. If not, it would print out this new ingredient and we would manually add it into the list. While adding the main ingredients to the list, we also created rough categories for each of them (vegetables, meats, alcohols, etc.), in order to make it easier for us to sort them into ~70 categories, as well as giving us an easy way to categorize all the alcohols together without a second round of searching. Grouping alcohols (as many would have brand names and not be easily discernible as alcohol), singular and plural forms of ingredients, and regional variations of each ingredient, led us create a `SPARSE` vector creation method. From this, we moved on to create the ~70 buckets that we’ll refer to as `GROUPED`, by researching each of the non alcoholic ingredients, and sorting them into buckets based on their flavor and texture.

We used similar approaches in creating methods `WEIGHTS` and `DIRECTIONS`: heuristical groupings based on the assumption that more data would lead to more accurate predictions. These two methods are more complex than `EXISTENCE`, which is the most simple, a binary label denoting the presence or absence of an ingredient or category of ingredient. To extract `WEIGHTS`, we created conversion tables for volumes and weights. Going through the list of ingredients again, we did pattern matching on the substring leading up to the main ingredient noun, in order to isolate the volume or weight quantity required by the recipe (for example, "3 tablespoons"). Having those, we mapped all weights to grams, and for volumes we mapped all conversions in terms of metric US cups, then manually created another mapping to map US cups into grams. For most categories we could simply map the density of the category, because most ingredients in certain categories would share very similar densities, but for more complex ones showing significant variance, we would create a specific mapping for them. This is one of our most complex featurization models, that was made to distinguish between very similar recipes by taking into account errors that could arise due to misuse/overuse of ingredients. The way we extracted `DIRECTIONS` was similar. We again used pattern matching to detect the existence of certain keywords in a recipe's cooking instructions, for example "bake" or "sauté". We also implemented an intermediate step between `EXISTENCE` and `WEIGHTS` which we named `FREQUENCY`. Here, the values of the vectors are determined by how many times we run into an ingredient of a certain category (categories being those of `GROUPED`), based on the assumption that not only does a recipe depend on the types and flavor of the ingredients used in it, but also how much of each type or flavor.

2.2 Our Goals

Our work aims to use machine learning to answer one fundamental question and explore other possible applications of our results: can we objectively look at a recipe, and its details like ingredients and directions, to try to determine its quality? If this is indeed possible, then we should also be able to rank recipes against each other and analyze flavor trends. We assume there is a correlation between the texture and flavor diversity of foods and their ratings, but this is difficult to confirm because of the subjective nature of ratings. For example, only those with semi-strong opinions or those who are outspoken tend to rate a recipe, not to mention the possible variability in their responses due to personal preferences with regard to cuisine styles. Should this experiment bear fruit, we could infer that there may be common combinations of flavors that are generally appealing to the average palate, which may be used in other areas such as studying the evolution of taste within cultures. The way we decided to split our data and generate our classifications reflects these ideas.

In order to approach these questions, we decided on the following setup for our experiments:

- The labels in the data are still very skewed towards high ratings, so we trained classifiers on undersampled data so that the distribution across the board was equal. The distribution of our test data was made to be equal as well. Note that the training and test points were chosen randomly across every experiment we ran.
- In addition to undersampling, we also looked at results obtained from combining low ratings into one larger class, resulting in two, three, and four class classification scenarios. Specifically, for the two class case we had buckets containing {0, 1, 2, 3} and {4, 5} ratings (good and bad recipes). For the three class case we had buckets containing {0, 1, 2, 3}, {4}, and {5} star ratings (bad, average, and good recipes). Finally, for the 4 class case we had buckets containing {0, 1, 2}, {3}, {4}, and {5} star ratings.

For featurization, we decided on the following methods:

1. Ingredient Grouping
    a. `ALL:` ~800 features, each corresponding to a unique ingredient string
    b. `SPARSE:` ~600 features, each corresponding to a unique ingredient, here we grouped singular and plural forms together, and we grouped all alcoholic beverages together
    c. `GROUPED:` ~70 features, each corresponding to a group of ingredients that all have similar flavors (sweet, savory, bitter, etc.) and textures
2. Value of Feature
    a. `EXISTENCE:` Binary numbering (0-1) representing the presence of a given feature column (i.e. ingredient or ingredient category)
    b. `FREQUENCY:` Counter representing how many many times an ingredient falls within one of the ~70 `GROUPED` categories
    c. `WEIGHTS:` Floating point number representing the percentage of the mass in grams of the ingredient or ingredient group
3. Cooking Directions
    a. `DIRECTIONS:` ~20 features, each corresponding to a unique preparation (e.g. sauté, boil, bake, roast), also features for cooking times and oven temperatures

We ran our experiments using SVM, Adaboost, and Neural Networks. SVM was run using a polynomial kernel and with parameters C and d obtained from running grid search, using 10-fold cross-validation. Adaboost was run using decision stumps as our base classifier with the number of iterations obtained using grid search, again using 10-fold cross-validation. We included Neural Networks to see if deep learning would yield significantly different results, which were obtained by finding an optimal value for the number of neurons in one or two hidden layers. Our Neural Networks used a DNN-Classifier with an Adagrad gradient optimiser. In the ranking

case, we tested ranking using Adaboost and SVM in a similar setup as in the case of classification, with labels -1, 0, and 1 denoting if recipe A is worse than, equal to, or better than recipe B, respectively.

3 Results and Analysis

The results of our experiments on the four, three, and two-class classification problems can be found in figures 1, 2, and 3, respectively. The results of our experiments on the ranking problem can be found in figure 4. Our test accuracy was calculated using the 0-1 loss function. This metric is pretty harsh for multiclass classification, so we also calculated a closeness score—the average classification error—detailing how many classes off our prediction was from the actual ratings. The average classification error we obtained across all of our test runs was about 1 or less, meaning that our model would either predict a 2 or a 4 for a 3 starred recipe.

4 Class Classification	**Feature Vector Details**	**Test Accuracy (%)**
SVM	GROUPED EXISTENCE DIRECTIONS	38.09
	GROUPED EXISTENCE	40.95
	GROUPED FREQUENCY DIRECTIONS	39.00
	GROUPED FREQUENCY	39.65
	SPARSE EXISTENCE DIRECTIONS	40.77
	SPARSE EXISTENCE	38.76
	SPARSE WEIGHTS DIRECTIONS	33.21
	SPARSE WEIGHTS	23.58
	ALL EXISTENCE	38.26
Adaboost	GROUPED EXISTENCE DIRECTIONS	40.60
	GROUPED EXISTENCE	33.22
	SPARSE EXISTENCE	35.07
	ALL EXISTENCE	34.23
Neural Networks	GROUPED EXISTENCE DIRECTIONS	39.50
	GROUPED EXISTENCE	38.50
	GROUPED WEIGHTS DIRECTIONS	41.63

	GROUPED WEIGHTS	37.10

(Figure 1: 4-Class Experimental Results)

In the 4-class scenario, we saw that we were beating the distribution of the test set by at most 16% (since our distribution was an even 25% split). We obtained our best SVM results using GROUPED EXISTENCE (40.95% accuracy), and our best Adaboost results were comparable, using GROUPED EXISTENCE DIRECTIONS (40.60% accuracy). Although never covered in class, we included Neural Networks and we were pleasantly surprised to see it perform as well as SVMs and Adaboost (41.63% accuracy using GROUPED WEIGHTS DIRECTIONS).

We initially included cooking directions thinking that it would lead to comparable, if not better, results. For the 4-class scenario, this held to be true. GROUPED EXISTENCE DIRECTIONS performed relatively as well as just GROUPED EXISTENCE, for example. In certain cases, for example with SPARSE WEIGHTS DIRECTIONS versus just SPARSE WEIGHTS, we saw that the addition of directions lead to significantly better test accuracy. Another interesting result that we noticed was that using WEIGHTS lead to significantly worse test accuracy compared to just using EXISTENCE. Initially we thought that WEIGHTS would lead to *better* accuracy, for example, just using sugar versus using a pound of sugar should play a factor in the rating of a recipe. Finally, we noticed was that using FREQUENCY did not significantly improve or worsen test accuracy compared to just using EXISTENCE. Initially we thought that FREQUENCY would lead to *better* accuracy, as logically, the amount of times a type of ingredient was used should impact the quality of a recipe.

Comparing the performance of GROUPED vs. SPARSE vs. ALL, we did see not see any noticeable differences in test accuracy. This is surprising, especially for GROUPED vs. SPARSE and GROUPED vs. ALL cases as we initially thought that grouping ingredients of similar flavor and texture would lead to more accurate results.

3 Class Classification	**Feature Vector Details**	**Test Accuracy**
SVM	GROUPED EXISTENCE DIRECTIONS	43.66
	GROUPED EXISTENCE	42.40
	GROUPED FREQUENCY DIRECTIONS	44.93
	GROUPED FREQUENCY	45.43
	SPARSE EXISTENCE DIRECTIONS	50.61
	SPARSE EXISTENCE	52.56

	SPARSE WEIGHTS DIRECTIONS	40.85
	SPARSE WEIGHTS	30.61
	ALL EXISTENCE	49.76
Adaboost	GROUPED EXISTENCE DIRECTIONS	42.68
	GROUPED EXISTENCE	43.78
	SPARSE EXISTENCE	44.63
	ALL EXISTENCE	44.51
Neural Networks	GROUPED EXISTENCE DIRECTIONS	46.00
	GROUPED EXISTENCE	42.93
	GROUPED WEIGHTS DIRECTIONS	40.27
	GROUPED WEIGHTS	40.13

(Figure 2: 3-Class Experimental Results)

In the 3-class scenario we again saw many of same trends as we saw in the 4-class scenario. We were again beating the distribution by at most 19%. Again, we noticed that the addition of DIRECTIONS led to comparable, if not better, results. Once again, we saw that using WEIGHTS led to comparable if not significantly worse test accuracy compared to just using EXISTENCE. Similar results held true for FREQUENCY versus just EXISTENCE, as it resulted in no significant change in test accuracy. In this experiment, we obtained our best SVM results using SPARSE EXISTENCE (52.56% accuracy), our best Adaboost results were obtained by using SPARSE EXISTENCE (44.63% accuracy), and our best Neural Network results were obtained using GROUPED EXISTENCE DIRECTIONS (46.00%). Comparing the performance of GROUPED vs. SPARSE vs. ALL, in the 3-class case we saw a noticeably worse test accuracy using GROUPED. This is surprising, as we initially thought that grouping ingredients of similar flavor and texture would lead to at least comparable, if not better, results.

2 Class Classification	**Feature Vector Details**	**Test Accuracy**
SVM	GROUPED EXISTENCE DIRECTIONS	60.14
	GROUPED EXISTENCE	59.62
	GROUPED FREQUENCY DIRECTIONS	59.88
	GROUPED FREQUENCY	60.38
	SPARSE EXISTENCE DIRECTIONS	65.43

	SPARSE EXISTENCE	63.27
	SPARSE WEIGHTS DIRECTIONS	58.10
	SPARSE WEIGHTS	49.22
	ALL EXISTENCE	64.83
Adaboost	GROUPED EXISTENCE DIRECTIONS	62.18
	GROUPED EXISTENCE	62.55
	SPARSE EXISTENCE	61.70
	ALL EXISTENCE	61.94
Neural Networks	GROUPED EXISTENCE DIRECTIONS	58.51
	GROUPED EXISTENCE	57.64
	GROUPED WEIGHTS DIRECTIONS	57.42
	GROUPED WEIGHTS	59.72

(Figure 3: 2-Class Experimental Results)

In the 2-class scenario, we again saw similar trends as in the 3 and 4 class scenarios. Again, we ended up only beating the distribution by at most 15%. In this setting, our best SVM accuracy was obtained using SPARSE EXISTENCE DIRECTIONS (65.43%), our best Adaboost accuracy was obtained using GROUPED EXISTENCE (62.18%), and our best Neural Network accuracy was obtained using GROUPED WEIGHTS (59.72%). Comparing the performance of GROUPED vs. SPARSE vs. ALL, as in the 4-class scenario we did see not see any noticeable differences in test accuracy.

Ranking	**Feature Vector Details**	**Test Accuracy**
SVM	GROUPED EXISTENCE	42.24
	GROUPED EXISTENCE DIRECTIONS	42.80
Adaboost	GROUPED EXISTENCE DIRECTIONS	44.39
	GROUPED EXISTENCE	43.50
	GROUPED FREQUENCY	42.54
	GROUPED WEIGHTS	41.77
	SPARSE EXISTENCE	45.77

	ALL EXISTENCE	46.47

(Figure 4: Ranking Experimental Results)

Finally, in the ranking case, SVMs and Adaboost performed roughly 10% better than just randomly assigning -1, 0, 1 labels to test points. As we saw in the classification scenarios, the inclusion of DIRECTIONS led to comparable or even better test accuracy. On the other hand, the inclusion of WEIGHTS instead of EXISTENCE led to comparable or worse test accuracy. Furthermore, the use of FREQUENCY instead of EXISTENCE resulted in no significant change in test accuracy. Comparing the performance of GROUPED vs. SPARSE vs. ALL, again, we did see not see any noticeable increase or decrease in test accuracy.

4 Discussion and Related Work

As predicted, across the board we saw that including DIRECTIONS led to better results. However, to our surprise, using WEIGHTS instead of EXISTENCE led to only comparable, if not worse results. Finally, using FREQUENCY instead of EXISTENCE resulted in no significant differences. Our most surprising results were obtained when GROUPED, SPARSE, and ALL were compared. Switching between these three methods did not lead to any noticeable increase or decrease in test accuracy across all experiments. If anything, using GROUPED led to worse test accuracy. Furthermore, no matter if we used SVMs or Adaboost or Neural Networks, we were able to predict ratings better than guessing a random class, but our results were not accurate enough to classify this rating problem as definitely learnable. The same can be said for ranking recipes.

We'll now compare our results to other work in recipe classification. The most relevant work we could find was conducted by Yu, Zhekova, Liu, and Kubler[3] in which they used a similar dataset to predict recipes on a 1-4 rating scale using SVMs. In their experiments, they found that the best predictions were obtained by incorporating actual reviews into their training and test sets. Our dataset did not include such information, furthermore we didn't think that reviews *should* be included to begin with. After all, the content and wording of each review directly corresponds to the rating of a recipe. For example, a review that says something along the lines of "That cake was so good" alludes to higher rating. While this previous work had a reported accuracy of ~62%, it is also worth noting that this was achieved using skewed data, with ~56% of points belonging to just one class. Although this model does seem reliable, it only performs ~6% better than the distribution, and without knowing the distribution of predictions, we cannot know if it is predicting or simply guessing. Even if recipes in the real world have this kind of distribution, the predictor is very limited in its decisions.

5 Conclusions and Further Work

In this project, we tried to see if ingredients, cooking directions, and flavor composition could be used to predict recipe ratings. However, we learned that this problem is not so simple because the taste of a recipe is subjective.

The discussion section touches upon several results that were surprising. Here, we'll offer some possible explanations. The fact that using `WEIGHTS` instead of `EXISTENCE` led to only comparable, if not worse results, may indicate that recipes are, after all, only guidelines. The weights of actual ingredients may not actually matter, as everyone has their own unique preferences with regards to food. For instance, a person may like spicy food so they may deviate from a recipe and elect to add more spice. Furthermore, using `FREQUENCY` instead of `EXISTENCE` resulted in no significant differences, which may also be due to similar reasons. In order to explain why using `GROUPED` led to only comparable, if worse results, compared to `SPARSE` and `ALL` may be due to the fact that there is some information lost when ingredients are lumped together. This might show that certain ingredients may not be as similar as they may seem, and that each ingredient may have slight nuances in their flavor and texture that are lost when we simply lump them together. Another possible explanation could be simply that our groups were too general, that jumping from ~800 ingredients to just ~70 categories is way too steep of curve. Perhaps it would be better to test using more categories.

In the future, we would also like to explore more methods of featurization. One possibility is to look at features composed of bigrams and trigrams of ingredients. For example, "lamb and rosemary" might be one such bigram, and "carrots, celery, and onions" might be a trigram. We could also consider which bigrams and trigrams to use. We would also like to incorporate some natural language processing methods in our classification and ranking scenarios. The scope of recipe classification could also be reduced. One of our initial concerns was that the data spanned across too many dishes and recipes. For instance, it would be interesting to see whether one could predict ratings for recipes that use chicken as their main ingredient. However, the narrower the focus, the harder it might be to obtain enough training and test data. After all, just how many ways are there to make lasagna? But if one could obtain a base learner that was decent at rating chicken recipes, and one that was decent at rating pasta recipes, perhaps we could combine these learners to create a stronger one that would be good at not only learning chicken and pasta, but also any crossover between the two.

6 References

1. **“**Epicurious - Recipes with Rating and Nutrition” https://www.kaggle.com/hugodarwood/epirecipes
2. https://www.epicurious.com/
3. N. Yu, D. Zhekova, C. Liu, and S. Kubler. Do good recipes need butter? predicting user ratings of online recipes. In Proceedings of the Cooking with Computer workshop at the International Joint Conference on Artificial Intelligence (IJCAI2013), 2013. URL: http://cl.indiana.edu/~skuebler/papers/epi13.pdf