Recipe for Success
Optimizing meals under dietary constraints

Benjamin Share, James Ordner, and Zack Cinquini
benshare, jordner, and icinquin at stanford.edu

CS 229 Project Final Report
December 15, 2017

Abstract

When faced with thousands of search results, finding a healthy recipe online that aligns with your personal preferences and dietary restrictions can be an intimidating task. Using data from allrecipes.com, we construct a machine learning model that maps recipes (as captured by their constituent ingredients) to success as measured by online reviews. Using vector representations of our ingredients, we propose a methodology for detecting logical ingredient substitutions.

1 Introduction

With the internet at every aspiring cook's fingertips, finding the best recipe among a sea of options can be challenging. In particular, it can be difficult to pinpoint which aspects of a dish are crucial to its success, and which are more flexible—where can a cook afford to upgrade to a healthier option, or what can be replaced to abide by a dietary restriction?

Our project aims to streamline this process by applying machine learning algorithms to the challenge of recipe selection. In particular, we explore methods for predicting recipe success given its attributes. For the scope of this project, we classify success based on a recipe's online rating, and we use the list of ingredients as features for each recipe.

To the challenge of recipe success prediction, we apply the Naive Bayes algorithm and a neural network. These techniques assign buckets to the continuous range of ratings, and calculate probabilities that a recipe will fall within a given range. We approach ingredient substitution by representing ingredients as vectors and adapting the word2vec approach, where neighboring vectors become candidates for exchange [Mikolov *et al.*].

2 Related Work

By virtue of its direct connection to human health, the issue of online recipe selection has attracted substantial attention from the machine learning community. In particular, it has been found that ingredient and thus nutrition data alone provide a good indicator of positive online ratings on food-focused social networks [Teng *et al.*]. Further research suggests that ingredient classification and substitution recommendation could be a valuable tool in combating the ongoing global obesity epidemic [Freyne *et al.*].

Additionally, past CS 229 students have explored recipe classification [Arffa *et al.*], substitution recommendation [Ivanov], and recipe success prediction [Arffa *et al.*, Ivanov]. Our project tackles the issue with a novel dataset and vector embedding-based approaches to detecting ingredient alternatives.

3 Dataset and Features

We scraped recipes from allrecipes.com, a website for casual and experienced cooks to share and rate recipes and the largest food-focused social network. We elected to focus specifically on lasagna, brownies, and cookies, as preparation of these dishes plays a minimal role in their overall quality—taste and success relies primarily on ratios of ingredients and less on the technical aspects of culinary preparation.

We extracted a set of 259 lasagna recipes, 383 brownie recipes, and 4707 cookie recipes. Of these, we elected to discard recipes with fewer than three reviews, leaving 210 lasagna, 298 brownie, and 3822 cookie recipes. Each recipe in our dataset is associated with a title, a quantity of reviews, an aggregate score from one to five stars, a number of servings, and a list of ingredients.

After collecting the raw data, we processed the lists of ingredients to extract the quantity. All volume measurements were converted into fluid ounces, all weights converted to ounces, and volumes and weights were made comparable with the approximate conversion of 4 fluid ounces = 3 ounces as calculated by averaging the densities of the ingredients that are most commonly measured by volume or weight. Quantities were then normalized according to the number of servings the recipe produced.

To extract the name of the ingredient from the raw data, we curated a list of ingredients that appeared in at least ten recipes (three for brownies and lasagna). Our parser identifies and coalesces identical ingredients referred to by multiple names (i.e. "flour", "all purpose flour", and "all-purpose flour"). To make full use of our limited data, we also grouped ingredients with nearly-identical functions (i.e. "salt" and "sea salt").

4 Methods

Our methods and results fall into two categories according to the goals of the project: rating prediction and substitution recommendation. Of our four primary methods for this project, two are concerned with the first task, and two with the second.

4.1 Naive Bayes

Naive Bayes is the first of our two rating prediction algorithms. Our implementation treats labels (recipe ratings) as predictive priors for ingredient presence/quantity. We use the multinomial event model, in which ingredients are assumed to be included with a given probability for each of label category. The likelihood of a given recipe conditioned on a label can thus be calculated; by comparing these probabilities (scaled by the categories' respective priors), we can derive a guess for the most likely label.

In particular, our implementation includes several adaptations to the standard algorithm. First, we make use of Laplace smoothing to avoid unwanted effects from rare ingredients. Second, we make use of a flexible bucketing schema, in which continuous ratings (ranging from 1.0 - 5.0) are placed into discrete ranges.

Parameters are calculated as follows:

$$\phi_{k|y=1} = \frac{\sum_{i=1}^{m}\sum_{j=1}^{n_i} 1\{x_j^{(i)} = k \wedge y^{(i)} = 1\} + 1}{\sum_{i=1}^{m} 1\{y^{(i)} = 1\} n_i + |V|}$$

$$\phi_{k|y=0} = \frac{\sum_{i=1}^{m}\sum_{j=1}^{n_i} 1\{x_j^{(i)} = k \wedge y^{(i)} = 0\} + 1}{\sum_{i=1}^{m} 1\{y^{(i)} = 0\} n_i + |V|}$$

$$\phi_y = \frac{\sum_{i=1}^{m} 1\{y^{(i)} = 1\}}{m}$$

where m is the number of training examples, each with some n_i words drawn for $|V|$ possibilities.

Once trained, we made use of the algorithm in three primary ways.

1. First, as a classifier. A test input is fed in, the likelihood of producing that input for each label is calculated, and then the most probable label is returned.
2. Second, the trained model can be used to generate assessments of overall ingredient scoring. We derived scores for ingredients as the average of the ratings corresponding to recipes in which they appeared, weighted by their quantity in that recipe. While this method has obvious limitations (most prominently rare ingredients being high-variance) it produces surprisingly plausible results.
3. Lastly, we also made use of the model's generative ability. By specifying a rating and a discrete unit with which to include components of the recipe (corresponding to words in a typical NB model), the model can utilize its trained probabilities to generate a complete recipe for the given category. This method is fast, applicable to all ratings, and engagingly random—since each recipe is a sampling from the learned probability distribution, it can provide many unique examples.

4.2 Rating Net

Rating Net is our deep learning approach to the same problem of recipe rating prediction. Ingredient quantities corresponding to a recipe are fed into the network; they undergo matrix multiplications, pass through the hidden layer, and are compared with the recipe label to generate a loss term. This then allows us to backpropagate onto the matrix weights, improving predictions in the future.

We altered the structure of the network at various points, changing features including the number of nodes in the hidden layers and the number of these layers (this process will be addressed further in the discussion section below). Ultimately, we achieved best performance using the following model with one hidden layer:

$$h^{(i)} = \sigma(x^{(i)} \times W_1 + b_1)$$
$$\hat{y}^{(i)} = h^{(i)} \times W_2 + b_2$$

The components have the following dimensions:

$x^{(i)}$	B $\times N$
W_1	N $\times H$
b_1	B $\times H$
$h^{(i)}$	B $\times H$
W_2	H $\times 1$
b_2	B $\times 1$
$\hat{y}^{(i)}$	B $\times 1$

Note that B is the batch size, N is the number of features (ingredients), and H is the size of the hidden layer.

Our implementation allows the network to preform either regression (predicting a rating value) or classifier (predicting a rating bucket). The dimensions given above are for the regression model. In this paradigm, the loss is given as the squared difference between the prediction $\hat{y}$ and the true label y, averaged over the batch. In the classification case, the dimensions above are modified such that W_2 has size $H \times K$ and b_2 is of dimension $H \times K$ where K is the number of buckets the ratings have been bucketed into. Here, loss is given by the sparse softmax cross entropy over this output and the true label.

We did fairly extensive cross-validation on the network using the dev set, most prominently to combat overfitting. This was mitigated by including a regularization cost in the continuous and discrete models. In both cases, this took the form of the sum of the L2 norms of the weight matrices, multiplied by a scaling factor γ. Tuning the paramaters by trial and error yielded $H = 20$, $B = 20$, and $\gamma = 10^{-5}$. We achieved our best performance with the Adam optimizing algorithm at a learning rate of 0.02.

4.3 Recursive Vector Embeddings

Recursive Vector Embeddings is the first of our two techniques intended to allow us to capture ingredients' pertinent information in the form of vector embeddings. We developed this method as a baseline test for the more complicated word2vec algorithm (see section 4.4). Its input is the list of recipes as given by ingredient quantities without the accompanying ratings and returns vector representations for each ingredient.

For each recipe included in the training data, the program extracts the co-occurrence counts for each pair of ingredients, and constructs a symmetric matrix of these counts. Ingredients that occur in recipes together frequently and in high quantities have high values in the corresponding indices of the matrix; diagonal entries corresponding to co-occurrence between an ingredient and itself are set to 0. The model then multiplies the co-occurrence counts for each ingredient by the current embeddings for all the ingredients, generating a summation of an ingredient's 'context'. These context vectors are then multiplied by a scaling factor and added to the previous vector representations. All updates occur simultaneously and the process repeats until convergence.

4.4 word2vec Network

Lastly, we attempted to provide a more robust generator of ingredient embeddings using a neural network. Drawing from the word2vec algorithm, we designed the following network:

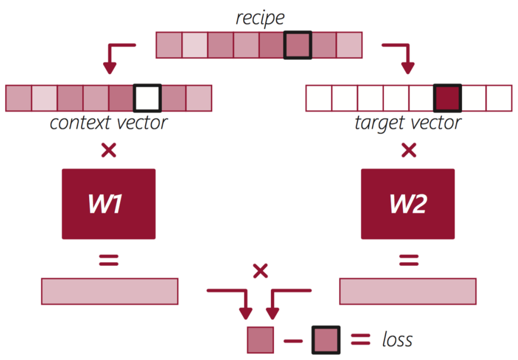


Figure 1: *Our network for generating ingredient embeddings.*

The inputs to the network are a list of recipes, represented by vectors of ingredient quantities. For each recipe, it generates a training instance corresponding to each of the available ingredients. For each (recipe, ingredient) pair, it generates a 'context' vector, equal to the recipe vector with the entry corresponding to the chosen ingredient set to 0, a 'target' vector that's one-hot in the given location, and a scalar label y equal to the original value of the recipe at that location. Each is modified using the corresponding weight and bias to produce context and ingredient "vectorizations." The dot product of these values produce the network's prediction $\hat{y}$. Loss is given by the absolute difference of y and $\hat{y}$.

Intuitively, this formulation encourages the weight matrices to function as vector representations of the ingredients. Each row is consistently multiplied by the recipe entries corresponding to a particular ingredient; the outputs from the first and second multiplications are thus the embeddings' rendering of the context and target, respectively. As in the word2vec model, the network is rewarded for making co-occurring ingredients resemble each other—since these vectorizations are dotted together, they are rewarded for their similarity.

Finally, we use Euclidean distance to identify ingredient similarity and suggestion appropriate substitutions.

5 Discussion

5.1 Rating prediction

After tuning hyper-parameters on the dev set, we tested our Naive Bayes model as a predictor on the test set. We also ran a train-on-train test to establish an upper bound for expected performance. Error rates on the test set by number of buckets are plotted in Figure 2. As expected, the model achieves perfect accuracy with one bucket, then generally increases in error as we increase our accuracy demand by narrowing bucket width.

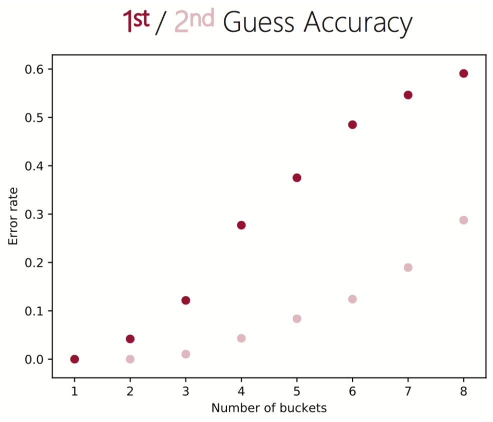


Figure 2: *Error rates on the test data by number of buckets. Red points indicate error rates on first choice prediction; pink points are the 'lenient' error rate, in which the model's first and second guesses are given credit.*

Naive Bayes serves as a valid baseline. It achieves passable accuracy—notably, in the 4-bucket range corresponding to full-star buckets. The error metric taking into account second-guesses also significantly improves this performance, indicating that the model may have a better intrinsic sense of ingredient value than the initial values suggest. Since the results of train-on-train error were significantly lower, it is likely that additional data would improve generalization.

We also used the trained Naive Bayes model to output the best and worst ingredients for a given dish type. We took the average of each ingredient's recipe ratings weighted by ingredient use, then calculated the highest- and lowest-scoring ingredients. For the model trained on the brownie data set using the four-bucket schema, this yielded:

Best ingredients	Worst ingredients
chocolate cake mix (3.87)	coconut sugar (1.63)
caramel (3.80)	avocado (1.70)
cream cheese (3.75)	black bean (1.81)
peanut butter (3.66)	vegan chocolate chip (1.90)
brownie mix (3.58)	cream of tartar (1.97)

While it's not clear how to precisely assess the accuracy of these outputs, they seem pleasingly intuitive—for one, the authors certainly would be more tempted by caramel chocolate cake brownies than coconut avocado ones.

We also used the probabilities calculated by Naive Bayes to generate novel recipes for a given rating:

Sample Generated 5-Star Cookie Recipe

2 tbsp vegetable oil	2 tbsp almond
2 tbsp condensed milk	4 tbsp of choc. cake mix
2 tbsp marshmallow	2 tbsp cocoa powder
2 tbsp brown sugar	2 tbsp raisin
2 tbsp walnut	2 tbsp pumpkin
2 tbsp shortening	2 tbsp confectioners' sugar
1 1/3 cups all-purpose flour	2 tbsp water
1/2 cup butter	3/4 cups semi-sweet choc.
1 floz vanilla extract	3 tbsp peanut butter cup
3/4 cups of white sugar	3 eggs

The Naive Bayes independence assumption means that this model is only able to determine the quanitity of each ingredient independently, resulting in an amalgamation of many varied tastes—marshmallows, pumpkin, walnuts, almonds, and raisins are rarely combined in conventional cookie making. Still, the ratios of key ingredients such as flour, butter, and eggs are consistent with real recipes.

Our second predictive method was our ratings neural network. Despite the network's success at assimilating data and lowering training loss, it generalized fairly poorly, showing the classic signs of overfitting—decreased training loss coupled with stagnant or increased dev loss, rising weight norms, etc. As a result, we added regularization terms and decreased the size/number of hidden layers. While this did solve the problem of overfitting, it did mean the model became somewhat less sophisticated. Ultimately, the continuous version of the model achieved a test loss of 0.28, which corresponds to an average of around half a star off from the truth.

The classification version of the model achieves 73% first-guess accuracy on the four-bucket breakdown, a shade above the baseline performance at this value. One caveat is that, since the ratings tend to cluster in the 4-star range, the model achieves this accuracy by biasing strongly toward guessing this bucket; most of its error comes from guessing the highest bucket for samples whose true ratings are lower. To address this issue, we hypothesized that the true label distribution followed what was approximately a normal distribution. By converting the labels to a standard uniform distribution scaled between 1 and 5, we force the model to adapt more. The results are pictured in Figure 3.

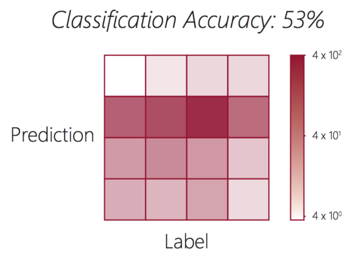


Figure 3: *Rating network predictions vs. ground truth*

The main diagonal (from bottom left to to right) corresponds to correctly-classified instances; in total, the network classified a little more than half correctly. While this isn't as good as might have been hoped, it does show the network performing significant generalization. The main challenge is once again the network's default tendency to prefer the most common bucket rating (in this case, the second row), a result of the steps we took to prevent overfitting. Perhaps our biggest priority moving forward would be continued attempts to improve the balance between these two factors.

5.2 Substitution Recommendation

The more nuanced word2vec approach produced far more intuitive results than the recursive embedding model. While there did appear to be noticeable structure in the stripped-down version's embeddings (for instance, it appeared to successfully cluster dairy products, as well as sugar sources), it failed a number of our benchmarks for commonsense identification.

One likely explanation for the neural network's improved performance is its use of two distinct weight matrices. One matrix (W1 in our formulation) is the 'out-vectors' of ingredients; multiplying by a recipe produces the context embedding. The other (W2) is the 'in-vectors'; multiplying it by the target vector produces the target embedding for the specified ingredient. As discussed in the word2vec literature, this strategy allows the model to capture multiple notions of a token's meaning: the 'out-vector' captures the way it changes the meaning of a sentence, and the 'in-vector' describes the types of contexts in which it occurs.

To see the importance of this distinction, consider two ingredients like eggs and cinnamon. Mathematically, the model predicts an ingredient to be present when the context and target embeddings are similar. Since both eggs and cinnamon are likely to occur in many standard cookie recipes, their target embeddings—generated by their in-vectors—should match the contexts of these recipes well. However, knowing that a recipe contains eggs does not change one's estimates of whether most other ingredients will be included, since eggs are generally ubiquitous. This means that its contribution to the context embedding should be small, and thus so should its out-vector. On the other hand, cinnamon imparts a much more specific flavor to a dish, meaning its inclusion should change the context embedding by a greater amount—and its out-vector should be correspondingly larger. Since the neural network utilizes separate weight matrices and the recursive formulation does not, it is able to capture this added level of complexity.

Additionally, we used these embeddings to generate suggested substitutions for ingredients by minimizing euclidean distance. For instance, the algorithm's output for shortening in cookies is shown below:

Shortening

Best substitutes	Worst substitutes
unsalted butter	chocolate candy kisses
walnut	peanut butter
water	brown sugar
vegetable oil	all-purpose flour
confectioners' sugar	peanut butter cup

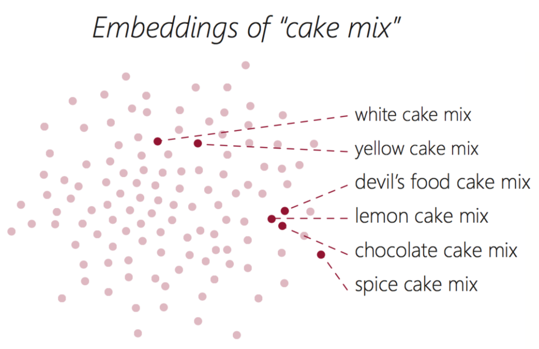
Ginger

Best substitutes	Worst substitutes
sunflower seeds	semi-sweet chocolate
nutmeg	white sugar
almond extract	brown sugar
cream of tartar	all-purpose flour
chocolate syrup	peanut butter cup

These algorithm's suggestions are encouraging, as they are fairly intuitive almost across the board. One issue with the model's output is that rare ingredients tend to be less recommended. This may stem from the fact that an ingredient's embedding magnitude is impacted by the number of updates it undergoes, which is in turn determined by its frequency of appearance. This means that the model tends to be less sure about their properties. In particular, the 'worst substitute' predictions should be taken with a grain of salt, as they are likely biased towards ingredients with high vector norms; since the embeddings are origin-centered, these will tend to be the farthest away. Explicitly incorporating information about the angles between vectors may help address this issue—as would more training data.

6 Conclusions and Future Work

Overall, we were relatively pleased with our results. While rating prediction turned out to be a more challenging problem than we had anticipated, both Naive Bayes and our neural net achieved reasonable accuracy. Both methods also produced sensible auxiliary results, most notably the ingredient scores and recipe generation above. Meanwhile, we were very satis-



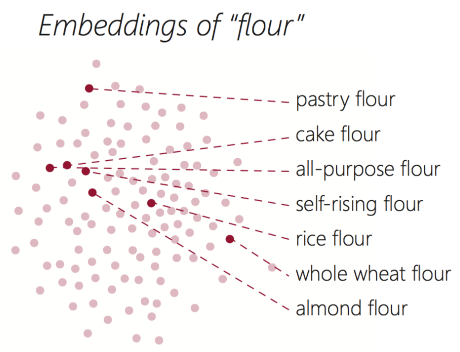


Figure 4: *Our word2vec model's embeddings of 'flour' ingredients projected into two dimensions with t-SNE. Whole wheat flour, with its distinctive texture, is expected as an outlier.*

fied with the performance of our ingredient embedding techniques. In particular, the word2vec network produced what appear to be genuinely substantive embeddings, allowing for helpful substitution recommendations. The use of projection algorithms like t-SNE was also extremely helpful in verifying the output's legitimacy.

The most logical next step would involve an integration of our tools into an end-to-end constrained recipe generator. This would be reasonably straightforward, since the challenging facets of the task are generally subsumed by the algorithms we designed. However, there is additional room for experimentation with optimization under constraint. For instance, one strategy could involve using the rating network to optimize rating with respect to input, with certain elements' updates set to 0. That way, a certain initialization value could be maintained and the network could adapt accordingly. Additionally, access to more data—both within dish type and of additional dishes—would allow us to continue to improve the performance of all models. It would also open the door to additional lines of inquiry, such as the various algorithms' ability to deal with dishes that contain discrete subcategories.

References

ABADI ET AL., 2015. TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

ARFFA, T., LIM, R., AND RACHLEFF, J. 2016. Learning to cook: An exploration of recipe data. *CS 229*.

FREYNE, J., BERKOVSKY, S., AND SMITH, G. 2011. Recipe recommendation: Accuracy and reasoning. In *UMAP*, Springer, 99–110.

IVANOV, I. 2015. What makes a good muffin? *CS 229*.

JONES, E., OLIPHANT, T., PETERSON, P., ET AL., 2001–. SciPy: Open source scientific tools for Python.

MIKOLOV, T., CHEN, K., CORRADO, G., AND DEAN, J. 2013. Efficient Estimation of Word Representations in Vector Space. *ArXiv e-prints* (Jan.).

RUDER, S. 2016. An overview of gradient descent optimization algorithms. *CoRR abs/1609.04747*.

TENG, C.-Y., LIN, Y.-R., AND ADAMIC, L. A. 2012. Recipe recommendation using ingredient networks. In *Proceedings of the 4th Annual ACM Web Science Conference*, ACM, New York, NY, USA, WebSci '12, 298–307.

7 Contributions

Zack: I made the scraper and crawled allrecipes.com to generate our data sets. Then, I built the framework for post-processing the ingredient lists that extracts the quantity and type of food for input into our machine learning algorithms, including the scheme for normalizing the inputs and filtering the data. I helped troubleshoot our methods, especially those involving neural networks, and made the poster and many of the figures.

James: I took the raw data scraped from allrecipes.com and formatted the ingredients into a cleaner format that mapped raw ingredient names to more common labels that could be read by the learning algorithms. I created several different mapping for each set of recipes, each with a different level of specificity. For example, the most broad mapping grouped all ingredients that were somewhat related under the same label, resulting in a small number of labels, and the most precise mapping grouped only very similar ingredients, resulting in a large number of labels.

Ben: I wrote our implementation of the Naive Bayes algorithm, and the accompanying functions for doing spatial analysis. I implemented both neural nets, and did the hyperparameter training on both. I also designed both vector extraction algorithms, including the neural net adaptation of word2vec, which I researched. I wrote additional methods to process the data and manipulate it as one-hot numpy arrays, as well as all the methods for generating the included plots.