The Recipe Learner

Doug Safreno, Yongxing Deng

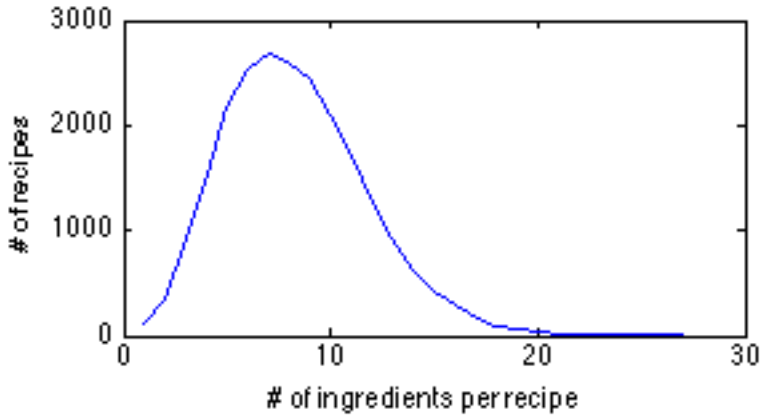
1. Introduction

Cooking is difficult, particularly when not following a specific recipe. Often times a cook wants to know how much of a common ingredient (e.g. salt) to throw into their mix of other ingredients. In particular, given n ingredients and $n-1$ amounts, the cook wants to find the nth amount. The probability of getting an exact match in a database of ingredients is low given these requirements, thus it is a good fit for a learning algorithm. We discuss several different approaches to solve this problem below.

2. The Dataset

Our dataset initially consisted of 23,101 recipes (rows in our M matrix) with 5,032 ingredients (columns in our M matrix). The value in each cell was specified as the amount in grams of that ingredient. This matrix was very sparse due to each recipe consisting of relatively few of these ingredients. The data was crawled from allrecipes.com, which states that it has over 40,000 recipes (though our crawler only found 23,101 unique recipes). We found after crawling that 75.4% of ingredients occurred less than 10 times in all recipes (making up a total of 4.56% of ingredient occurrences in recipes). Thus, we first removed all ingredients with fewer than 10 occurrences, and then removed recipes that became empty. After filtering we had 23,081 recipes and 1,238 ingredients, each of which occurs in at least 10 recipes.

Our recipes are distributed by how many ingredients they require in the chart below:



The majority of the recipes require fewer than fifteen different ingredients, verifying our claim that M is a sparse matrix. Moreover, our ten most prominent ingredients are:

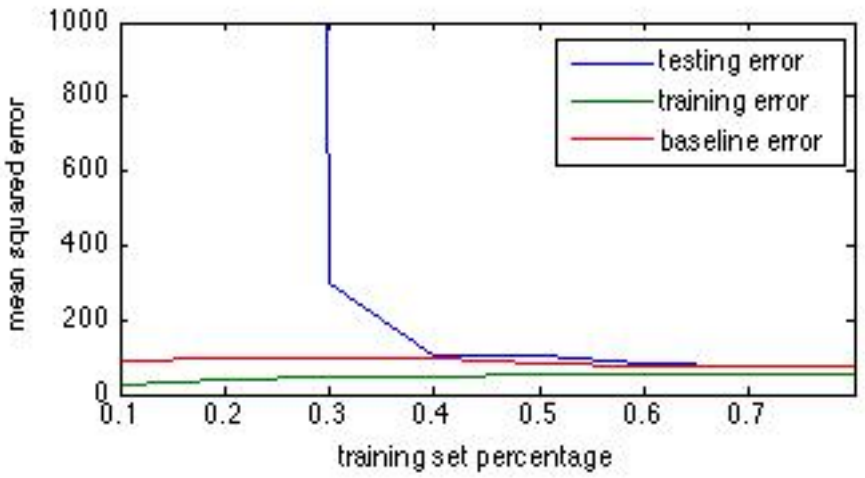
Rank	Count	Name
1	9374	salt
2	6365	butter, melted
3	6052	white sugar
4	5901	egg
5	5057	all-purpose flour
6	4783	water
7	5142	onion, halved and thickly sliced
8	4661	garlic, minced
9	3707	black pepper to taste
10	3149	milk

For the purpose of our project, we attempted to predict the amount of **butter, melted**. This is because the ingredient appeared frequently and because we suspected that recipes that contained this ingredient were more similar to one another (dessert). To accomplish this goal, we only looked at the recipes that contained the ingredient **butter, melted**.

3. Approach 1: Linear Regression

To predict the amount of butter to put in, we first attempted a linear regression. We took away the row that represented butter and used the other ingredients as features in this supervised learning setting. As a baseline, we computed the average of the target variables in the training set, and predicted this value for every testing example. We divided up all the examples into training ($X\%$) and tesing ($100 - X\%$) where X varies from 10 to 80.

We used mean squared error to measure the result:



The plot shows that the training error and testing error converge at the baseline error as the percentage of training examples goes to 1. A naive linear regression algorithm gave us a baseline result, from which we improved upon.

4. The Multimodal Problem

One reason why the naive linear regression approach did not work is the multimodal issue. To illustrate this problem, consider two ingredients: chicken and vegetables. Then, consider two dishes that consist of these ingredients: chicken caesar salad and roasted chicken with mixed veggies. The main item in the former dish is the vegetables, while the main item in the latter is chicken. This discrepancy causes a plot of chicken vs. vegetables to result in a bimodal distribution: one mode for chicken salad dishes and one for grilled chicken dishes with veggies. It is easy to see how this could be extended to many modes across many recipes. Each mode can be considered a dish. The proportions between ingredients within a dish are more consistent with one another.

5. Approach 2: Clustering and Linear Regression

To address this problem, we used a k-means algorithm to cluster all the recipes, then filtered the ingredients, and then applied linear regression to predict our target. We hoped that the clusters would correspond to dishes, and that the linear regression would then predict the correct amount for that dish.

5.1. Clustering

We first subsetted our dataset to the 6,365 recipes that have butter. We chose to subset our data in order to improve the running time of k-means as well as to remove examples that don't have any information about butter. After normalizing all the features [1], we then applied k-means to the training set to produce 40 clusters. We use 40 as the k

[1] so that mean is 0, and standard deviation is 1 for all features.

in our k-means because it seemed to produced the most subjectively accurate clustering. A few of these clusters are shown below (displayed examples are randomly chosen from the clusters):

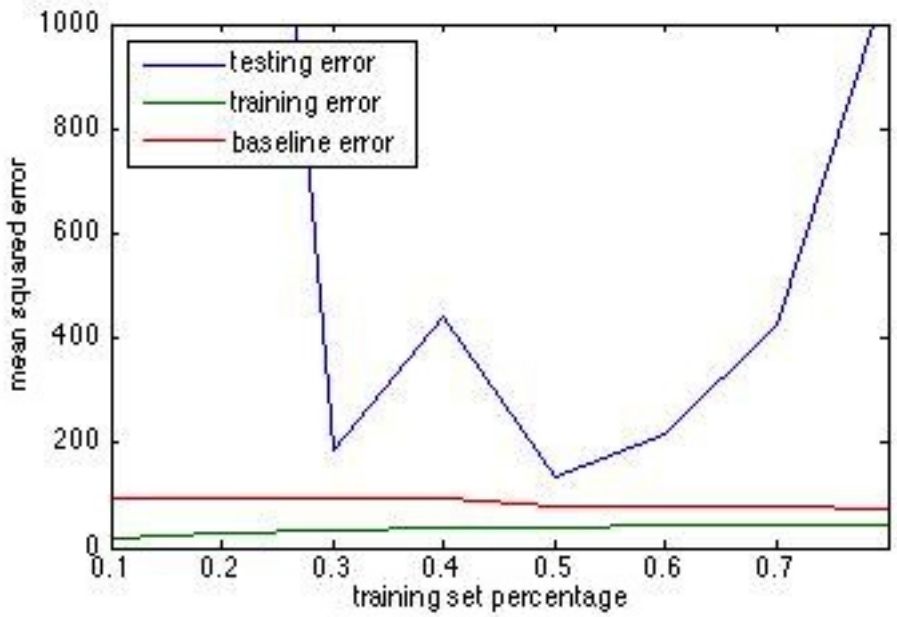
Cluster 7	Cluster 14	Cluster 23
white-chocolate-...-cookies-i	homemade-pan-rolls	...-macaroni-and-cheese
white-chocolate-...-cookies	mall-pretzels	...-cheesecake
white-chocolate-...-cookies-ii	...-bread-machine-rolls	...-seafood-lasagna
white-chocolate-blondies	...-cheese-bread	lasagna-spinach-roll-ups
secret-chocolate-fantasy-cake	clone-of-a-cinnabon	...-macaroni-and-cheese-ii
red-velvet-cake-iv	cinnamon-raisin-bread-ii	spinach-enchiladas
macadamia-...-cookies	ds-whole-wheat-challah	baked-spaghetti-3
white-chocolate-...-cookies-iii	best-rolls-ever	ginas-creamy-...-lasagna
swiss-white-chocolate-cake	pams-bierocks	taco-queso-dip
...-white-chocolate-cookies	serendipity-bread	...-red-beans-lasagna
⋮	⋮	⋮
(12 more)	(143 more)	(99 more)

5.2. Directly running linear regression in clusters results in overfitting

Our first algorithm with linear regression is as follows:

```
p_whole = linear_regression(whole_training_set)
for i = 1, 2, ..., K:
  if size(cluster_i) >= MIN_CLUSTER_SIZE:
    p_i = linear_regression(cluster_i)
    use p_i to predict cluster_i
  else:
    use p_whole to predict cluster_i
```

Here, MIN_CLUSTER_SIZE is a constant thresold (16) that we determine if a cluster is meaningful enough to run linear regression on its own. However, we found that the training error was far below the testing error. This is shown in the plot below.

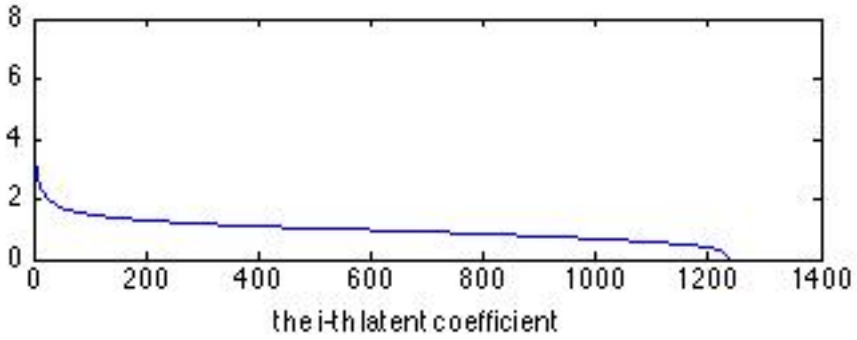


As the number of training examples in each cluster increases, the testing error actually goes up. This means that the linear regressions in clusters do not predict the amount well. Clusters are effectively representing different types of recipes, so why isn't the algorithm effective? **By using clustering, we are reducing the number of training examples, but we are not reducing the number of features. The ratio of number of features to number of examples increases significantly. The combination of the two results in overfitting.** As a result, we need to find a way to reduce the number of features.

5.3. Some attempts at dimensionality reduction...

Attempt 1. Independent component analysis - failed. If there are two ingredients that are linearly correlated to each other, then independent component analysis would help us remove one of them. Unfortunately the dataset does not have feature pairs like that. We suspect this is because columns like "Butter, salted" and "Butter, unsalted", which would initially seem linearly dependent, are in fact most often mutually exclusive, and thus very independent.

Attempt 2. Principle component analysis - failed. If all the examples actually lie in a subspace of R^n, then principle component analysis could reduce the dimension. After standardizing the matrix, we ran principle component analysis:



Latents, or the eigenvalues of the co-variance matrix, are proportional to the variance [2]. Since the latents are close to each other, if we only use the first few principle components, then we lose a very high proportion of the variance of the dataset. As a result, we cannot use principle comoponent analysis to reduce dimensionality.

Attempt 3. Cluster-specific dimensionality reduction - **success!** Since each cluster represents a different dish, the principal features in each are different. As a result, **dimensionality reduction should be done within each cluster.** With # of examples in a cluster $\ll$ # of ingredients, we suspect that the simplest algorithm will work the best. For a cluster i, we only consider C_i ingredients that have the highest variances within that cluster.

But how do we decide C_i? Since our original reason for dimensionality reduction is to prevent overfitting, the larger a cluster is, the more ingredients we should allow our algorithm to consider. Thus, for a cluster with T_i recipes, let $C_i = T_i/D$ ingredients, where D is a pre-determined constant.

5.4. Putting it together

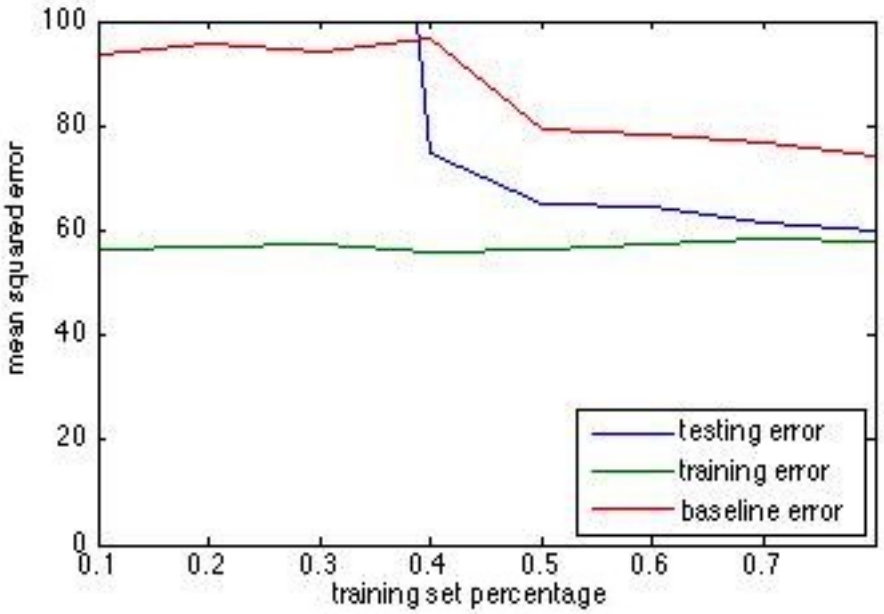
This is our final algorithm, combining the ideas of clustering, dimensionality reduction and linear regression:

```
p_whole = linear_regression(whole_training_set)
for i = 1, 2, ..., K:
  if size(cluster_i) >= MIN_CLUSTER_SIZE:
    c_i = size(cluster_i) / 8
```

[2] see http://www.mathworks.com/help/stats/princomp.html

```
    reduced_features_i = c_i features with highest variance in cluster_i
    p_i = linear_regression(reduced_features_i)
    use p_i to predict cluster_i
  else:
    use p_whole to predict cluster_i
```

6. Results and conclusions



Our results show that our algorithm performs noticeably better than the baseline, and thus also better than our linear regression. It also removes the overtraining problem that we experienced before reducing the columns, as is evidenced by the convergence of the training and testing errors.