Rahul Agarwal
Daniel Jachowski
Shengxi David Wu

RoboChef: Automatic Recipe Generation

"byte-sized recipes"

Introduction:

Our algorithm intends to synthesize new recipes by identifying promising mixtures of ingredients and classifying by recipe type. Our dataset comes from the International Conference on Case-Based Reasoning's (ICCBC) Computer Cooking Contest[1]. The data consists of a smaller training set of about 900 recipes in XML format to allow for relatively easy parsing of individual recipes, as well as a larger extension set for us to use once our algorithm has reached a more scalable level.

Parsing:

A significant challenge was parsing the semi-structured recipe data. We used statistical NLP techniques to help with the parsing. The recipe data was stored in an XML format, where each recipe has its own tag as shown below:

<RECIPE>
<TI>Ginger And Peach Chicken</TI>
<CAT>Entrée</CAT>
<IN>4 x Chicken Breast halves (12 oz total), boned, skinless</IN>
<IN>8 oz Can Peach slices, lite syrup</IN>
<IN>1 ts Cornstarch</IN>
<IN>1/2 ts Grated Ginger root</IN>
<IN>1/4 ts Salt</IN>
<IN>1/2 c Sliced Water Chestnuts, drained</IN>
<PR><PS>Spray a large skillet with non stick spray</PS><PS> Preheat skillet over medium heat</PS><PS> Add chicken</PS><PS> Cook over medium heat for 8 - 10 minutes or till tender and no longer pink; turn to brown evenly</PS><PS> Remove from skillet; keep warm</PS><PS> Meanwhile, drain peaches, reserving juice</PS>
</RECIPE>

The first parsing task is to extract the main ingredient word from each ingredient tag. For example, we want to extract the word "chicken" from the line "4 x Chicken Breast halves (12 oz total), boned, skinless". First, numbers and units were removed from the line in a preprocessing step. Then we counted how often each word in the ingredient line occurred in the preparation steps, and took the most common word as the main ingredient. This worked under the assumption that the ingredient word (i.e. "chicken") should be the most important word in the line, and hence should be mentioned the most frequently.

However, it was often the case where no word in the ingredient was mentioned in the directions. For example, if the ingredients listed "beef", there would be situations where the

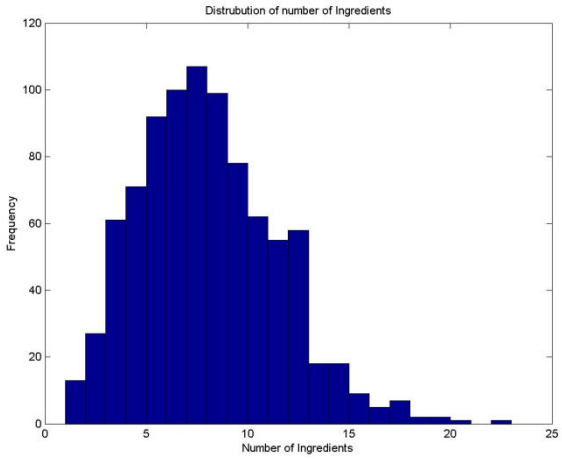
[1] http://www.wi2.uni-trier.de/eccbr08/index.php?task=ccc&act=5

directions would always refer to "meat", so "beef" would have 0 occurrences. To help alleviate this problem, we took a second pass through the data, where we counted how often each word in the line occurred as an ingredient in other recipes. The assumption for this model was that many recipes share common ingredients, and that common ingredients, such as "beef" are the most likely to be replaced in natural language directions. This technique yielded significantly cleaner lists of ingredients.

To parse the units and the amounts, we assumed that the first number in the line was the amount, and the following word was the unit. This assumption worked reasonably well, although it failed on sentences such as "A medium bag of rice", where units are not explicitly specified numerically.

Selecting Ingredients:

We assume that the number of ingredients in a recipe, N, follows lognormal distribution that can be fit through the maximum likelihood estimates. From this lognormal distribution we sample $\widetilde{N}$ to serve as the target number of ingredients.



Let the vector of m unique ingredients be given as $\boldsymbol{I} = (I_1, I_2, \dots I_m)^T$. Given an initial input seed of favorite ingredients, we then generate a set of ingredients that complement that seed, with an expected number of $\widetilde{N}$ ingredients. The seed will be a binary vector, with 1's at the index of where ingredients are given. To calculate the probability of selecting a new ingredient I_j given a seed with the n ingredients $I_{s_1}, I_{s_2}, \dots I_{s_n}$, we can use Bayes' Rule:

$$P\left(I_j \middle| I_{s_1} I_{s_2}, \dots, I_{s_n}\right) = \frac{P\left(I_j, I_{s_1} I_{s_2}, \dots, I_{s_n}\right)}{P\left(I_{s_1}, I_{s_2}, \dots, I_{s_n}\right)}$$

However, as the number of ingredients in the seed grows, we find that oftentimes there is no recipe that has all of these ingredients. But we still want to assign a score that provides some information on how well a new ingredient might complement the existing ones. We propose the following modification:

Suppose we have n ingredients. Then for a "fun factor" of k the probability of adding a new ingredient j is the union of all $\binom{s}{k}$ subsets.

For example, for $k = 2$:

$$\rho_j = P\left(I_j \middle| I_{s_1} I_{s_2}, \dots, I_{s_n}\right) = P\left(I_j \middle| I_{s_1} I_{s_2}\right) \cup P\left(I_j \middle| I_{s_1}, I_{s_3}\right) \cup \dots \cup P\left(I_j \middle| I_{s_2}, I_{s_3}\right) \dots \cup P\left(I_j \middle| I_{s_{n-1}}, I_{s_n}\right)$$
$$= 1 - \left(1 - P\left(I_j \middle| I_{s_1} I_{s_2}\right)\right)\left(1 - P\left(I_j \middle| I_{s_1}, I_{s_3}\right)\right) \dots \left(1 - P\left(I_j \middle| I_{s_2}, I_{s_3}\right)\right) \dots \left(1 - P\left(I_j \middle| I_{s_{n-1}}, I_{s_n}\right)\right)$$

where we assume that the complement of each subset is independent.

The probability vector $\boldsymbol{\rho}$ is then scaled by $\beta = \frac{\widetilde{N}}{\sum \boldsymbol{\rho} - \sum \boldsymbol{I_s}}$ such that we get $\widetilde{N}$ in expectation, after using a random number generator to decide whether or not the ingredient is ultimately included, with probability $\beta\rho_j$. Thus, a lower "fun factor" increases the likelihood a less correlated ingredient is picked and allows the algorithm to find new, plausible, ingredient clusters.

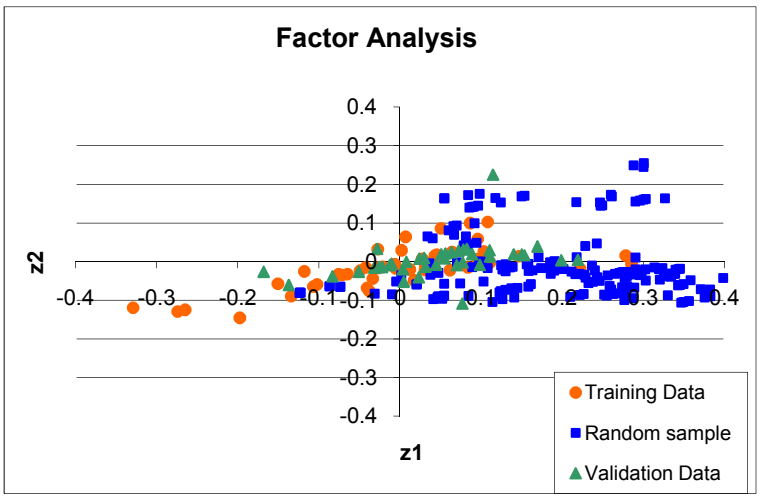
Predicting Ingredient Amounts:

Another important component to recipe generation is determining how much of each ingredient is needed for a particular recipe. Our approach to this problem was to create a model of what quantities of various ingredients go together, and then use this model to estimate ingredient amounts for new ingredient sets. Although we considered using a Mixture of Gaussians and Principal Component Analysis to create this model for relative ingredient amounts used in recipes, we ultimately selected Factor Analysis. This decision was made due to the high dimensionality of the sample space (the number of ingredients in the training set) being approximately equal to the size of the data set. Without significantly more training samples, neither of the other two options would be very reliable. Using Factor Analysis, we were able to reduce our n dimensional sample space down to k dimensional space using the transformation $x = \mu + \Lambda z + \varepsilon$, where x is the feature vector, μ is the mean of the feature vectors, z is the factor vector distributed $N(0, I)$, ε is an error term distributed $N(0, \Psi)$, and Λ maps from R^k to R^n.

In training the Factor Analysis model, the feature vectors were n dimensional, where n was the number of unique ingredients in our sample space, and each vector represented the ingredients of a recipe. Within each vector, i of the entries corresponded to the i ingredient amounts used in the recipe. All other $n - i$ entries were initialized to 0. Some difficulties were encountered here in that it was sometimes challenging to extract an amount for an ingredient (if an ingredient was just "salt", or "bag of rice"), and units were not always standardized. However, as can be seen from the figure below, when fitting a hold-out set to our model, ingredient values were mapped similarly to the distribution we trained, suggesting that despite these parsing issues, the resulting model is valid.

One of the major problems encountered in performing Factor Analysis was not that real recipe information did not map well into the Factor space, but rather that non-recipe data also mapped well onto the Factor space. With too few factors, nearly all feature vectors were mapped close to the mean, which would make it difficult to predict which recipe amount combinations were desirable. The limiting factor in the number of features we had was that for larger k values, we ran into underflow issues in calculating the mapping function.

Therefore, we used as many factors as possible that would reliably produce a mapping function from R^n to R^k. We determined that these higher dimensional transformations were more effective by mapping randomly seeded ingredient vectors of 1 to 10 ingredients onto the Factor space, where the random seeds were within realistic amount ranges. We expected these points to map near the mean of the trained data, because they have the right number of ingredients and reasonable values for the amounts, but to not be strongly correlated with the actual ingredients, because random seeding will likely yield undesirable mixes. An example of this is shown in the graph below.



After performing factor analysis to find a low dimensional subspace of plausible amount vectors, coordinate ascent was performed to select the amounts of each ingredient selected by maximizing the likelihood of each ingredient amount. For each ingredient in the recipe, the amount was initialized with a random value, and coordinate ascent was run to maximize the probability of the projected vector. Coordinate ascent was used instead of gradient ascent in order to enforce the constraint that the amounts for ingredients not selected for the recipe should remain zero.

Assume there is an affine mapping from $x^{(i)}$ to $z^{(i)}$: $\Lambda z^{(i)} = (x^{(i)} - \mu)$. Because Λ is not square, we use the normal equations to get a least squares fit, giving us:

$$z^{(i)} = (\Lambda^T \Lambda)^{-1} \Lambda^T \left(x^{(i)} - \mu\right) = A(x^{(i)} - \mu)$$

Based on the factor analysis model, z is a low dimensional standard Gaussian, so:

$$f\left(A\left(x^{(i)} - \mu\right)\right) = f\left(z^{(i)}\right) = P\left(z^{(i)}\right) = \frac{1}{(2\pi)^{k/2}} \exp\left(-\frac{1}{2} {z^{(i)}}^T z^{(i)}\right)$$

Now we want to maximize $f\left(z^{(i)}\right)$ with respect to the amounts of our chosen ingredients $x^{(i)}$. Because there is a linear relationship between $z^{(i)}$ and $(x^{(i)} - \mu)$, we can derive the following update rule:

$$x_j^{(i)} := x_j^{(i)} + \alpha \frac{\partial f\left(z^{(i)}\right)}{\partial x_j^{(i)}}$$

where we define:

$$\frac{\partial f\left(z^{(i)}\right)}{\partial x_j^{(i)}} = A_{\cdot,j}^T \nabla_z f\left(z^{(i)}\right) = \left(-\frac{1}{(2\pi)^{k/2}} \exp\left(-\frac{1}{2} {z^{(i)}}^T z^{(i)}\right)\right) A_{\cdot,j}^T z$$

Here, $A_{\cdot,j}^T$ is the j_{th} column of matrix A, and α is our learning rate. This update was run in order on each coordinate for which there was a valid ingredient, until convergence. For simplicity, we assumed that the unit associated with each ingredient was the most frequently occurring unit of that ingredient. Due to numerical instability issues, we had trouble getting the amount vector to converge, which would be the focus of a future experiment.

Classifying Recipes:

Each recipe in the training set was hand tagged as an appetizer, entrée, dessert, or other. Then, using the ingredients as features, we generalized the binary classification Naïve Bayes model into one that classifies for multiple classes. The user could input what type of category they wish to cook, and our algorithm would keep generating sets of ingredients until it finds one that matches their category.

In short:

$$Y \in \{\text{Appetizer, Entree, Dessert, Other}\}$$
$$x_j^{(i)} = \mathbf{1}\{\text{Ingredient } j \text{ is in recipe } x^{(i)}\}$$

To form a prediction, we find

$$\arg\max_c P(y = c|x) = \frac{P(x|y = c)P(y = c)}{\sum_{c'} P(x|y = c')P(y = c')}$$

with the Naïve Bayes assumption of

$$P(x|y = c) = \prod_{i=1}^{n} P(x_i|y = c)$$

and the maximum likelihood estimates are given by:

$$\phi_{j|y=c} = P(x_j|y=c) = \frac{\sum_{i=1}^{m} \mathbf{1}\left\{x_j^{(i)} = 1 \wedge y^{(i)} = c\right\} + 1}{\sum_{i=1}^{m} \mathbf{1}\{y^{(i)} = c\} + |Y|}$$

$$\phi_c = P(y=c) = \frac{\sum_{i=1}^{m} \mathbf{1}\left\{y^{(i)} = c\right\}}{m}$$

Further Studies:

In order to extend this project, we would like to obtain more data to get a denser sampling of the "Recipe Space". This would allow us to fit a more accurate distribution of recipe combinations, yielding more plausible combinations and amounts of ingredients. Furthermore, having a more densely sampled space would allow us to use more search techniques over the distributions, such as genetic algorithms or simulated annealing. It would also allow us to use other techniques to create a model for good recipe amount combinations, such as Mixture of Gaussians and Principal Component Analysis.

The natural language processing problems inherent in extracting useful information from recipes is another area in which this project could be expanded on. Having more accurate techniques for extracting ingredient names from recipe lists could help reduce errors and redundancy in our list of ingredients.

Although we were ultimately unable to tackle the problem of synthesizing preparation instructions, more intelligent parsing techniques would also help achieve this goal. Lack of reliable parsing techniques is a huge obstacle for this goal, as it requires understanding where verbs are in sentences and which objects they are operating on. Another major obstacle in synthesizing instructions is defining a structure for ingredient steps. The structure we envisioned consisted of each instruction step equating to a recipe state, and one could move to different recipe states through actions such as “mix”, “stir”, or “bake.” If the data could be parsed sufficiently well, we could try to fit some sort of Markov Model, such as a Hidden Markov Model, to the data in order to learn transitions between different cooking actions and ingredients. Combining this step generation with our existing ingredient clustering method would then be able to produce complete recipes.