Predicting the Structure of Cooking Recipes

Jermsak Jermsurawong and **Nizar Habash**
Computational Approaches to Modeling Language Lab
Computer Science
New York University Abu Dhabi
United Arab Emirates
{jermsak.jermsurawong,nizar.habash}@nyu.edu

Abstract

Cooking recipes exist in abundance; but due to their unstructured text format, they are hard to study quantitatively beyond treating them as simple bags of words. In this paper, we propose an ingredient-instruction dependency tree data structure to represent recipes. The proposed representation allows for more refined comparison of recipes and recipe-parts, and is a step towards semantic representation of recipes. Furthermore, we build a parser that maps recipes into the proposed representation. The parser's edge prediction accuracy of 93.5% improves over a strong baseline of 85.7% (54.5% error reduction).

1 Introduction

Cooking recipes are a specific genre of how-to instructions which have been gaining interest in recent years as they may allow us to discover insights into culinary and cultural preferences. Most of the work studying recipes relies on simple ingredient bag-of-word representations. While such representations may suffice for many purposes, they fail to capture much of the recipes' internal structure. A smaller number of efforts focus on semantic representations of cooking recipes with an eye toward more complex and deep understanding. Natural language processing (NLP) techniques have been used to interpret cooking instructions; however, the results have not been as successful as other language genres due to the unique aspects of cooking recipes.

This paper makes two contributions. First, we propose an ingredient-instruction dependency tree representation of recipe structure. This representation abstracts textual recipes more expressively than bag-of-word representations; and it allows for more nuanced comparisons of recipes. Second, we present a cooking recipe parser that maps text recipes into our proposed ingredient-instruction tree structures. The overall accuracy of predicting edges of our ingredient-instruction trees is 93.5%, beating a strong baseline of 85.7%, and achieving a relative error reduction of 54.5%.

We present next some related work (Section 2) followed by a discussion of the structure of recipes and our representation (Section 3). Section 4 details the recipe parser design, implementation and evaluation.

2 Related Work

There have been many efforts on the processing of cooking recipes using models that range from bags of words to complex semantic representations.

Among the approaches to studying recipes as ingredient bags of words, Ahn et al. (2011) constructed a data-driven flavor network relating ingredients together. Jain et al. (2015) adopted Ahn et al. (2011)'s framework to further analyze culinary practices of specific cultures. Nedovic (2013) examined underlying ingredient groupings from their recipe co-occurrences, using topic modeling techniques (latent Dirichlet allocation), and further improvised novel ingredient combinations using deep belief networks.

Among the structured representation approaches, Tasse and Smith (2008) proposed MILK (Minimal Instruction Language for the Kitchen), a formal language to describe actions required in directive cooking instructions. They used MILK in developing CURD (Carnegie Mellon University Recipe Database), a corpus of manually annotated recipes. Mori et al. (2014) also manually annotated the procedural flow of Japanese cooking recipes using directed acyclic graphs (DAGs) where graph nodes correspond to food ingredients, cooking instruments, and actions. Tasse and Smith (2008) had limited success in parsing into MILK; and Mori et al. (2014) did not report on parsing experiments.

Other studies explored different machine learning and NLP techniques to processing recipes.

Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, pages 781–786,
Lisbon, Portugal, 17-21 September 2015. ©2015 Association for Computational Linguistics.

	Text Recipe
	ingredients
0	6 (2 inch thick) slices French bread
1	1/4 cup ricotta cheese
2	1/4 cup cottage cheese, whipped
3	2 tablespoons lowfat cream cheese
4	2 teaspoons white sugar
5	2 teaspoons vanilla extract
6	3 cups egg substitute
7	1/4 cup evaporated milk
	instructions
0	Cut a pocket in each slice of bread.
1	Open carefully
2	In a large bowl, combine the ricotta, cottage cheese and cream cheese.
3	Add the sugar and flavoring extract and beat until smooth.
4	Spread the mixture evenly into each bread pocket.
5	Beat together the egg substitutes and milk.
6	Dip the slices of bread in the egg mixture.
7	Heat a nonstick pan over medium-high heat.
8	Coat with cooking spray.
9	Cook the toast on each side for about 3 to 4 minutes per side until golden brown.

SIMMR

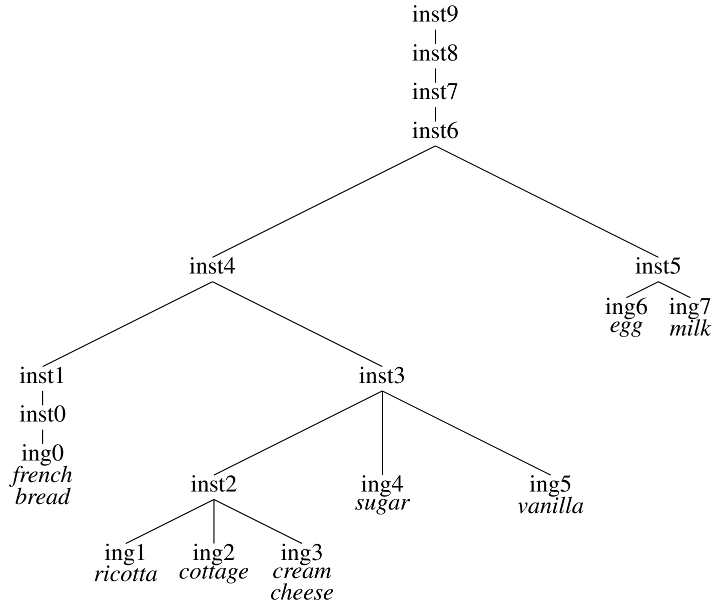

Figure 1: Example of a text recipe for *Surprise-inside French Toast* and its SIMMR representation. **ing**<*index*> and **inst**<*index*> refer to specific ingredients and instructions, respectively.

Mori et al. (2012) applied word segmentation, named entity recognition, and syntactic analysis to extract predicate-argument structures from Japanese recipe instructions as part of an effort to develop complete recipe flow representations. Malmaud et al. (2014) proposed a Markov Decision Process, in which the context of ingredients and tools is propagated along the temporal order of cooking instructions.

Most recently, Abend et al. (2015) proposed an *edge-factored* model to determine the likely temporal order of events based solely on the identity of their predicates and arguments. They demonstrated their approach on recipe text, under the simplifying assumption that such text is also temporally ordered.

In this paper, we present an ingredient-instruction dependency tree representation of recipe structure, which we call SIMMR (Simplified Ingredient Merging Map in Recipes). The SIMMR representation captures the high-level flow of ingredients but without modeling the semantics in each individual instruction unlike other efforts (Tasse and Smith, 2008; Mori et al., 2012; Mori et al., 2014). We create a corpus in our representation by converting the recipes in the CURD corpus (Tasse and Smith, 2008) from MILK to SIMMR. We also develop a parser to generate SIMMR trees from input recipes.

3 Recipe Representation

3.1 Text Recipes

The prototypical text recipe consists of two parts: an ingredient list that declares the food items to process, and a set of instructions that mostly describe the transformations of the ingredients or the actions using the kitchen tools. The instructions relocate, process, combine, and separate ingredients, as well as heat or cool utensils in the recipes. For the most part, the output produced from one instruction feeds as input into another instruction. The list of ingredients can be generalized as special *fetch* instructions whose output feeds as input to one of the cooking instructions.

3.2 SIMMR

Our proposed representation is SIMMR: Simplified Ingredient Merging Map in Recipes. SIMMR represents a recipe as a dependency tree whose leaves (terminal nodes) are the recipe ingredients, and whose internal nodes are the recipe instructions. Figure 1 exemplifies the SIMMR tree of a recipe for *Surprise-inside French Toast*. Indices for all ingredients and instructions are provided here to illustrate mapping between the different parts of the text recipe and its SIMMR tree. The subtree headed by *inst2* indicates that ingredients #1, #2 and #3 (three cheeses) are inputs to instruction #2, whose output is then an input to instruction #3, together with ingredients #4 and #5 (sugar and vanilla). The SIMMR tree provides additional insights into the structure of recipes, suggesting in the case of this example, that multiple actions can take place in different orders without changing the recipe so long as the order of combinations is not changed. Some instructions simply transforms their input to produce their output, e.g. instructions #1, #7, #8 and #9.

3.3 From MILK to SIMMR

We use MILK commands from the CMU CURD database (Tasse and Smith, 2008) to construct a database of SIMMR trees. The MILK language is a lot more expressive that SIMMR. For example, instruction #2 in Figure 1 translates into create_tool(t0, "large bowl"); combine(ing1, ing2, ing3, ing9, "cheeses", ""); and put(ing9, t0). These details of ingredient processing are abstracted away in SIMMR. To accomplish SIMMR instruction and ingredient linking, we process MILK instructions in order, tracing with MILK id numbers when each ingredient or its transformed or combined form at one instruction node is called for by a subsequent instruction node. The MILK intermediate names for instruction outputs (e.g., "cheeses" above) are not adopted in SIMMR. Additionally, food items that appear in instructions but are not part of the recipe ingredient list text are not included in SIMMR although MILK assigns ingredient ids to them. For example, instruction #8 mentions *cooking spray*, which is not on the ingredient list. As a result *inst8* looks like it has no ingredients coming into it other than the output of *inst7*.

We assume in SIMMR that at most one output is produced from each instruction. One MILK command, *separate*, violates this assumption. For example, the instruction *drain off the fat, and place the mixture into a slow cooker* is MILK-tagged with a *separate* command to dispose fat, and retain the mixture. We ignore the *separate* command which occurs in 3% of all recipes (and 1.3% of the time involves disposing of a separated ingredient, such as *draining off fat*).

Instructions that do not interact with food, such as those calling for preheating the oven or lining the baking sheets, are considered to take the ingredient mixture of the cooking instruction immediately prior and produce the *same* output.

4 Recipe Parser

In this section, we present our SIMMR tree recipe parser. The input and output of the parser correspond to the left hand and right hand sides of Figure 1. We split the parsing process into two phases: first we link ingredients to instructions where the ingredients are first used; then we link instructions to other instructions, where the target instruction uses the source instruction's output. We present the different challenges and solutions employed in each phase below and present evaluation results.

4.1 Experimental Setup

4.1.1 Data

We use a total of 260 recipes downloaded from CMU Recipe Database. MILK tags are used to construct the SIMMR trees as mentioned above. The dataset is randomly split (along recipe boundaries) into training, development, and test sets with ratios of 50%, 20%, and 30%, respectively. The development set is used for tuning parameters. We report here on the test set only. We preprocess the data using standard NLP packages to tokenize, stem, and POS tag the words (De Marneffe et al., 2006; Bird et al., 2009; De Smedt and Daelemans, 2012). We further remove stop words and quantity measures.[1]

4.1.2 Metrics

The evaluation metric is the accuracy of predicting edges in the SIMMR tree. This is comparable to the *attachment score* in dependency parsing. The overall accuracy of the task is computed at the edge level (counting all edges in the data set), and at the recipe level (average accuracy over all recipes).

4.2 Ingredient-Instruction Linking

4.2.1 Challenges

There are several characteristics of recipe text that do not reveal explicit linking of ingredients to instructions that first use them. The ingredients are sometimes referred to by their qualifiers or ontological classes. For example, the ingredient *1 (15 oz) can sliced peaches, drained* may be referred to in the instruction as *canned fruit*. In addition,

[1] Since we start from the MILK representations, the set of ingredients and instructions are clearly identified in each recipe. However, we do not expect this to be the case when starting from raw text recipes, which need to be processed to segment the different ingredients and instructions. We do not attempt this step in this paper and leave it as future work.

	Edge-level Accuracy	Recipe-level Accuracy
Baseline	84.3	84.8
SVMrank	**95.3**	**95.8**

Table 1: Performance of different learning models for ingredient-instruction linking

a number of ingredients are sometimes referred to as a group entity without specific mentions, e.g., *Add the remaining ingredients*, or *mix together the dry ingredients*.

4.2.2 Baseline

For each instruction, in order, we compute all the maximum *stemmed* n-gram chunk matches against all ingredients. For each matching token chunk of the instruction, if it matches only one ingredient, we link the ingredient to the instruction and mark the ingredient as used. Otherwise, we compute the Levenshtein distance between the *unstemmed* surface text of both the ingredient candidate matches and the instruction token chunk, and take the highest matching ingredient. Once an ingredient is used, it is no longer available for linking with subsequent instructions.

4.2.3 Features and Linking

For each ingredient-instruction edge, we provide 1,136 features to classify whether the edge exists or not. The following are some of the most important features we use:

- The baseline decision for the ingredient-instruction edge.
- The number of distinct unigram matches.
- The largest match size.
- The sum of the relative frequency (in the list of instructions) of every word in the largest match (henceforth SRFM).
- The degree of similarity between SRFM and the instruction relative position (InstRP) in the instruction list. The intuition for this feature is that if an ingredient is mentioned more often in the recipe, its first mention is likely to be in an earlier instruction. This feature is computed as $(1 - |SRFM - InstRP|)$.
- The degree of similarity between the ingredient relative position (IngRP) in the ingredient list and the InstRP. This feature captures an observation that earlier listed ingredients are used by earlier instructions. We compute the feature as $(1 - |IngRP - InstRP|)$.
- To model the first word of each instruction (i.e., the directive verb such as *heat* or *mix*), we use a binary term vector whose vocabulary is constructed from all the instruction first words in the training data.
- We model the maximum ingredient-instruction word match using a binary term vector whose vocabulary covers *non-directive recipe words* (NDRW). We construct the NDRW vocabulary by taking the union of the set of words from the compiled food item list of Ahn et al. (2011) and the non-first words of the instructions in the training data.
- If there is a match, we model its surrounding words using a similar binary term vector (NDRW vocabulary). If there is no match, the vector will be empty.
- We use a binary term vector (NDRW vocabulary also) to model the non-first words in the instruction.

We train using Linear SVMrank (Joachims, 2006).[2] The test set results are shown in Table 1. Ranking edges using SVMrank and then picking the best one significantly outperforms the baseline.

4.2.4 Linking Errors

Among the linking errors, three patterns appear. First, stemming reduces the specificity of some of the terms, e.g., one prediction links *baking powder* to an action *bake for 30 minutes*. Second, our approach does not handle negated mentions, e.g. the instruction *mix together the dry ingredients,* **except** *the candies* is incorrectly linked to ingredient *candy*. Finally, the ingredients *flour* and *butter* are specifically part of many erroneous links because they are often used in small quantities to facilitate the cooking process such as board flouring and pan greasing. MILK does not always account for this trivial use of these ingredients and neither does SIMMR.

4.3 Instruction-Instruction Linking

4.3.1 Challenges

Although cooking instructions are written with an implied temporal order, they are not linked in a linear chain. Rather, the instructions describe different cooking *stages*, where the output of apply-

[2] We also experimented Linear SVM and Gaussian Kernel SVM (Pedregosa et al., 2011). We used the distance from the hyperplane to select the best edge among the set of edges connecting an ingredient. However, these techniques underperformed compared to SVMrank.

ing a number of instructions waits until other instructions are finished to be used again, e.g., the output of *instruction #1* in Figure 1 waits until instructions #2 and #3 are done. Sometimes stage switching is explicitly stated as in *Set aside the flour mixture, and combine eggs and oil together*; however, this is not the common case. Furthermore, the waiting output of an earlier stage may be referred to collectively or using the main ingredient which makes linking harder, e.g., referring to the output of an instruction combining *chicken, salt and pepper* as *chicken*.

4.3.2 Baseline

We link the instructions in a linear chain. In the training set, 89% of the instruction-instruction links are to the immediate neighbor.

4.3.3 Features and Linking

For each possible instruction-instruction edge, we provide 1,573 features to classify if the edge exists or not. The features group into three categories.

First are words that suggest cooking stage switching, e.g., ingredient mixture words such as *mixture*, *dough*, and *batter*, or the simple mention of new containers and utensils. They are represented as binary features indicating whether the words appear in either or both instructions along the considered edge, as well as in immediate neighboring instructions. Logic operations (and, or) are further applied to all pairs of these features. Another feature in this group is a binary verb conjugation feature that marks the presence of a past participle verb in the target instruction and a non-past-participle form of the same verb in the source instruction. The intuition here is that an instruction that asks to *chop* an ingredient would be followed later by an instruction containing the word *chopped* when referring to the ingredient.

The second group consists of features that encode whether the n previous or future instructions has m linked ingredients, where n ranges from 1 to 3 and m ranges from 0 to 4.

Finally, the third group deals with term vectors describing the source and target instructions, as well as, the instructions' first words (the directives). The term vector vocabularies used are the same as those discussed above in the ingredient-instruction linking section.

We additionally consider decoupling the features for the case of immediate neighboring instructions from the *further* apart instructions. In the decoupled mode, we effectively double the number of features for each edge with *nil*s used to fill in the gaps. This allows us to learn different models for adjacent and apart instructions. The results are in Table 2 and show that decoupling features in SVMrank is our best setting.

	Edge-level Accuracy	Recipe-level Accuracy
Baseline	87.6	89.5
SVMrank	90.5	92.0
SVMrank - decoupled	**91.3**	**92.4**

Table 2: Performance of different learning models for *instruction*-instruction linking

Examining the weights learned for adjacent and long-distance edge features gives some interesting insights. One example is that the verb conjugation feature has a higher weight for long-distance edges compared to adjacent edges. This is understandable as most adjacent pairs of instructions would not exhibit this feature to express cooking progress: it is redundant to state, *cook the pasta*, and immediately refer to that pasta as the *cooked pasta*.

4.3.4 Linking Errors

As the distance between linked instructions increases, the likelihood of error also increases: while long-distance (i.e., non-adjacent) links constitute 12.3% of the reference, they are 91.7% of the errors. We expect that more training examples of long-distance linking can help address this issue.

4.4 Overall Accuracy

The overall edge-level accuracy of constructing the **full** SIMMR tree is 93.5%, outperforming a baseline of 85.7%, and achieving error reduction of 54.5%.

5 Conclusions and Future Work

We proposed a new ingredient-instruction dependency tree representation to capture the internal structure of cooking recipes, and built a parser for it. Our overall parsing accuracy is 93.5%, outperforming a strong baseline of 85.7%.

We will make our SIMMR database, and our SIMMR parser publicly available. Further, we plan to build on SIMMR to get closer to the MILK representation. We also plan on parsing a large corpus of text recipes to provide structural features on a large scale that will allow us to discover new patterns of similarity across and within cuisines, as well as generate new recipes.

References

Omri Abend, Shay B Cohen, and Mark Steedman. 2015. Lexical event ordering with an edge-factored model. In *Proceedings of NAACL*.

Yong-Yeol Ahn, Sebastian E Ahnert, James P Bagrow, and Albert-László Barabási. 2011. Flavor network and the principles of food pairing. *Scientific reports*, 1.

Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural language processing with Python*. "O'Reilly Media, Inc.".

Marie-Catherine De Marneffe, Bill MacCartney, Christopher D Manning, et al. 2006. Generating typed dependency parses from phrase structure parses. In *Proceedings of LREC*, volume 6, pages 449–454.

Tom De Smedt and Walter Daelemans. 2012. Pattern for python. *The Journal of Machine Learning Research*, 13(1):2063–2067.

Anupam Jain, Ganesh Bagler, et al. 2015. Spices form the basis of food pairing in indian cuisine. *arXiv preprint arXiv:1502.03815*.

Thorsten Joachims. 2006. Training linear svms in linear time. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 217–226. ACM.

Jon Malmaud, Earl J Wagner, Nancy Chang, and Kevin Murphy. 2014. Cooking with semantics. *ACL 2014*, page 33.

Shinsuke Mori, Tetsuro Sasada, Yoko Yamakata, and Koichiro Yoshino. 2012. A machine learning approach to recipe text processing. In *Proc. of the 1st Cooking with Computer Workshop*, pages 29–34.

Shinsuke Mori, Hirokuni Maeta, Yoko Yamakata, and Tetsuro Sasada. 2014. Flow graph corpus from recipe texts. In *Proceedings of the Nineth International Conference on Language Resources and Evaluation*, pages 2370–2377.

V Nedovic. 2013. Learning recipe ingredient space using generative probabilistic models. In *Proceedings of Cooking with Computers Workshop (CwC)*, volume 1, pages 13–18.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Dan Tasse and Noah A Smith. 2008. Sour cream: Toward semantic processing of recipes. Technical report, Technical Report CMU-LTI-08-005, Carnegie Mellon University, Pittsburgh, PA.