Food Recipe Processing & Entity Recognition

Kevin Pang, Kathryn Hamilton
{yp2182, kathrynhamilton}@berkeley.edu
GitHub Repository, Presentation Slides

Abstract

We are interested in converting cooking recipes into directed acyclic graphs (DAGs). We break the problem into two steps: named entity recognition and relationship extraction. We present a model to solve the first problem and propose a solution approach to the second. We apply a combination of known algorithms and best practices from recent papers to a dataset that has not been used in this area.

1 Introduction

Online cooking recipes are plentiful, but remain largely unformatted with wide variation. Often, the same action can be described in many different ways, and the steps listed in a recipe are non-atomic and lengthy. Semantic parsing poses additional challenges, as cooking instructions are not always complete sentences and feature implicit arguments not easily understood by machines.

By creating a model that can parse text-base recipes into DAGs, we can provide a more visually descriptive, structured, and informative interpretation of existing recipes. In addition, DAGs are well suited for computer use, which opens up opportunities for automation in cooking. An example cooking DAG is depicted in Figure 1.

Results in parsing recipes extends far beyond food: this type of problem also exists in other instruction-based areas like how-tos and engineering requirement documentation.

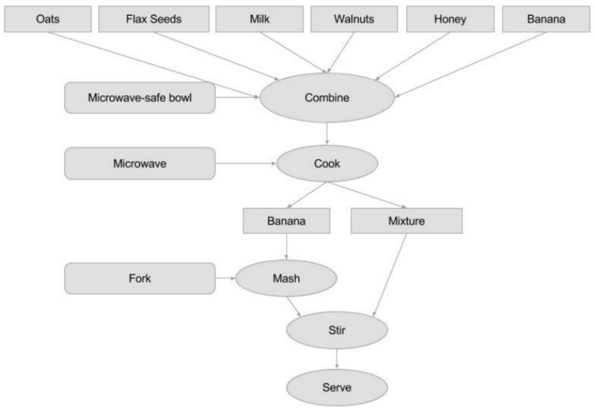

Figure 1: DAG describing an oatmeal recipe [1]

2 Background

2.1 Literature Review

Cooking recipes are plentiful, diverse, and easily accessible on the web. Due to their inherent action-based structure, they are also a great candidate for semantic parsing research. Hence, literature exists on the topic, though most of it is quite recent and the problem remains mostly unsolved.

One of the earliest available papers [2] suggests a minimal instruction language (called MILK) to formally write recipes. This language is similar to using a DAG, but is expressed in text as opposed to a graph and does not use universally accepted notation. Researchers performed some preliminary semantic parsing on a dataset of 300 recipes manually-translated into MILK and 350 unannotated recipes, together known as the CMU Recipe Database. Both a Naive Bayes and Weighed FST approach were tested, each of which achieved an accuracy of around 50%.

The next major milestone in recipe translation was paper [4] written six years later looking to extend state-of-the-art semantic parsing algorithms from other research areas into instruction-based text like cooking recipes or how-tos. The researchers investigated the feasibility and expected results of applying an SRL system to the CMU Recipe Database. They pointed prospective challenges, such as difficulty recognizing predicates as imperative verbs due to lack of relevant training data, and dealing with dropped conjunctions such as “for” or “to”. However, it does not appear that

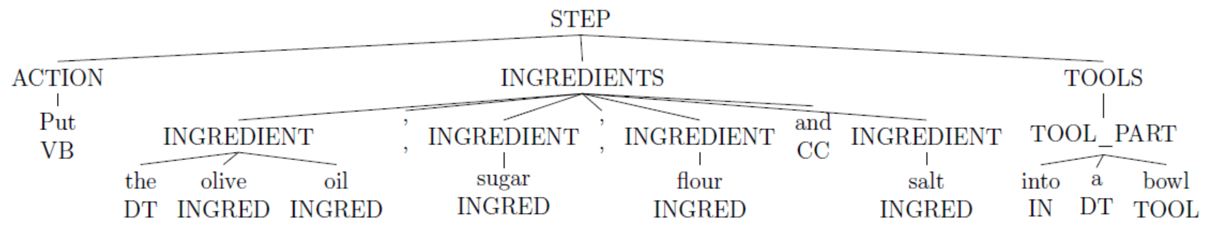

Figure 2: Sample part of speech tree [3]

the researchers created any full-scale parsing implementation beyond simple exploration.

A year later, in 2015, a paper [3] was published that was successful in converting text-based recipes to a dependency tree format at scale. An example tree can be seen in Figure 2. This paper is the first we see that makes concrete progress towards our goal, although the scope appears to be limited to part-of-speech tagging and basic relations, which is insufficient for creating full DAGs. The algorithm can identify part-of-speech with a 97% accuracy and can generate graphical trees with a 70-80% accuracy. While this is not sufficient for reliable commercial use, this paper presents a great baseline for further exploration.

In August 2017, a similar thesis [1] was published exploring statistical learning methods applied to recipe entity and relation extraction using, again, a handmade dataset. The paper presents an annotation framework that describes precise steps to drawing a DAG. An MAE approach to DAGs was attempted, but did not achieve results better than previously published algorithms.

Within the past couple years, several other notable papers on the topic have been published, from attempting to solve entity recognition and relations in a single step [5], to matching recipes with food images [6]. While several papers depict a DAG representation of recipes, they appear to have been generated manually, as in Figure 1. We have not been able to find an existing model for bulk translation of recipes into DAGs.

Among the literature, we also find that many researchers preprocess recipes to reduce variance in text-base format, or alternatively focus on recipes from a single source with consistent formatting. Over the past decade, hRecipe [7], a simple and open-source format for embedding recipe information has become prominent and is currently employed by many online recipe sources.

2.2 Data Source

The dataset we used is called Recipe1M [6], a structured corpus of over one million cooking recipes. Introduced as a joint venture between several academic institutions, this dataset was initially used to construct a computer vision model that relates recipes to food images. In the original paper, researchers postulate that the dataset and embeddings they created provides a basis for many food and cooking-related machine learning tasks beyond image relation.

To our knowledge, this dataset has not been used in the context of DAGs. This is likely because it only contains ingredient, instruction, and source information, and does not provide labels useful for supervised creation of DAGs such as named entities or relations. We are interested in combining this dataset with known algorithms and best practices from recent papers to create a model that effectively translates plain-text recipes into DAGs.

3 Methods

3.1 Exploratory Analysis

Due to the computational expense of working with the full dataset, our exploration scope was three tenths of the dataset, 300,000 recipes.

The recipes are structured into two parts: ingredients and instructions, each of which are in json format. For example, an ingredient might be {`'text': '1/2 pound lean beef, preferably sirloin, sliced as thinly as possible'`} and an instruction might be {`'text': 'Mix the beef with the garlic, soy sauce, and sesame oil and marinate for a few minutes.'`}. Instructions need not be atomic.

To estimate the complexity of included recipes, we counted the number of ingredients and instructions. The average recipe appears to have around 9 ingredients and 10 steps. Some recipes, however, can be incredibly complex, having more than 50 steps or 25 ingredients.

Additionally, we extracted the "collection" or source website from the recipe's URL. Recipes were taken from over twenty popular food websites, with about half from `www.food.com`. From our experience browsing these websites, we know that not all sources provide a similar structure. Hence, we focused on collections following the hRecipe format.

We preprocessed the dataset into 10 smaller files, pickled for further compression. Each file contains similar distribution of training, validation, and test data. To make our manual annotation task easier, we filtered down to one collection of recipes called "Epicurious", which uses hRecipe format. Additionally, we further simplified to recipes with 5 or fewer instruction steps. This reduction still kept enough data for training, validation, and testing (roughly 500, 100, 100) across pickled files.

3.2 Solution Approach

As we did not find a parser that converts recipes directly to DAGs, we had to create a pipeline, or at least an approach to a pipeline, ourselves. DAGs consist of two components and therefore two tasks: named entities which form vertices, and relationships which form edges.

The first task, and the main scope of this project, is named entity recognition specific to the three components of a DAG: ingredients, tools, and activities. We approach named entity recognition iteratively, using various tools to augment and extend a human's intuitive approach to the problem.

The second task, relationship extraction, is outside of the scope of this project. However, we will propose a solution framework based off learnings from the first task.

We used spaCy [8], an open source NLP library with useful tools for tasks like dependency tree creation and part-of-speech tagging. We also used an annotation tool called Prodigy, which is an spaCy extension library coupled with a user interface that helps us train and evaluate models. Prodigy can be used for active learning, but it was most helpful to us for creating ground truth data.

3.3 First Iteration: Seeded Words

Our first model is quite primitive and follows the most basic approach to assigning entity tags: using a pre-constructed dictionary or lookup table. This model mimics a human's behavior by classifying known entities and ignoring all others.

We manually scraped a list of ingredients, tools, and actions from the Internet. This seeding word set was used to extend a traditional English part-of-speech tagging algorithm from spaCy: nouns mapped to either ingredients or tools, and verbs mapped to actions. Results were compared against 350 instructions manually annotated using Prodigy.

3.4 Second Iteration: Bootstrapping

While the seeding word set scraped from the Internet was already quite large, the intuitive next step was to use machine learning to make it bigger and better in an intelligent way. We used a few bootstrapping techniques to create our second model iteratively from the first.

First, we normalized words to their lemma form so we could account for grammar variations. Next, we used cosine similarity to automatically add words from Prodigy's corpus (OntoNotes 5) that were highly similar. Finally, we used Prodigys active learning interface to manually annotate suggestions the algorithm was less confident in.

This created an augmented seeding word set that we again used to extend spaCy's POS tagging algorithm. The new seeding word set was roughly 2-3 times larger than it was before bootstrapping. Note that during the entire process, the seeding word set never used our dataset as an input, therefore reducing potential overfit.

Results were compared against the same 350 manually annotated instructions as before. To follow the human analogy, this model is equivalent to learning more words and recognizing known words even if they are conjugated differently.

3.5 Third Iteration: Neural Network

While we could have further improved the seeding words set, the next logical step was to form a method for predicting labels of words that have never been seen before. Humans do this quite naturally: given the context of a sentence we can deduce the meaning of an individual word even if we can barely pronounce it. We wanted to create the same sort of process within our model. This problem is very well suited for machine learning and, in particular, neural networks.

We decided that a LSTM/RNN variant is not appropriate because each text phrase is relatively short and independent, meaning there's no need

	Precision	**Recall**	**F1-Score**
Seeding Words	0.814	0.524	0.638
Bootstrapping	0.770	0.577	0.660
Neural Network	0.850	0.856	0.853

Table 1: Iterative NER Model Results

to store longer context. Instead, we used spaCy's NER custom neural network model [8]. It uses a pipeline-based process where each step (embed, encode, attend, predict) can be further customized.

The model, described in Figure 3, features a embedding strategy using word-level meta features (normalized, prefix, suffix, shape) and the hashing trick to circumvent out-of-vocabulary words. The encoding uses a depth of 4 CNN with residual connections. The attention uses a mixture of previous tagged entities and words in the buffer. This allows use of entities arbitrarily far back in the text, which differs from a CRF approach conditioned on a fixed number of previous decisions. The prediction uses a multi-class softmax. The custom model also employs transition-based approach to avoid invalid sequences and allow us to easily define arbitrary features. Tuning this custom model to our needs produced a good balance of efficiency, accuracy and adaptability.

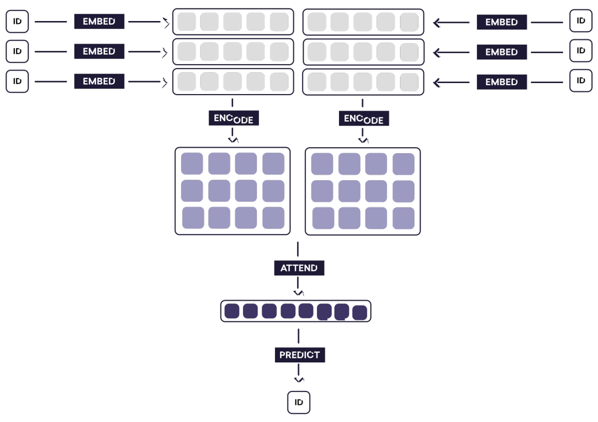

Figure 3: Neural Model Architecture [8]

Because we had taken the time to create a robust seeding word set, the process of manually annotating training data had become quick and efficient. At this point we had collected over 450 manually annotated examples, which we fed in a 70:30 partition to the CNN model.

4 Results and Discussion

4.1 NER Model Performance

Results of the previously described models are shown in Table 1.

For this problem, we are particularly interested in recall as there is a relatively high cost to false negatives: while it is important to bin each recipe entity correctly, we are initially more focused on ensuring no relevant entity goes untagged.

Our preliminary model containing seeding words scraped from the Internet performed decently well. It has a high precision, meaning that there are few false positives. However, recall is quite low. This indicates that our seeding words are not representative of words the model encounters. The major takeaway is that our seeding word set needs to capture a wider variety of patterns.

This was the motivation for the next model, which attempts to address deficiencies of the manually assembled seeding word set through bootstrapping. The second iteration makes an improvement to recall but at a slight cost to precision. This may be because as the seeding word set increases, words begin to appear in multiple lists: for example, "mix" may be both an action and an ingredient, and "spoon" may be both an action and tool. Additionally, we have still not solved for multi-word patterns like "lemon juice", where all three of "lemon", "juice", and "lemon juice" are known to be ingredients. At this point, confusing cases have evolved to the point where manually coding in exceptions is overwhelmingly arduous. Machine learning is a promising solution to the problems this model presents.

This leads directly to our final iteration, a neural network that identifies entities and predicts class based on context. This model resulted a large improvement in recall while bringing precision back up as well. We see a very good balance between the two, which indicates that we are successful in ensuring no relevant entity goes untagged, while also ensuring that tags are more or less correct and non-relevant words remain untagged.

4.2 Proposed Method for a Full DAG Pipeline

In our opinion, many graphical representation attempts by other researchers either missed the critical path or defined overly complicated node and edge rules which made the graph as laborious to read as the text itself.

To draw a DAG on a recipe's critical path, we need to properly define what information we want represented. Although we were not able to explore and implement this part for the project, the following is an outline and commentary on how we would approach the problem:

Rather than drawing a single graph for the entire recipe, separating steps into their own DAGs might be both easier to assemble and easier to interpret. Recipes naturally take on a step-by-step format, and it is easier to connect a smaller amount of nodes than to coordinate how all nodes should be connected. Hence, a bottom-up approach of assembling a DAG for each individual step, with the option of combining the results into a full DAG at the end, seems preferable.

Unlike freeform text, recipes follow an implicit structure and completes a series of smaller goals. Hence almost all relations will likely be bound by a action. Rather than making the effort to learn pair-wise relationships with high accuracy, we can instead pre-structure our DAGs to be centered around each action, and fill in ingredients and tools around the action when available.

A "verbal predicate argument" structure [9] fits well with what we need to map the critical path of a DAG. For example, the sentence "use the knife to cut the apple into pieces" becomes:

- "the knife" (arg1) "use" (verb) "to cut the apples into pieces" (arg2), in addition to
- "the apple" (arg1) "cut" (verb) "into pieces" (goal)

We can make the DAG using this information by matching the argument span with the entities recognized from the NER model. The above example would look like the DAG in Figure 4.

To put these observations together, we propose creating a simple yet concrete DAG notation, for example: tool (left), ingredient (above), action (center), modifier (right), as described by "cut" in Figure 4. Note again, how the diagram centers around the main action or goal of the instruction.

Important modifiers are location, manner, and goal or purpose. Additionally, to include pronouns

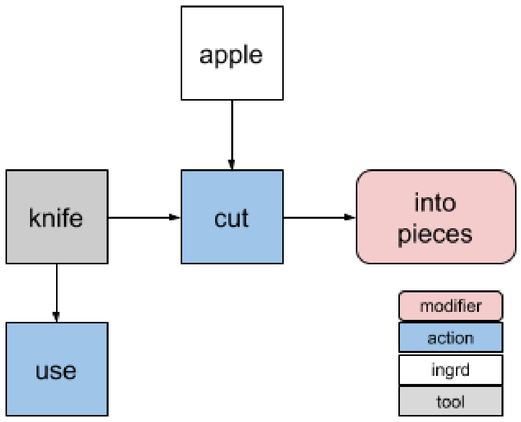

Figure 4: Proposed DAG Representation

as additional ingredient entities (eg. "it", "them"), coreference resolution can be used to print out the noun and pronoun clusters, so one can follow which ingredients are implied.

The Recipe-DAG problem is also difficult in that labeled training data does not exist and is quite difficult to make. Similar to NER, we propose using a bootstrap (semi-supervised) approach to label enough training examples before trying supervised models to generate a generalized solution. Incidentally, with the refined scope, we can approach drawing the DAG as another NER problem!

AllenNLP [10] could be used to implement this along with spaCy as the underlying NLP framework. Specifically, neural network (sequence) based SLR and coreference resolution models will be handy.

While at times powerful, unsupervised learning does not always produce the desired output and can be hard to control. We believe this is the case for recipe parsing. By combining active learning and bootstrapping, a more controllable approach can be achieved that allows use of more familiar and well-established supervised learning techniques. We believe that a model similar to what we have created for the first part of the problem, named entity recognition, can be extended to solve the entire recipe-to-DAG translation process.

5 Conclusions

The Recipe-DAG problem is a very interesting area that extends far beyond food; the results we see here extend to parsing many semi-structured, instruction-based text problems.

We concluded that an unsupervised approach to this problem is too uncontrollable, and that an iterative combination of active learning, neural networks, and bootstrapping is ideal.

We created a model that recognizes recipe entities with a precision, recall, and f1-score of around 0.85. This is a powerful result given the small amount of time that went into the model and the large amount of improvement possibilities that have yet to be exhausted.

Additionally, we distilled the second task of DAG creation, relationship extraction, into an NER-type problem that can be solved following a very similar approach.

References

[1] Yuzhe Chen. *A Statistical Machine Learning Approach to Generating Graph Structures from Food Recipes*. Brandeis University, 2017.

[2] Dan Tasse and Noah A. Smith. *SOUR CREAM: Towards Semantic Processing of Recipes*. Carnegie Mellon University, 2008.

[3] Kristinn Ardal. *Graphical Recipe Automation*. Reykjavik University, 2015.

[4] Jon Malmaud, Earl J. Wagner, Nancy Chang and Kevin Murphy. *Cooking with Semantics*. Association for Computational Linguistics, 2014.

[5] Yankai Lin et al. *Neural Relation Extraction with Selective Attention over Instances*. Tsinghua University and Jiangsu Collaborative Innovation Center for Language Competence, 2016.

[6] Amaia Salvador et al. *Learning Cross-modal Embeddings for Cooking Recipes and Food Images*. Universitat Politecnica de Catalunya, Massachusetts Institute of Technology, and Qatar Computing Research Institute, HBKU, 2017.

[7] Frances Berriman et al. *hRecipe Microformat Specification*. Microformats, 2008.

[8] Matthew Honnibal. *spaCy*. Explosion AI, 2008.

[9] Palmer et al. *The Proposition Bank: An Annotated Corpus of Semantic Roles*. Association for Computational Linguistics, 2005.

[10] *AllenNLP*. Allen Institute for Artificial Intelligence, 2017.