Identification and Placement of Recipe Refinements from User Comments

Ben Schreck, Nico Rakover, Ambika Krishnamachar
https://github.mit.edu/bschreck/robotChef

ABSTRACT

On online recipe websites, a significant portion of user-generated comments include *refinements*, or modifications to the recipe. We propose a method to modify a recipe based on these comments. The method consists of two parts: 1) a neural language model that scores a comment segment on whether it is likely a refinement, and 2) an RNN that scores each recipe segment for how likely a given refinement targets said segment. Both parts assume that refinement wording is more similar than background text to the language used within recipes. Using this assumption, we can frame part 2 as a supervised learning problem -- we train a network by taking an existing recipe, perturbing or removing a segment, and making the network identify the index of the modification given the original segment and modified recipe. By framing this task as two supervised learning problems we hope to surpass the performance of prior, unsupervised approaches.

Introduction

There are a variety of instructional "how-to" resources available on the internet. Recipes make up a huge proportion of this category, with websites hosting millions of them. However, recipes are not necessarily perfect, and the user reviews accompanying a recipe often contain suggestions for improving it. Reviews are often a gold mine for these modifications. On allrecipes.com, many recipes have hundreds and even thousands of reviews, and based on a random sample performed by [1], 57.8% of reviews contain a refinement. Currently, readers of online recipes must read through these reviews themselves to evaluate which modifications, if any, to apply. This manual task is tedious and imperfect, especially when there are possibly conflicting modifications. We hope to improve this user experience. In this project, we explore one facet of this problem space by asking the question: If given a review, can we assess whether it contain a refinement, and if so, where in the recipe does the refinement belong? We are inspired by the work done by "*Spice it Up? Mining Refinements to Online Instructions from User Generated Content*" (Druck and Pang, 2012), who explored the same question. We take a supervised approach on this task and report our results.

We use the same definition of "refinement" as the related paper. A refinement must be a piece of *actionable* information in the review. Druck and Pang exclude *hypothetical* refinements (i.e. "Next time I would add more salt") and *failed* refinements ("I tried doubling the sugar, but it was terrible"), but we do not make this distinction. In theory, a refinement could fall into one of three categories. It could be an insertion between existing recipe steps, a modification to an step, or a deletion of a step. In our work, we focus on insertion and modification only. Furthermore, we do not actually modify the

original recipe steps. Ascertaining the exact modification to the step was beyond the scope of this project. Additionally, it is likely that recipe readers would still prefer to see the original recipe. Instead of modifying the original, a reasonable approach would be to annotate its modified step(s) with the relevant refinements. This way, readers can see both the original and the refinements.

Related Work

Previous research in mining recipe refinements has been done by *Spice it Up?*, which applied an unsupervised generative approach. This approach split both recipes and refinements into segments. Each segment is approximately a grammatical clause, and is typically the size of an atomic refinement. This segmentation allows greater precision in matching refinements to parts of the recipe, and it also allows the model to ignore document structure that can be highly variable between recipes. Using these segments, they apply a mixture of HMMs, and then use EM to maximize the marginal likelihood of the observed reviews. The emissions model used is basically a mixture of bags-of-words. The output is a set of recipe-segment/refinement-segment pairs that represents the alignment of the modifications with the original review text. The authors measure the success of their algorithm at the review level and the segment level. At the segment level, they observe an F1 value of 77.0%.

Other than this domain-specific work, there have been a number of efforts to parse information out of user comments in several other contexts. For example, there have been efforts to locate mentions of adverse drug reactions across a large body of user comments (Nikfarjam and Gonzales, 2011). Additionally, there has been research into intelligently using user comments to improve recommendation systems (for example, product recommendations on a site like Amazon) (Stavrianou and Brun, 2013). This research involved identifying links between comments and the comment target (such as a feature of a product), and using these links to recommend “better” products. While these research areas have similarities to the problem we tackled, many of them are still largely focused on sentiment analysis rather than identification or summarization of actionable information. Additionally, the alignment aspect of our problem (aligning a review segment with a recipe segment) is not well studied.

Methods

We outline our approach and then give a detailed description of each component.

Model Overview

We formalize the problem as follows: Given a review segment and a recipe, what is the index (if any) of the recipe that this review segment belongs?

The way we define “indices” are one per recipe segment, with blank ones in between. This way, both insertions and modifications can be represented uniformly.

We can define the probability of a comment segment *S* referring to recipe *R* at index *i* as follows:

$$P(i \mid S,\ R) \ = \ P(i \mid S \textit{ is a refinement},\ S,\ R) \ \cdot \ P(\ S \textit{ is a refinement} \mid S,\ R)$$

Then we can apply this probability, along with an appropriately selected threshold, to make predictions.

Thus, we have divided the problem into two parts:

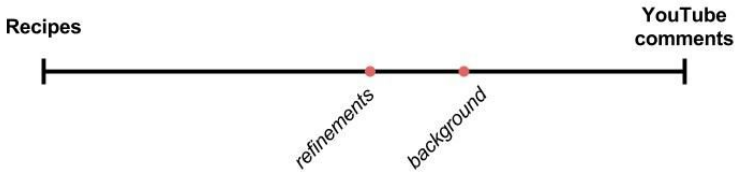
1) For a given comment segment, calculate the probability it is a refinement (a *generic* refinement)
2) Given a refinement, what is the probability that the refinement should be inserted into an index of a given recipe? Note that we don't constrain the sum of probabilities over all the indices of a recipe to sum to 1, as a given refinement can refer to one, several, or none of the indices of a *particular* recipe.

We will then use separate supervised approaches to solve each of the subproblems.

The first subproblem corresponds to the probability $P(S \textit{ is a refinement} \mid S,\ R)$. We will actually make the simplification that $P(S \textit{ is a refinement} \mid S) \ \approx \ P(S \textit{ is a refinement} \mid S, R\ )$, and calculate the left probability. This is justified by the fact that we care about the probability that a given segment is a generic refinement, not necessarily a refinement of the recipe in question.

We will use a language model to evaluate the probability, and we make an important assumption with regard to the language model: We assume that refinements are *more similar* to recipe language than general review background text is to recipe language. This assumption is based on the fact that refinements generally reference details in the recipe ("I used sherry *in place of wine*"), whereas non-refinements are less likely to ("This was great for the whole family!").

If we were to place these types of text on a spectrum, it might look something like this:



Regardless of the absolute position on this hypothetical spectrum, what is really important for the language model is that the language of refinements and the language of the review background text are separable. Given our assumption, we will train the language model on recipes, and then use it to evaluate the probability that a given review segment is a refinement.

The second subproblem corresponds to the probability $P(i \mid S \text{ is a refinement}, S, R)$. We use a recurrent neural network to make a binary prediction for each recipe segments, namely whether the given refinement corresponds to the given recipe segment. We are able to train the RNN by using artificial recipe modifications we generate ourselves.

Now we go into detail about the various components that make up the model.

Segmentation Process

We work at the segment-level for both recipes and reviews. We tried two different segmentation approaches. The first was modeled after the *Spice it Up?* paper, which split recipes and reviews into segments that were approximately the size of a clause. Distinct refinements are usually not longer than this; therefore, segments offer an appropriate granularity at which to match refinements to recipe portions. We also worked with a simpler segmentation model that splits recipes and reviews just on sentence breaks. We tried this model because we found that the heuristic we were using to compute the granular segments yielded many unusual segments that impacted our models' performance. We explain just the more granular segmentation approach here.

The *Spice it Up?* paper used the Stanford parser[5] to split both sentences and reviews into short, phrase-length segments. We found that with the size of our set of recipes and reviews, such a parser would take too long to run to be feasible for use. However, the rules used on top of the Stanford parser to segment the sentences were fairly simple, and we were able to split sentences into phrases with high accuracy using a few heuristics instead. Also, the *Spice it up?* parsing and segmenting performance degraded significantly when applied to ungrammatical review language, so both of these approaches were imperfect. The rules we used to break a sentence into phrases were the following:

- First, split on all end-of-sentence punctuation, semicolons, and colons.
- Then split the resulting segments on commas, and the words "and", "or", "but", "though", and "although". However, if this split produces a segment of less than five words, the short segment is joined with the segment that directly precedes it.

As an example, if the sentence is "I do not like green eggs and ham", even though the word "and" is in the sentence, the segmenter will keep this phrase together as one segment.

Language Model

We train a language model on recipe segments using a simple LSTM-based recurrent neural network. The implementation is a port of the TensorFlow language model tutorial based on Zaremba et al., 2014. The LSTM used is 2-layer cell, with a hidden state size of 200. We used most of the default training parameters intact.

Originally, we let $P(S \text{ is a refinement} \mid S)$ be the likelihood predicted by the recipe language model, which we'll refer to as $score_{RLM}(S)$. However, we achieved much better results (discussed further in the evaluation section) by training an additional language model on the Penn Treebank dataset and combining the predicted likelihoods as follows

$$P(S \text{ is a refinement} \mid S) = \frac{score_{RLM}(S)}{score_{RLM}(S) + score_{PTBLM}(S)}$$

where $score_{PTBLM}(S)$ is the likelihood predicted by the PTB-trained language model. The rationale behind this modification is that we don't care about the grammaticality of a segment, only about its 'refinement-ness', and normalizing by a generic language score helps highlight this quality in a given segment.

Recipe Modifier

We first use an LSTM to encode both the given review segment and each segment in the recipe. At each step, the encoder LSTM takes as input the next word in the segment as well as the previous hidden state, and eventually generates an embedding of the full segment. These embeddings are passed to a second LSTM, which, at each index in a recipe, takes the concatenation of two embeddings as input: the embedding for the review segment, and the embedding for the recipe segment at the current index. The LSTM outputs a vector of scores of length n, where n is the number of indices in the recipe. Each number in the vector is between 0 and 1 inclusive. The score for index *i* in the vector specifies the probability that the refinement segment corresponds to the ith index of the recipe.

We train the recurrent neural network with artificial modifications that we feed into the network. Training data is not readily available in this domain, so we have an alternate approach. Instead of using existing modifications, we can choose an segment at random from the recipe. We can then *remove* or *alter* this segment (we detail the alteration process in a subsequent section), and then have the RNN 'guess' where the *original* segment belongs in the modified recipe. Because in this case we do know the exact index where the segment originally resided, we can apply a loss function to adjust the weights.

The second LSTM outputs a vector of scores. To train the model, we can define a loss function as the difference of this score vector and the 'true' vector of scores, which will effectively be a one-hot-vector. Additionally, we tried incorporating an attention model, where we run this second LSTM twice, both forward and backward over the recipe. This generate two score vectors. We then train two attention weight vectors that multiply either the forward or backward score, and add the result.

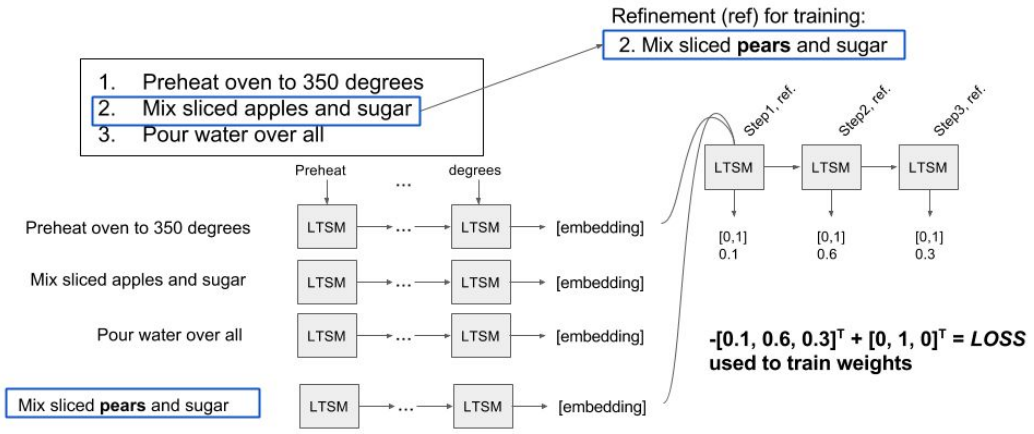


Figure 1: The RNN training procedure.

Generation of Training Refinements

Our goal in creating artificial recipe refinements was to help the RNN learn more generally how to place refinements in a recipe. In determining how to construct these artificial refinements, we observed general patterns in recipe refinements. Common recipe refinements include substitutions for one ingredient for another, and insertions with references to the surrounding steps ("Before frying the chicken..."). Additionally, there are many adjustments to quantities of ingredients used, either noted by the words "increase" or "decrease", or by directly specifying the new quantity ("use 3 cups of milk").

At first, we considered generating artificial refinements that would help the model learn these patterns. This method would involve keeping sets of common substitutions across different categories of food (for example- pasta, rotini, spaghetti, linguini....), and generating modifications that randomly chose from these substitutions. Additionally, we could create modifications by selectively incrementing or decrementing ingredient quantities, and augmenting these modifications with keywords "increase", "decrease", "more", "less", etc.

We realized that this approach, while likely to learn the rules we had provided and the similarity word banks we had generated, would generalize poorly when applied to real recipes and modifications. Additionally, we found it difficult to capture any of the nuances that come with recipe ingredient substitution (for example, whether baking the chicken instead of frying it is an acceptable substitution is highly dependent on the specific recipe).

Instead, we noticed that the similarity of a refinement to the original sentence was probably a more accurate, and generalizable, measure. Thus, to generate artificial modifications, we would perturb a specific recipe segment in one of the following ways:

1. Partial segment removal - Remove a random chunk of the segment.
2. Internal word swap - Randomly swap pairs of words in the segment.
3. Random word substitute - Substitute a small number of random recipe dictionary words with words in the segment.

Effectively what these rules should do is teach the network that a bag-of-words / word-counts-vector representation is a suitable one, with the additional benefit that our network has access to context within a recipe, not just the single segment in question. We believe this might work reasonably well in practice because the HMMs used in [1] relied on bag-of-words emission models.

Evaluation

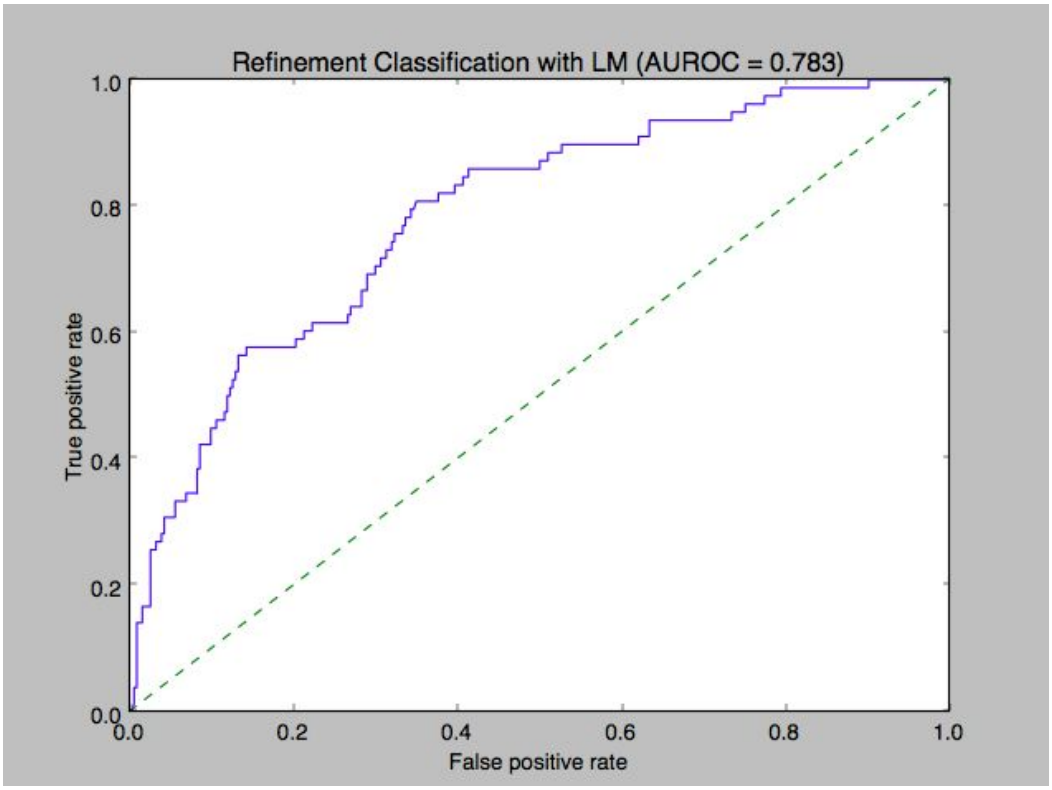
We scraped data from allrecipes.com, including 60,000 recipes, and around 10 reviews per recipe. We manually labeled approximately 1000 refinement sentences. This limited number was due to time constraints. We performed 3 separate evaluations. The first two were on how well our models did on the two individual unsupervised problems. This only gave us directions to move forward and alter parameter settings, and give us confidence that the models would work on the joint model. We list some numbers for these evaluations below. To evaluate the full joint model, we needed to run both the language model and the refinement index locator model together, on our small labeled dataset. We present some evaluation numbers below. The results indicate high accuracy but low precision and recall, indicating a somewhat skewed dataset (toward negative examples, or segments that are not refinements). However, we were unsure about the correct index of many of these manually labeled recipes, so our maximum precision and recall is much lower than 1. More work remains to be done in terms of analyzing how well these results compare to the *Spice It Up?* paper, especially since our work was performed on non-parsed sentences and evaluation was performed on a limited dataset.

We grouped data into buckets representing the number of phrases/sentences in the recipe, and the length of the longest phrase/sentence in the recipe.

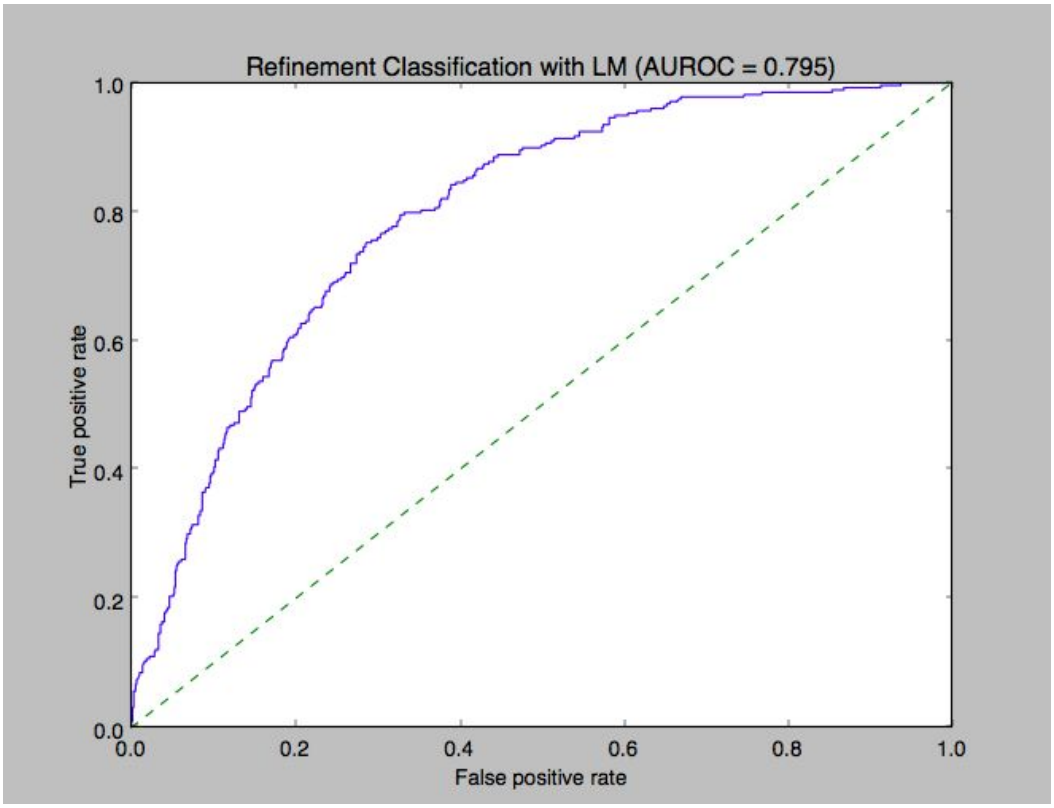
The buckets are:
Bucket 0: (maximum of 10 phrases, maximum of 15 words per phrase)
Bucket 1: (maximum of 15 phrases, maximum of 20 words per phrase)
Bucket 2: (maximum of 20 phrases, maximum of 25 words per phrase)
Bucket 3: (maximum of 25 phrases, maximum of 30 words per phrase)

We ignored recipes that did not fit in any of the buckets because when we included them the model used too much memory and would not run. We believe this is a serious flaw in Tensor Flow, as we were not able to limit the number of steps it unrolled for performing backpropagation on these long lists of words and phrases. Furthermore, as some context, even while limiting ourselves to these 4

buckets, the recipe-modifier model takes between 10 and 20 minutes to initialize on a machine with 12 3.5GHz i7 cpus, a 1.5GHz Nvidia Ge-Force GTX 970 with 4GB of RAM, and 32 GB of main memory RAM.



Performance on phrase-level segments



Performance on sentence-level segments

Language model

(ROC plots shown above)

-- on phrase-level segments:

best F1 = 54.3%

best threshold = 0.9054

-- on full sentences:

best F1 = 57.9%

best threshold = 0.7098

Recipe Modifier perplexity

-- on phrase-level segments, after 17,900 steps:

train: 2.0

eval, bucket 0: 3.0

eval, bucket 1: 2.6

eval, bucket 2: 2.0

eval, bucket 3: 1.7

-- on full sentences without attention, after 6600 steps:

train: 1.4

eval, bucket 0: 2.0
eval, bucket 1: 1.4
eval, bucket 2: 1.3
eval, bucket 3: 1.2

-- on full sentences with attention, after 11,000 steps:

train: 1.3
eval, bucket 0: 1.4
eval, bucket 1: 1.3
eval, bucket 2: 1.2
eval, bucket 3: 1.1

Note that the model with attention performs better, but this is after many more iterations than the model without attention. We unfortunately did not keep track of the perplexity at earlier steps for this model, and so we cannot fully compare the two. Given more time, we could produce a comparison. Furthermore, the perplexities numbers, especially on the eval set, were highly variable, as they were calculated per 100 steps. They could vary by as much as +-.3 every time they were calculated.

End to End Model Performance

Top 1

predicted correct index with highest probability, or correctly predicted no index

precision: 0.182

accuracy: 0.764

recall: 0.114

TOP 2

predicted correct index with at least 2nd highest probability, or correctly predicted no index

precision: 0.18

accuracy: 0.764

recall: 0.208

TOP 3

predicted correct index with at least 3rd highest probability, or correctly predicted no index

precision: 0.182

accuracy: 0.764

recall: 0.265

Baselines

Since we were unable to get access to the original dataset from [1], we had to resort to more ad-hoc baselines. For the task of identifying the best index into a recipe given a refinement, we used a simple word-counts-vector approach. For modification and insertion refinements (separately), we used used two vector distance metrics to identify the best index into a recipe: cosine similarity and Bray-Curtis

distance. For modifications, we simply find the recipe segments closest to the refinement vector. For insertions, we find the pair of consecutive recipe segments that, averaged together, are closest to the refinement vector. Note that these tasks are arguably easier than the task we give our network, since the review segments used in the baselines are known to be refinement, and moreover, pre-distinguished as modifications or insertions (we model each separately in the baselines).

-- on the phrase-level segments:
- Modifications: 65 examples
  - Using *cosine similarity*
    - Top 1 error: 72.3 %
    - Top 3 error: 53.8 %
  - Using *Bray-Curtis distance*
    - Top 1 error: 70.8 %
    - Top 3 error: 49.2 %
- Insertions: 13 examples
  - Using *cosine similarity*
    - Top 1 error: 69.2 %
    - Top 3 error: 53.8 %
  - Using *Bray-Curtis distance*
    - Top 1 error: 69.2 %
    - Top 3 error: 53.8 %

-- on the sentence-level segments:
- Modifications: 225 examples
  - Using *cosine similarity*
    - Top 1 error: 56.9 %
    - Top 3 error: 24.4 %
  - Using *Bray-Curtis distance*
    - Top 1 error: 57.8 %
    - Top 3 error: 22.2 %
- Insertions: 54 examples
  - Using *cosine similarity*
    - Top 1 error: 88.9 %
    - Top 3 error: 57.4 %
  - Using *Bray-Curtis distance*
    - Top 1 error: 88.9 %
    - Top 3 error: 59.3 %

Conclusion

We proposed a method to modify a recipe based on user-generated reviews. The approach consisted of two parts: first, a neural language model to assess whether or not a given review segment is a refinement, and second, a RNN to compute a score for how well the review segment targets a

particular index in the recipe. In this way, we frame the problem as two supervised learning problems. Our model shows promise, and our assumptions seem reasonable, but it needs more work and more time to perfect. We believe that, given more time, we could develop our model enough so that it performs better than the unsupervised HMM approach in the *Spice it Up?* paper. As of now, we do not have the data to make an apples-to-apples comparison of the two approaches.

Future Work

There are several areas of future work that we would pursue. First, we found that our segmentation approaches (both the heuristic phrase-level segmentation and sentence-level segmentation) were often unable to accurately segment review phrases - Often, review phrases would be split between segments, or multiple modifications would make up the same segment. In future, we would incorporate the use of the Stanford parser or a similar tool and see whether that helps achieve better segmentation.

Our approach for generating artificial recipe refinements was simple, and it taught the RNN to match refinements to indices largely based on a bag-of-words model. While this is generalizable, there are more sophisticated ways we could generate training data. We would explore more recipe-specific patterns of modifications that could still be generalizable, and we could also generate large amounts of realistic training data through a crowd-sourced tool like Mechanical Turk.

References

[1] Druck, Gregory, and Bo Pang. "Spice it up?: mining refinements to online instructions from user generated content." *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*. Association for Computational Linguistics, 2012.

[2] Nikfarjam, Azadeh, and Graciela H. Gonzalez. "Pattern mining for extraction of mentions of adverse drug reactions from user comments." *AMIA Annual Symposium Proceedings*. Vol. 2011. American Medical Informatics Association, 2011.

[3] Stavrianou, Anna, and Caroline Brun. "Expert Recommendations Based on Opinion Mining of User-Generated Product Reviews." *Computational Intelligence* (2013).

[4] Zaremba, Wojciech, Ilya Sutskever, and Oriol Vinyals. "Recurrent neural network regularization." *arXiv preprint arXiv:1409.2329* (2014).

[5] http://nlp.stanford.edu/software/lex-parser.shtml