James Mao Spring 09-10
CS 224N - Natural Language Processing

Final Project
Modification Identification in Recipe Comments

1 Abstract

Recipes are commonly modified for health, taste, ingredient availability reasons. A summary of modifications by other users of a recipe is useful in determining whether a proposed modification will be successful. We present a system for identifying modifications in comments posted on recipe websites. As a secondary goal, our system design process demonstrates how we can combine human understanding and machine learning techniques to rapidly construct a practical application with a minimal of annotated data.

2 Introduction

2.1 Background

There is a proliferation of recipe websites and data available on the internet. While tools exists to assist in the discovery of the best recipes (e.g., recommendation systems based on user ratings and history ala Netflix), there is no quick way of determining whether a particular recipe will is suitable. Typically, users are forced to skim through large number of comments, easily exceeding hundreds on the most popular recipes, in order to determine whether they can make a particular substitution (e.g., to replace an ingredient not on hand with one that is available) successfully. This is essentially a sentiment classification task ("I think this recipe should be modified"). In this domain, we are helped by the domain specific jargon[1] but we are hurt by the subtleness of sentiments[2]. We will attempt to construct a system that collects the domain specific factors through unsupervised learning and use a small human annotated data set to learn about the semantics of various sentiments.

2.2 Design Process

Our main design methodology was to quickly build a commercially viable system (i.e. one that can be plugged into an existing website immediately). We do this through a combination of code reuse and smart pipeline development. We try to avoid custom code whenever possible, opting for well supported libraries instead. Our processing pipeline is designed to be fast and robust, but more importantly, the process to design the pipeline was designed to be intuitive. That is, we try to demonstrate a method for developing other such classification tools quickly. This was achieved through a combination of supervised machine learning with small datasets and human reasoning. In the end, we hope to have gained insights into the problem rather than just a set of coefficients. As a result, we prefer methods like decision trees and feature based classifiers over principal component analysis based classifiers.

3 Data

3.1 Sources

Most recipe websites provide some sort of commenting facility and can be used as the input data. We chose the highest rated recipes for the most popular chefs on the Food Network website (`http://www.foodnetwork.com`).

3.2 Collection, Extraction, and Annotation

Since our goal is to leverage common libraries (which may be available in different programming languages), we used an intermediate YAML representation for the data. This allows us to decouple the data collection, extraction, and annotation task from the classification task.

We created a Ruby based crawler that extracts a list of the most popular chefs and the raw markup for each of their top recipe pages. We leveraged the `mechanize` gem (a port of `WWW::Mechanize` for Ruby) to perform the core crawling task (e.g., following "Next" links to gather recipe listings that are paginated). Once we have the HTML, we use the `Nokogiri` gem (an HTML parser with the ability to search using XPath) as the core of our post processor. We extracts title, ingredients, and procedure for each recipe and the rating, title, and body for each comment to a recipe using hand crafted XPaths. The advantage of using an XPath based extractor, which uses a parsed tree, over one that uses regular expressions is ease of change. That is, we can quickly adapt our crawler and extractor to changes in the source website or to different websites. Once the data is extracted, it is stored in YAML files by recipe.

The crawler and post processing is built on top of a `Resque` (a `redis` backed message queue). This allows us to parallelize the operations for performance and fault tolerance (e.g., timeouts may stall a particular fetch but the overall performance hit due to timeouts is negligible when the fetch is parallelized).

Before annotating, we need to split the comments into sentences. At first, we tried using our custom regular expression based tokenizer but found that the Punkt sentence tokenizer[3] works much better (this also fits with our philosophy of using well maintained libraries whenever possible). There is an implementation of this tokenizer in NLTK (the Natural Language Toolkit for Python). In order to leverage that implementation, we switch to Python for the remainder of our processing flow. We created a console annotation program to annotate each sentence for selected comments after tokenizing.

4 Processing Pipeline

After the annotation process, we extract a variety of features and feed them to a maximum entropy classifier (described in section 7). The complete processing pipeline is illustrated in figure 1.

5 Testing Methodology

As mentioned in our design process, our goal is to build a useful system using a minimum of human annotated data. For our data set, we tagged a couple of recipes with over 100 comments, a few recipes with tens of comments, and about a dozen recipes with less than 5 comments. From what

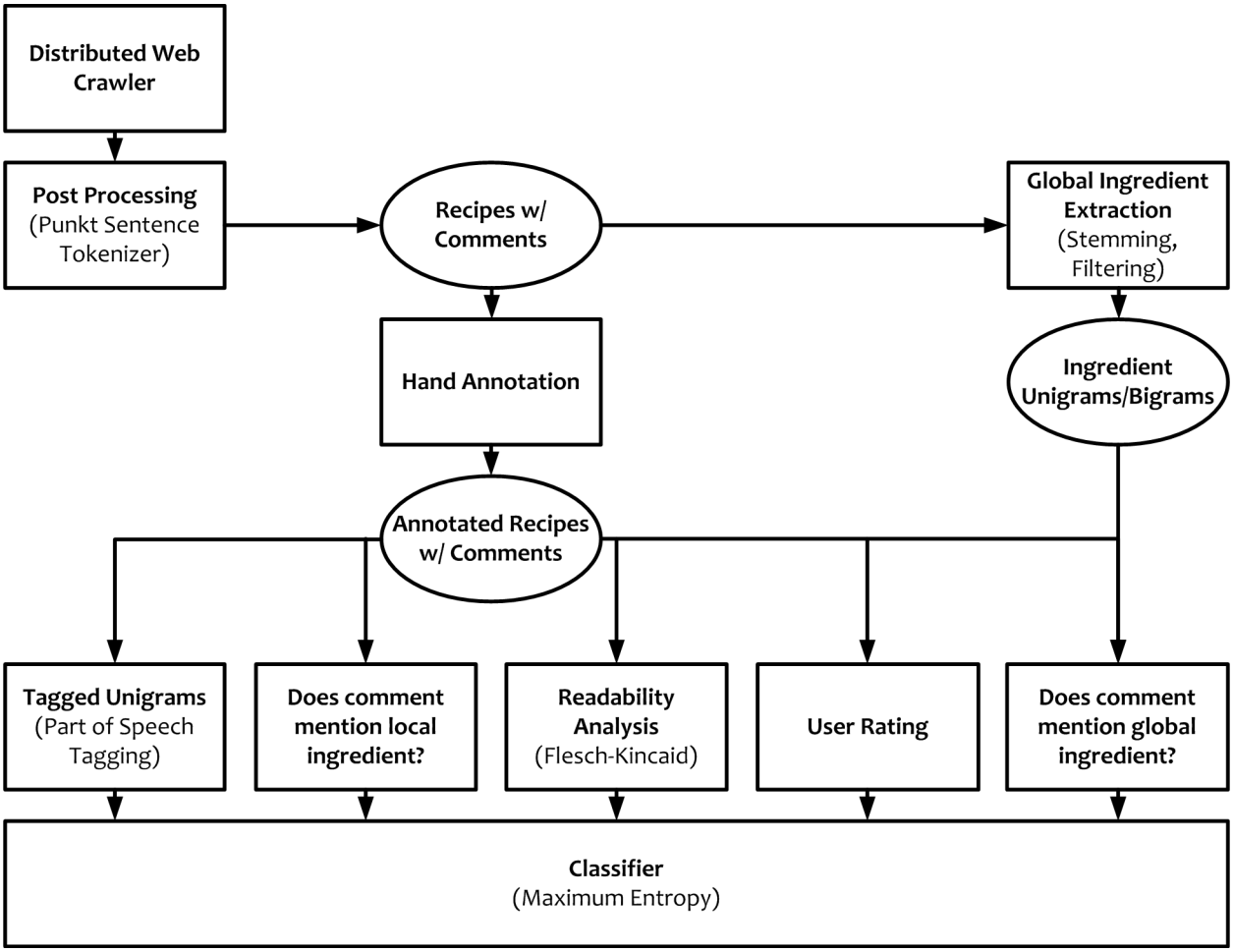

Figure 1: Our processing pipeline.

we observed, this is approximately the same distribution as the entire recipe library we constructed. In particular, we add the recipes with few comments as a way to demonstrate the generality of our system. If we only include the recipes with lots of comments, our classifier will be rewarded for over fitting. The smaller recipes use different vocabulary so it generally reduces performance. We wanted to create a scenario representative of the real world.

Because of the small data set, system performance can be heavily dependent on training data and test data selection. To avoid any semblance of bias, we use repeated random sub-sampling cross validation. Our data set contains approximately 1100 sentences. We random choose 250 of these as the test set and use the rest as the training set.

6 Baseline Naive Bayes Classifier

For comparison purposes, we use a naive Bayes classifier that uses the words of a sentence and the rating of the recipe by the commenter as the features. The performance of this classifier suffers from several problems. First, naive Bayes does not handle duplicate features gracefully and overweights them. In addition, this classifier over fits to the particular recipes in our data set. It also severely over predict the number of substitutions. As seen in figure 2, the naive Bayes classifier has high recall but low precision because of uneven class distributions[4]. Despite these failings, it serves a good baseline for comparison and we can learn from the features identified.

Below we have a list of the most informative features from the classifier. We note that there are only word features and no rating features. This is expected due to naive Bayes classification's problem with feature correlation. From this, we can identify three separate types of features. Some words, like `beef`, `lamb`, `pork` and `turkey` are ingredients from a particular recipe, a three meat meatball recipe (one of our recipes with over 100 comments). In particular, the sentiment is that most people substituted the beef, lamb, and pork mixture with beef or turkey. The actions of doing so are contained in words like `omitted`, `except`, `instead`, and `used`. `omitted`, `except`, `instead` are very good indicators that a substitution has occurred and appear near the top of the list correspondingly. `used` is must tricker. It is sometimes used in the substitution sense, "*I didn't have the mini muffin pan so I used a regular one*", and sometimes used in the complement sense, "*I have used these with spaghetti and in sandwiches and I get raves*" (this is described in more detail later). The third class of words are other nouns that indicate substitution. It turns out that `children` are picky and people have different tastes for `spicy` recipes.

```
Most Informative Features
          contains(beef) = True                T : F      =     60.1 : 1.0
          contains(hand) = True                T : F      =     51.3 : 1.0
          contains(lamb) = True                T : F      =     42.2 : 1.0
          contains(pork) = True                T : F      =     39.6 : 1.0
       contains(omitted) = True                T : F      =     37.6 : 1.0
        contains(turkey) = True                T : F      =     37.6 : 1.0
        contains(flakes) = True                T : F      =     30.8 : 1.0
           contains(1/2) = True                T : F      =     30.8 : 1.0
```

```
   contains(ground) = True              T : F      =     28.0 : 1.0
        contains(1) = True              T : F      =     24.9 : 1.0
   contains(except) = True              T : F      =     23.9 : 1.0
    contains(round) = True              T : F      =     23.9 : 1.0
  contains(instead) = True              T : F      =     21.4 : 1.0
   contains(grated) = True              T : F      =     18.5 : 1.0
    contains(water) = True              T : F      =     17.1 : 1.0
contains(children) = True               T : F      =     17.1 : 1.0
   contains(olives) = True              T : F      =     17.1 : 1.0
    contains(panko) = True              T : F      =     17.1 : 1.0
    contains(wheat) = True              T : F      =     17.1 : 1.0
    contains(chuck) = True              T : F      =     17.1 : 1.0
    contains(spicy) = True              T : F      =     17.1 : 1.0
    contains(added) = True              T : F      =     17.1 : 1.0
    contains(extra) = True              T : F      =     14.4 : 1.0
      contains(fan) = True              T : F      =     14.4 : 1.0
    contains(pound) = True              T : F      =     13.2 : 1.0
contains(followed) = True               T : F      =     12.5 : 1.0
   contains(crumbs) = True              T : F      =     12.5 : 1.0
   contains(pepper) = True              T : F      =     12.1 : 1.0
     contains(used) = True              T : F      =     10.6 : 1.0
```

7 Maximum Entropy Classifier

We tried several various classifiers including AdaBoost, random forests, and support vector machines before settling on using a maximum entropy Markov model based classifier. The alternatives often suffered from a combination of performance problems, over fitting to data, and poor handling of correlated features. Most of the alternative classifiers were implemented using R while the maximum entropy classifier is from NLTK.

7.1 Features

7.1.1 Unigrams

We use split each sentence into words (using `nltk.word_tokenize`) and use each lower case word as a feature. We tried using a threshold (i.e. ignore words that occur infrequently) based on the global word distribution but that did not seem to enhance performance. It is likely that the rare words are indicative of other factors (e.g., poor spelling, special ingredients generally not used in recipes). Therefore, this global word extraction and thresholding filter was not included in our final classifier. As with our other word based features, we ignore the case of the word.

7.1.2 Ratings

Recipes were ranked between 1 and 5 stars by each commenter. We use this directly as a feature in our classifier. This was also used in the baseline classifier but the gets lost in the noise as the naive Bayes algorithm overweights the word features.

7.1.3 Local Ingredients

We add features for words in a sentence that are also in the local ingredients (i.e. the ingredients for the recipe that is the topic of the current sentence).

7.1.4 Global Ingredients

Using a dedicate program, we gather all the words used in recipe ingredients from all the recipes (including the unannotated recipes) and generate a unigrams and bigrams. We remove the infrequent occurring tokens with a fixed threshold. Words in each sentence are compared to this list and features are created for unigram and bigram matches.

7.1.5 Part of Speech Tagged Unigrams

We use a pre-trained tagger from NLTK to tag the sentence and use each word-tag pair as a feature. The pre-trained tagger has accuracy of about 60% on this data set (casual observation, not statistically tested due to data set size). This can likely be improved with a larger data set and more testing.

7.1.6 Readability

From our feature engineering efforts, we saw that the style of the sentence affects the likelihood of substitution. Looking at the data, we can note several different types of authors. There are ones who write very detailed reviews that discuss everything, often including a substitution. There are other authors who comment just to give praise or criticism (typically shorter and less erudite). These different authors use different style and vocabulary (similar to the detection of non-native speakers in [5]). We use a rounded Flesch-Kincaid Grade Level as a feature to capture some of the style differences. The part of speech tagged unigrams also help to capture the different style and vocabulary.

7.1.7 Sentence Position

We use the position of a sentence in a comment as a feature. This adds a small boost but it may not be statistically significant.

8 Results

Our maximum entropy classifier with all the features (as described in section 7) was cross validated against the naive Bayes classifier. The results are provided in figure 2. Our classifier achieve significantly higher on precision but lower on recall (it is hard to better on recall when compared to a baseline classifier that is grossly over tagging the substitution class). For our application, automated summary and change discovery, we care more about precision than recall. A low precision means that the filtering is not very useful and the user sees too many comments. In practice, the naive Bayes classifier is not much better than quickly skimming all the comments. A low recall means we may miss some of changes used by the users. This is usually not a problem since the same substitutions are repeated by multiple users. For example, many people replace panko (a Japanese style breadcrumb) with the regular American variety.

The output of our classifier generally identifies most of the groups of substitutions in most recipes.

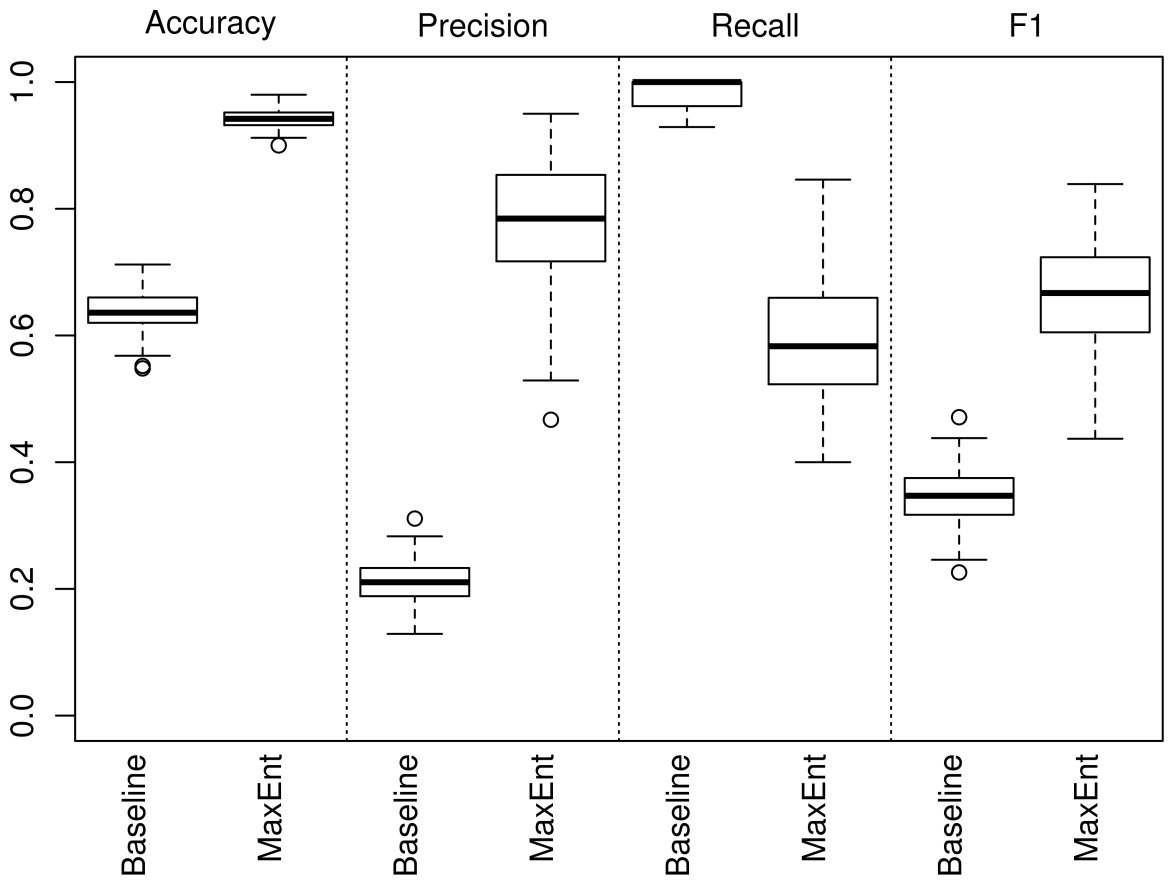

Figure 2: Repeated random sub-sampling cross validation results using 100 iterations with 250 test data.

We tested this by classifying previously unseen recipes and looking the results tagged as substitutions. Qualitatively, there are lots of false positives (we only look at the results marked as substitutions). However, running the output through a k-means clustering system or identifying common phrases using latent semantic indexing, we get a reasonable set of substitutions. We believe that recipes websites should offer this immediately as it provides a more useful view of the comments (rather than the chronological ordering that most recipes websites current use). Automatic summarization (such as that described in [6]) is sufficient to make the output of our classifier production ready.

From analyzing the results, we also notice that substitutions generally mention the same ingredients. It is possible to use a measure like tf-idf to identify the ingredients that are mentioned more often than expected in a comment set. Conveniently, we already have the ingredient data to perform such a post processing. In a quick test, this performs similar to clustering. However, since we are using a dictionary (unsupervised learning from the ingredient lists provided with recipes), this behaves more consistently than clustering. Clustering sometimes combines mislabeled sentences like "*Alton is my favorite*" or "*I love this recipe*". Using the dictionary will prevent sentiment other than ingredient substitution from being identified.

8.1 Errors

8.1.1 Subtleness of Sentiment

The main source of misclassification is the subtlety of human sentiment. This is mentioned for the movie domain in [2] but we'll provide an example in this domain. Consider the feature `contains_tagged_word(used/VBD)==True and label is True`. The verb `used` can indicate many different sentiments.

i. **Other readers may have substituted by accident but I did not.** - "*For those who got dry meatballs, perhaps you didn't realize that on the show he used FRESH bread crumbs, not the dry stuff in the can.*"

ii. **I substituted a cooking utensil.** - "*I didn't have the mini muffin pan so I used a regular one.*"

iii. **I am suggesting a complement food or alternative serving method.** - "*I have used these with spaghetti and in sandwiches and I get raves.*"

iv. **I subclassed this recipe.** - "*i even used this recipe to make a meat loaf I loved it so much.*"

v. **Used...only to mean that I omitted ingredients.** - "*I, too, used only ground beef (no lamb or pork on hand) and they turned out great*"

vi. **Used...instead to mean that I substituted ingredients** - "*I used Deer burger instead of the meat that he used and Itailian sausage.*"

For our problem, we consider only (v) and (vi) to be ingredient substitutions. Given the challenge of choosing between all these sentiments, the results in figure 2 seem very good.

8.1.2 Over Fitting

We did our best to prevent over fitting to the annotated data. The most significant features from an earlier version of our algorithm are given below. It includes `veal` and `beef` which are specific to the recipes we annotated.

```
7.131 rating==3 and label is False
6.181 contains(!)==True and label is False
4.457 rating==1 and label is False
4.242 contains(added)==True and label is True
4.108 contains(veal)==True and label is True
4.024 contains(.)==True and label is False
3.964 contains(beef)==True and label is True
3.788 contains(these)==True and label is False
3.761 contains(used)==True and label is True
3.755 contains(substituted)==True and label is True
3.574 contains(from)==True and label is False
3.454 contains(substitute)==True and label is True
3.402 contains(except)==True and label is True
3.115 contains(you)==True and label is False
3.111 contains(best)==True and label is False
```

In our final classifier, we improve this by skipping word features that are in our global ingredient lists (and using a low threshold for that list). A similar listing of top 100 features only includes `italian`, `pimento`, and `catfish`. We believe that this makes our classifier more general and allows us to have confidence that our results will be similar on different recipes. Essentially we are making our system more like an unsupervised classifier by reducing amount of annotated data required. This comes at a significant cost to our metrics. Without taking steps to reduce over fitting, our precision, recall, and F1 scores are about 0.1 higher than depicted in figure 2.

```
2.040 contains_tagged_word(italian/JJ)==True and label is False
1.883 contains_word(pimento)==True and label is True
1.516 contains_word(catfish)==True and label is True
1.516 contains_tagged_word(catfish/JJ)==True and label is True
```

8.2 Ideas for Improvement

8.2.1 User Specific Factors

The recipe rating is key feature in our classifier. In general, recipes rated 3, 4, or 5 are less likely to include substitutions. Surprisingly, recipes rated 1 are also unlikely to include substitutions. Recipes rated 2 are ambiguous. Rating related features are listed for a run below (results are meant to be qualitative rather than quantitative as actual weights depend on the partition between training and test data). We may be able to improve our classifier by taking into account a particular user's rating history. This is a problem similar to the Netflix challenge. Based on our experience with that problem, it is probably useful to normalize ratings for both the average recipe rating and the average user rating. In addition, we would likely benefit from using the history of our predictions with this users. If we have confidence that a particular user is very picky (or arrogant or skilled) and often suggests substitutions then we should use that as a factor.

```
1.768 rating==5 and label is False
1.668 rating==3 and label is False
1.291 rating==4 and label is False
0.851 rating==1 and label is False
```

8.2.2 Long Distance Patterns

This goes against our goal of quick development, we would get better results by using pattern based features. For example, we can use 'used...only' and 'used...instead' to capture some common substitution mechanics. We avoided doing such because of our goal of rapid prototyping. Such patterns would be language specific. Currently, we do not rely on any language specific factors. Our system can be targeted toward almost any latin alphabet language with a simple modification (replacing the part of speech tagger and stop words with language specific versions). For other languages, we would need to use a method of tokenizing the strings such as the JUMAN system[1].

8.3 Example of Feature Weights

The following is an example of the feature weights in our final classifier. This shows that our system mostly use semantic cues from the sentences which should generalize across recipes(similar to [7]) and we did not over fit. As a result, the qualitative output is very good as discussed above.

```
5.757 contains_word(!)==True and label is False
5.628 has_global_ingredient_bigram==True and label is True
5.249 rating==3 and label is False
4.245 contains_tagged_word(./.)==True and label is False
4.245 contains_word(.)==True and label is False
4.142 contains_tagged_word(added/VBD)==True and label is True
4.077 rating==4 and label is False
3.758 contains_tagged_word(i/PRP)==True and label is True
3.689 length==8 and label is False
3.686 contains_tagged_word(!/.)==True and label is False
3.647 contains_word(added)==True and label is True
3.509 contains_word(i)==True and label is True
3.072 grade==2.0 and label is False
2.685 rating==5 and label is False
2.657 grade==5.0 and label is False
2.593 has_local_ingredient==True and label is True
2.523 contains_word(substituted)==True and label is True
2.496 contains_word(used)==True and label is True
2.383 contains_word(omitted)==True and label is True
2.341 grade==8.0 and label is True
2.337 rating==1 and label is False
2.319 contains_word(from)==True and label is False
2.319 contains_tagged_word(from/IN)==True and label is False
2.281 contains_tagged_word(this/DT)==True and label is False
2.281 contains_word(this)==True and label is False
2.277 contains_word(these)==True and label is False
2.277 contains_tagged_word(these/DT)==True and label is False
2.275 contains_word(them)==True and label is False
2.259 contains_tagged_word(had/VBD)==True and label is True
2.185 contains_word(are)==True and label is False
2.185 contains_tagged_word(are/VBP)==True and label is False
2.115 contains_tagged_word(1/CD)==True and label is True
2.115 contains_word(1)==True and label is True
2.095 contains_tagged_word(with/IN)==True and label is False
2.095 contains_word(with)==True and label is False
2.083 contains_tagged_word(omitted/VBD)==True and label is True
2.040 contains_tagged_word(italian/JJ)==True and label is False
2.039 contains_tagged_word(them/PRP)==True and label is False
1.899 grade==3.0 and label is False
1.883 contains_word(pimento)==True and label is True
1.865 length==1 and label is False
1.848 contains_tagged_word(except/IN)==True and label is True
1.848 contains_word(except)==True and label is True
1.805 contains_tagged_word(only/RB)==True and label is True
1.783 contains_word(loved)==True and label is False
1.776 contains_word(than)==True and label is False
1.776 contains_tagged_word(than/IN)==True and label is False
1.773 contains_word(over)==True and label is False
```

```
1.730 contains_word(think)==True and label is False
1.726 grade==-15.0 and label is False
1.701 grade==6.0 and label is True
1.694 grade==1.0 and label is False
```

References

[1] R. Hamada, I. Ide, S. Sakai, and H. Tanaka, "Structural analysis of cooking preparation steps in japanese," in *IRAL '00: Proceedings of the fifth international workshop on on Information retrieval with Asian languages*. New York, NY, USA: ACM, 2000, pp. 157–164.

[2] B. Pang, L. Lee, and S. Vaithyanathan, "Thumbs up?: sentiment classification using machine learning techniques," in *EMNLP '02: Proceedings of the ACL-02 conference on Empirical methods in natural language processing*. Morristown, NJ, USA: Association for Computational Linguistics, 2002, pp. 79–86.

[3] T. Kiss and J. Strunk, "Unsupervised multilingual sentence boundary detection," *Comput. Linguist.*, vol. 32, no. 4, pp. 485–525, December 2006. [Online]. Available: http://dx.doi.org/10.1162/coli.2006.32.4.485

[4] Y. Yang and X. Liu, "A re-examination of text categorization methods," in *SIGIR '99: Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*. New York, NY, USA: ACM, 1999, pp. 42–49.

[5] L. M. Tomokiyo and R. Jones, "You're not from 'round here, are you?: naive bayes detection of non-native utterance text," in *NAACL '01: Second meeting of the North American Chapter of the Association for Computational Linguistics on Language technologies 2001*. Morristown, NJ, USA: Association for Computational Linguistics, 2001, pp. 1–8.

[6] Y. Gong and X. Liu, "Generic text summarization using relevance measure and latent semantic analysis," in *SIGIR '01: Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*. New York, NY, USA: ACM, 2001, pp. 19–25.

[7] R. F. Karlin, "Defining the semantics of verbal modifiers in the domain of cooking tasks," in *Proceedings of the 26th annual meeting on Association for Computational Linguistics*. Morristown, NJ, USA: Association for Computational Linguistics, 1988, pp. 61–67.