Extracting Menu Items from Yelp Reviews of Madison Area Restaurants

Lokananda Dhage, Mary Feng, Varun Naik

CS 838 Project Stage 2

1. Entity Type Extracted and Examples

   Our two data sources, Yelp and Zomato, both contain user written reviews of various restaurants in the Madison area. Restaurant reviews were a natural choice of text documents. For this project stage, we decided to use solely Yelp reviews for two reasons -- quantity and length. We have 61169 Yelp reviews and 1758 Zomato reviews. Yelp reviews also tend to be longer than Zomato reviews, which is beneficial to ensure there are enough mentions of the entity type to extract.

   After debating over various entity types to extract, we settled on extracting menu items from Yelp reviews. This was a reasonable choice since mentioning menu items is fairly common in a restaurant review. We defined a menu item as something that you could order while speaking to a host or hostess as a restaurant. However, this does not include ingredients in a dish. For example, if a review mentions a salad and its ingredients, we would consider “salad” as a menu item, but not individual ingredients such as tomatoes and lettuce. That is, we not extracting all mentions of food, just items people would order from a menu. Somewhat surprisingly, no Yelp reviews contained the characters “<” nor “>”, so menu items are simply marked up like <menu item>.

   Example of a marked up document:

   {"review_id":"hhm9zgzaLHTSnG7VVxq4zw","user_id":"ewtuS5t5Cj6ypmTnxAJaYg","business_id":"Dcj69NZJnbJhK-YHP_nULA","stars":4,"date":"2015-07-29","text":"The highlight was the **<Truffle Fries>** appetizer(enough for three), with the special dipping sauces. **<Avocado BLT>** paired nicely with the **<Sweet Corn Soup>** and the **<Leffe Blonde>**. All around excellent lunch experience.","useful":0,"funny":0,"cool":0,"type":"review"}

   Example of a marked up document:

   {"review_id":"CVIJC1475omOqEP0F_ajdw","user_id":"fgpD5-TswwW_x2ggNTq4BQ","business_id":"t1w-AqAHPL-apnKtoboodQ","stars":5,"date":"2015-01-06","text":"The best **<chai>** in Madison!! Aimee and Kelly were so inviting. The **<chocolate chip banana bread>** was delicious and homemade. The **<espresso>** had a velvety crema with a bright body and smooth finish. The vibe in the place is so inviting and chill. I love it and will definitely travel from Milwaukee again to visit the Froth House!","useful":1,"funny":0,"cool":0,"type":"review"}

2. Number of Mentions Marked Up

   Throughout the 300 reviews, we marked up 967 mentions. The 300 reviews were split into three sets of 100 reviews, and each team member marked up a set. We discussed

various cases in which menu items may be ambiguous before marking up reviews to help maintain consistency in labeling. No two positive examples overlap. There were 199 mentions in one set of 100 reviews, 388 in another set, and 380 in another set. Thus, we had 967 positive examples in the whole dataset.

We generated negative examples by randomly choosing different starting words within the review text, and giving each example a length from 1 through 6, chosen uniformly at random. Furthermore, we ensured that no two negative examples overlapped (although negative examples could overlap partially with positive examples). We also checked that each negative example fit within a single sentence. In the end, we generated 2245 negative examples in the entire dataset. The negative examples generated this way ensures that our classifier will be trained to correctly classify potentially a large variety of unseen negative examples.

3. Number of Documents and Mentions in Sets I, J
    - Dev set I: 200 documents, 661 mentions of menu items/positive examples
    - Test set J: 100 documents, 306 mentions of menu items/positive examples

Set	Number of documents (reviews)	Number of mentions of menu items (positive examples)	Number of negative examples	Number of examples (positive + negative)
Dev set I	200	661	1520	2181
Test set J	100	306	725	1031
Total	300	967	2245	3212

4. Type of Classifier Selected After Performing Cross Validation on Set I *the first time*'
   We initially extracted the following features from each positive example and negative example. For simplicity, we refer to the sentence containing the example as S:
    1. Number of characters in the example
    2. Whether the last word in the example ends with "s"
    3. Whether all words in the example are capitalized
    4. Whether the example contains a common cooking style as a substring ("baked", "fried", etc.)
    5. Whether the example contains a common food name as a substring ("pizza", "burger", etc.)
    6. Whether the example contains a word in the restaurant name as a substring
    7. The position of the first character of the example within S, scaled by the character length of S (first character in S corresponds to 0, last character corresponds to 1)

8. Whether S contains a common food-related adjective as a substring (“tasty”, “delicious”, etc.)
9. Whether the word before the example within S (assuming such a word exists) is the definite article, “the”

The five classifiers were decision tree, random forest, support vector machine (SVM), logistic regression, and linear regression. We performed 4-fold cross validation on set I for each classifier. For linear regression, we classified a prediction as either 1 (positive) or 0 (negative) by rounding to the closer value, and by rounding up 0.5 to 1.

Linear regression provided the highest precision (0.754), and recall 0.513. SVM provided precision 0.750, and the highest recall (0.626). We judged that SVM provided the best combination of precision and recall, and we selected it as our desired classifier moving forward.

5. Type of Classifier Finally Selected

SVM was selected as the final classifier as it had the highest precision and recall compared to the other classifiers. We added the following features:

10. Whether the definite article, “the”, is a word in the example
11. The number of commas in S containing the example
12. The number of words in S containing the example
13. The number of words in the example
14. Whether the first word in the review is capitalized
15. Whether any word in the review ends with “ing”
16. Whether any word in the review ends with “ed”
17. Whether any word in the review ends with “es”
18. Whether the word before the example within S (assuming such a word exists) is an definite article, “a” or “an”
19. Whether an indefinite article, “a” or “an”, is a word in the example
20. The fraction of words in the review that are capitalized (as a floating-point value from 0.0 to 1.0)
21. The position of the first word of the example within S, scaled by the word length of S (first word in S corresponds to 0, last word corresponds to 1)
22. Whether the example contains a digit “0”, “1”, …, “9”

After performing further cross-validation using SVM, we ignored Features 3, 7, 11, 12, 15, 16, 17, 18, 19, 21, 22, as they did not increase our precision any further. At this point, our precision was 0.850, and our recall was 0.832.

6. Rule-Based Post Processing

We used rule-based post processing to improve the precision to the required 90%. The rules were applied after the trained classifier was used to make predicted labels. The

rules can be found in debugM.py in the apply_rules() function. The two rules used are as follows:

1. *Check if the text of the instance contains a noun*

   Intuitively, the “menu items” that we are extracting should contain a noun. By examining false positives, we discovered that some of these example texts contained verbs or other words and no nouns. For example, one false positive example had the text “both enjoyed”. This phrase does not contain a noun. Therefore, this rule checks through all predicted labels as output from the trained classifier. If a predicted label is positive (predicting the instance is a menu item), this rule checks to see if the example text contains a noun. If a predicted label is positive yet the example text does not contain a noun, then the predicted label is changed to negative. The Natural Language Toolkit (NLTK) Python package was used to help determine whether the example text contains a noun. First, the example text is tokenized so that the example text is split into individual words. Then, NLTK’s pos_tag (parts of speech tag) function is used to attach a speech tag to each word. Then, we can simply go through all the speech tags to check whether the example text contains a noun, either singular (NN) or plural (NNS). This way we are are able to reduce the number of false positives.

2. *Check if the stop words occur at the beginning or at the end of the example text*

   Again, while examining false positives, we observed that some false positives contained a food item and a stop word. For example, one false positive was “noodles and”. Since menu items should not start nor end with a stop word, we enforced this rule. If the example text starts or ends with a stop word, its label is negative -- that is, it is not a menu item. For this, we used NLTK’s corpus of stop words.

7. <u>10 Fold Cross-Validation Precision, Recall, F1 on Dev Set I Including Rules</u>

   This is the train_metrics() function in debugM.py.
   training set metrics:
   mean 10 fold cv precision: 0.92302512765
   mean 10 fold cv recall: 0.741232182479
   mean 10 fold cv f1: 0.820977082119

8. <u>Precision, Recall, F1 of classifier Y on Test Set J</u>

   This is displayed by running debugM.py with dev_set.json test_set.json. This is obtained by training our SVM classifier on the dev set dev_set.json, followed by rule-based post processing.
   Test set metrics:
   Precision: 0.919028340081
   Recall: 0.741830065359
   F1: 0.820976491863

9. <u>Steps to replicate this procedure</u>

   Our code contains seeds for randomized processes, for repeatability purposes.
   1. join_reviews.py: add on restaurant and user info to each of the 300 labeled reviews

2. generate_feature_vectors.py
3. split_reviews_dev_test.py: randomizes order of 300 labeled reviews, splits into train & test sets
4. classifiers.py: compute precision/recall for various classifiers for comparison
5. debugM.py: contains various functions used for debugging, applying rules, and does the precision/recall/f1 calculations on the test set after training the classifier on the dev set and applying rule-based post processing

python join_reviews.py yelp_restaurants.json yelp_users.json labeled_reviews.json > joined_reviews.json
python generate_feature_vectors.py < joined_reviews.json > feature_vectors.json
python split_reviews_dev_test.py feature_vectors.json dev_set.json test_set.json
python classifiers.py dev_set.json
python debugM.py dev_set.json test_set.json