Suhan Liu & Stan Thornhill

CMPS 290H

04 June 2016

**Using DDLite to Find Pizza Types**

**ABSTRACT**

We found that some restaurants do not have menus on their Yelp webpages. Thus, we were thinking about whether we could do a research project which could help Yelp automatically generate menus. In this project, we focus on pizzerias for the experiments. We got data sets from yelp data challenge and extracted data using DDLite. It is a light version of DeepDive based on Python. We chose it because it was easy to get started. In this project, we have two main stages. The first one is the extraction of candidates and the second stage is the learning process.

# 

**DATA SOURCE**

Yelp has set up a competition called “Yelp Dataset Challenge” where we could download their data as JSON files. They contain business information, check-in information, reviews, etc. For our project, we only used two files, business information and reviews. Both of files are really big so we avoided opening them or importing them for several times. What we wanted are as many reviews from pizzerias as possible. After observations of the structures of those two files, we found that we needed to match “business\_id” for both files. The “business\_id” is the unique ID of each business unit, including restaurant. Also, to restrict the businesses to be pizzerias, we set the categories to contain both “Restaurant” and “Pizza”. At first, we extracted all reviews about pizzeria in one txt file. However, we found that we could not match reviews to each restaurant because all reviews were saved as a whole string. Thus, we decided to imported all reviews into separate files and each restaurant had one file to save its reviews. When we used “DocParser” function we found that it could not read too large file. Therefore, we decided to constrain every restaurant having at most 50 reviews.

**TOOLS**

DeepDive is a framework that processes unstructured data and outputs a knowledge base. From the unstructured data it extracts possible mentions, entities, and relations. Features are extracted that relate mentions to each other. Also rules are written that capture relations between mentions. A factor graph is created that forms a probabilistic knowledge base where entities, features, an input dictionary, and rules train the graph. The goal of DeepDive is to be able to make inferences in order to build a structured database. The queries such as "who is married to who" can be asked.

It's not a general purpose knowledge base meaning even though deepdive parsed through a lot of unstructured text, the user is unable to query it to find out for example if M. Obama is a male or female even though M. Obama is an entity and it has features about her and relations between her and other entities.

DDLite is different than DeepDive in that the user writes labeling functions to build a classifier to extract positive candidates. DDLite does not use supervision rules to build a factor graph. The user creates labeling functions that with the combination of a gold standard dataset creates a trained logistic regression classifier. The trained classifier is used to classify positive candidates. In theory, DDLite could be used to complement DeepDive. The positive candidates that the trained classifier outputs could be a "Programmatic Supervision" approach instead of "Distant Supervision" approach to serve as DeepDive’s input. It could be used as a substitute for DeepDive's training data if no External KB is able. Slides 39 - 42 of Theodoros Rekastinas's DeepDive presentation hints about having this "Programmatic Supervision" approach.

Also, we used Jupyter as our editor. It is also based on Python. Like a notebook, we could not only write documentations in Jupyter, but we could write Python codes and run them inside Jupyter. It is a locally driven software but could be opened in a browser so it’s quite user-friendly.

**EXTRACTION**

Our purpose is to extract pizza type names from the reviews. Our first intuition is to search the keywords in review texts. We first considered that if we just searched “pizza” then the results would contain a lot of redundant results because this keyword is too vague. Then we thought about some common types of pizzas, for example, pepperoni, sausage, vegetarian, etc. Nevertheless, we found that we could only find those exact words. The problem is that some are not pizza types, and we also missed a lot of pizza types. Then we got a new idea that we tried to use regular expressions for pizza types. Our first try is to restrict the expression to be no more than four words and ending with word “pizza”. However, it failed again. Over half of the results are not pizza types. Some of them are even not complete phrases, for instance, “and had a pizza”. Finally, after observation, we found some characteristics of pizza types. Most of pizza types would begin with a general topping and end with “pizza” Thus, we limited the regular expression of pizza type to be less than or equal to five words, starting with a topping name and ending with “pizza”. What’s more, we created a small dictionary for the general toppings for the regular expressions.

**LEARNING**

The next step in our project is to use DDLite to go through the learning and labeling function iteration. To create an accurate tagger for pizza types in yelp reviews we get candidates from the extraction phase, Create a test set, Write labeling functions that serve as features for our tagging model, and finally Learn the tagging model.

First, we created an Entities object from our set of candidates. The DDLite Entities class has an extract\_features() method that we use to generate generic features that DDLite offers. It creates lemmas, pos tags, and a dependency tree of the mention and the sentence that the mention is found in. This extraction process generated 27944 features for each of 1021 mentions.

Next, we hand labeled 198 candidates with truth values. We assigned a value of 1 for positive candidate or a value of -1 for negative candidate. Each candidate has a mention() method that prints out the words of the candidate that allowed us produce a candidate. After that, we assigned half of the gold data to a validation set for parameter tuning, and the other half to a test set. In addition to this approach to label the data, DDLite required us to build more ground truth items using MindTagger. MindTagger is GUI interface that highlighted each candidate in the sentence in it is found in. For each candidate, we performed a similar process to labeling the gold data, where we gave a response to yes if it is a mention is a pizza type, and no otherwise. We also had the option to abstain from labeling a particular candidate. This additional labeling was used for the development set.

After all the labeling was done, we wrote labeling functions. The following are labeling functions with short description:

**Positive LFs**

LF\_like(): Candidate is positive if it has the lemma "like" in the pre or post\_window.

LF\_love(): Candidate is positive if it has the lemma "love" in the pre or post\_window.

LF\_usually(): Candidate is positive if it has the lemma "usually" in the pre or post\_window.

LF\_favorite(): Candidate is positive if it has the lemma "favorite" in the pre or post\_window.

LF\_pepsau(): Candidate is positive if it has the words ["pepperoni", "sausage", "cheese"] in the mention.

LF\_order(): Candidate is positive if it has the words ["ordered", "had", "tried", "ate", "have", "has", "eat", "order"] in the mention.

**Negative LFs**

LF\_notLike(): Candidate is negative if it has "don't like" in pre or post window.

LF\_bad(): Candidate is negative if it has "bad" in pre or post window.

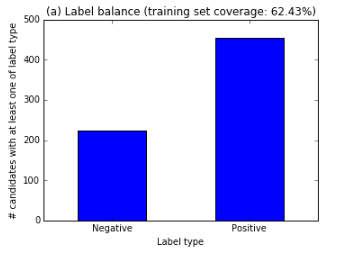
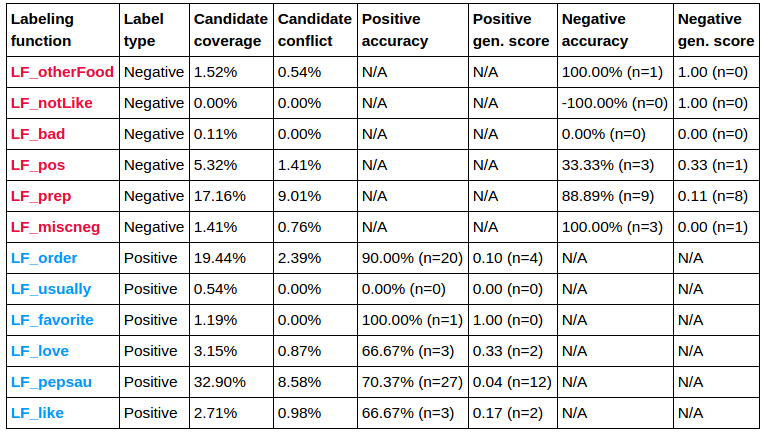
LF\_pos(): Candidate is negative if it has verb or adverb in the mention.

LF\_prep(): Candidate is negative if it has a preposition in the mention.

LF\_miscneg(): Candidate is negative if it has "or" in the mention.

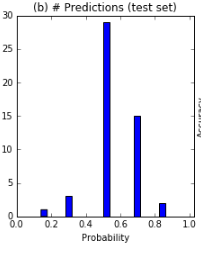
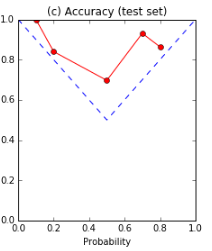
LF\_otherFood(): Candidate is negative if it has any the words ["burger", "salad", "sandwich", "wings", "beer"] in the mention.

Next, we applied the labeling functions to mentions and used DDlite's assessment utilities to debug and analyze our LFs before we perform inference. DDlite's assessment utilities allowed us to view the coverage, conflict, and accuracy on the development set. The following bar graph and table generated showed these statistics:



The table shows that our LFs had a wide range of accuracy, coverage, and generalization score range (some very low). The bar chart shows we have 62% percent coverage. In our initial run through the data, we started with 30% coverage, but as we increased the number of labeling functions, our coverage increased.

The next step we performed was to learn weights for the LFs. DDLite uses a logistic regression model with elastic net regularization. We used the same parameter values that the gene example used. The model trained on the training set, tuned on the validation set, and generated performance results on the test set. The following is a graph of the classification accuracy:



The model produced an F1 score of .86. This is interesting since most of the predictions were near 50% probability. The v-shaped calibration plot shows that if the classifier was more certain it was more accurate. Interestly, even when it was uncertain it was still more accurate than a coin flip.

Finally, we used our trained model to predict on all candidates. The model classified 136 as negative and 885 as positive. After that, we used python's counter tool to provide us with a rapid tally of the most frequent pizza types. The following are the most frequency pizza types:

(u'pepperoni and sausage pizza', 29),

(u'chicken pesto pizza', 25),

(u'pepperoni and mushroom pizza', 18),

(u'sausage on the pizza', 11),

(u'sausage and pepperoni pizza', 11),

(u'sausage on my pizza', 10),

(u'Italian style pizza', 10),

(u'bacon cheeseburger pizza', 9),

(u'sausage and mushroom pizza', 8),

(u'spinach and artichoke pizza', 6),

(u'pepperoni and cheese pizza', 6),

(u'Italian sausage pizza', 6),

(u'pepperoni and bacon pizza', 5),

(u'Chicken Alfredo pizza', 5),

(u'chicken and bacon pizza', 5)

**DRAWBACKS OF DDLITE**

A difficulty of DDLite was figuring out how to do basic things with a lack of documentation. If a large community of developers started using it, it would be a more pleasurable experience to program in it.

Another difficulty was the rapid change in functionality. We ended up getting familiar with two different versions of it which made switching between versions difficult.

MindTagger is a nice tool to label gold data. It was unclear why we had to label the training and validation set directly. It would have been nice if we could have labeled all data through MindTagger.

We found that it was difficult to know which labeling functions conflicted with what. If they developed a conflict matrix, it would be more clear where the conflict lies rather than just guess.

As mentioned earlier, DeepDive is not a general purpose knowledge base. A possible direction to making it that way would be to give functionality to link DeepDive graphs together similar somehow to how semantic web data is linked together with “sameas” links. Performing inference on a massively linked factor graph may not be traceable, but maybe it’s worth exploring.

**DIFFICULTIES**

We wanted to use Pizza Ontology as our pizza type reference. We tried the software called Protege but failed to load the owl file. Thus, we used our common sense to build a dictionary for pizza types.

When we first used JSON files, we did not know how to extract specific reviews, since simply extract all review did not make any sense. We found that the review.json file include all the review text. For each review, it has its corresponding business\_id. We noticed that business\_id was also in business\_infomation.json file, so we did more research on this file. This file includes business\_id, restaurant name, categories, etc. Then we found that each restaurant has a unique ID. However, simply matching business ID was not enough for extraction since we saved all review text in one file. We did not know if it was a restaurant, if the restaurant served pizza, and which restaurant the review came from. Then we found we could utilize categories to solve this problem. Categories are like tags of those business units. We restricted categories to contain “Restaurant” and “Pizza”. Also, we generated reviews of one restaurant to be saved in one file and put its business ID in the file name so that we could know which restaurant the review came from.

We got an error passing Unicode to DocParser function in DDLite. We had to convert to ASCII and chose to ignore any characters that didn't have an ASCII mapping. There's a single python line I threw in that does that.

Another difficulty was figuring out what we could reasonably accomplish with what we had. We initially thought about picking the best pizza types but would have required capturing sentiment which wasn't clear how to do that. Thus, we thought about using regular expressions to have the candidate beginning with a topping name and ending with “pizza”.

The biggest difficulty is finding the reasonable labeling functions. At first, we wrote some labeling functions but got low coverage at about 20 percent. That was not enough data for learning process. Thus, we tried a lot of labeling functions to figure out how to raise the coverage. Then the serious problem was that the uncertainty of the results. A large portion of results have the probability of 0.5 even though our accuracy was pretty good. We still need to find where the problem is.

**CONCLUSIONS**

We build a knowledge base about pizza types in this project and found the most popular pizza types from Yelp reviews. We learnt a lot about tools, information extraction, machine learning, and so on. However, we have met a lot of problems. For the future works, we may use some name entity recognition method. We may find some other ways to reduce the uncertainty. Also we could try to find a way to detect some special pizza types.