# Sentiment based Summarization of Restaurant Reviews
# Final Project Report
# CS 224N

Abhishek Gupta, Tejaswi Tenneti, Ankit Gupta

June 3, 2009

## 1 Introduction

Over the last decade, we have moved from scarcity of information to an abundance of information. Due to the plethora of restaurant choices today, restaurant selection has become really difficult. Although Yelp allows us to read reviews and understand the qualities of a restaurant ourselves, it is really hard to get a holistic view of the restaurant without reading all the reviews. This is due to several issues as follows

- Currently, Yelp assigns an absolute rating for every restaurant. There is no way to distinguish between a restaurant with great food and below average ambience from a restaurant with the exact opposite with such a measure. This lack of fine-grained reviews and ratings in individual categories (e.g. food, ambience, and service) hinders easy comparison between restaurants and quick decision making. Further, it is inconvenient for a user to rate a restaurant on all these metrics making the process cumbersome.
- Reading all the reviews and collecting the pros and cons so as to decide whether a restaurant is good or not is really time consuming. While reading recommendations, what one is really looking for is the degree of adjectives (positive e.g. good food, comparative e.g. better than X, and superlative e.g. best food) describing the restaurant.
- Since several people add reviews for a restaurant on Yelp, it is hard to extract just the significant aspects of a restaurant. For example, while talking about an excellent *mushroom and olives pizza* at *Pizza, My Heart* in *Palo Alto*, reviewers also delve into several other recommendations in addition to everything they did until that point of the day. Even if you read all the reviews, the goodness of the *mushroom and olives pizza* fades due to all the noise.
- While reading reviews, one maintains a mental model of the restaurant for further comparison with other restaurants. This is a highly inefficient process and an unnecessary investment of energy and time.

Consider an alternate scenario in which you can read a summary of any restaurant with its pros and cons and a rating for each of the categories *Food*, *Ambience* and *Service*. You can keep these short summaries side by side and efficiently compare two restaurants in a matter of seconds!

In this project, we aim to generate significant attributes of a restaurant in the 3 categories : *Food*, *Ambience* and *Service*. We then attach descriptors from the reviews of the restaurant to these attributes. Using our sentiment classifier, we classify the descriptors into pros and cons, thus generating a visually appealing and quickly readable summary of the restaurant.

## 2 Background

Our work is closely related to M. Hu et al's [1] work on mining and summarizing customer reviews. Their system identifies product features and user opinions in a product review to automatically produce a summary of the review. After determining the most important features in a product, their system looks for opinion words occurring in close proximity to the feature and uses WordNet to determine the semantic orientation of the opinion word. They use this information to generate a summary of the review. Patrick et al [2] have also attempted summarization of restaurant reviews by determining high-information snippets inside a review.

Our system is a more fine-grained approach than the above work in the sense that we classify each of our frequent features into three classes viz. *food*, *service* and *ambience*. Our focus is on extracting sentiments about these three categories in a set of reviews of a restaurant rather than summarizing the reviews.

Other related work in this area include I Titov et al [3] who have proposed a Multi-Aspect Sentiment model to build topics that are representative of ratable aspects, and a set of sentiment predictors for each aspect. In sentiment extraction and classification, Turney [4] uses Pointwise Mutual Information (PMI) for review polarity classification. He queries the ALTAVISTA engine, which has a NEAR operator to restrict the search between the keywords to a fixed proximity, for determine PMI.

# 3 Data

## 3.1 Yelp Data Extraction and Parsing

We used Jobo [5], an open source Java based web crawler, for scraping data from Yelp. We collected data of 120 Pizza places. We then used HTML Parser [6], an Open Source Java based HTML parser, for filtering out all the reviews from all the restaurants.

## 3.2 POS Tag and Dependancy Tree Generation

Most of the features of a restaurant generally occur in the reviews as Nouns and Noun Phrases. The opinions about these features are adjectives that describe these features. We use the Stanford POS [7] tagger to identify Nouns and Adjectives in the sentences in the reviews of all the restaurants. The Stanford POS tagger tags Nouns and Noun Phrases as NN, NNP, NNS and adjectives as JJ. Furthermore, a sentence could contain more than one Noun or Noun Phrases (features) and Adjectives (opinions). Thus, we need to determine the Noun that a particular Adjective modifies i.e. the feature about which a certain opinion has been expressed. For this purpose we used the Stanford Parser to generate the parse tree of a sentence and extract "Typed dependencies" among the words of a sentence. The typed dependencies provide a description of grammatical relationships between the words of sentence. We use two of these relations viz. *amod* and *nsubj* to determine the Noun that an adjective modifies. *amod*, short for adjectival modifier is any adjectival phrase that serves to modify the meaning of the NP. For example, in the sentence *Sam eats red meat* the Noun *meat* is modified by the adjective *red* and their relationship is expressed as *amod(meat, red)*. *nsubj*, short for nominal subject is a noun phrase which is the syntactic subject of a clause. For example, in then sentence *The baby is cute* the relation between *cute* and *baby* is expressed as *nsubj(cute, baby)*. Note that the relation *nsubj* does not always express a relation between an NN and an adjective modifying it. Therefore, when determining a feature and opinion pair from the dependency relation we use the POS tag of the opinion to confirm that it is an adjective and then use the above dependency relations to determine the Noun which is being modified by this adjective.

# 4 Our Approach

We start with identifying significant features of all restaurants. We then classify each feature into three categories *Food*, *Ambience* and *Service*. From the reviews of a restaurant, we then extract all the adjectives that describe the features of the restaurant. We then determine the sentiment of each descriptor of the features of the restaurants. Finally, we form three separate tag clouds for each class of features, thus, visually summarizing the opinions about each restaurant.

## 4.1 Feature Generation

Features are a set of words representing a particular quality of a restaurant. We assume that features can only be singular, plural or common nouns (NN, NNS). Thus, we consider all the words with POS tags NN or NNS in our feature set. We also include those words with the POS tag NNP which, when converted to lowercase, occur as either a word labeled NN or NNS in the rest of the reviews. We completely ignore those NNPs(proper nouns) which can never occur as a common noun(singular or plural) as they are usually names like John, Steve etc. which are not representative of a restaurant's characteristic.

#### 4.1.1 Frequent Feature Extraction

For our task, we require features that represent a restaurant. Since several words occur as nouns in a restaurant's reviews, intuitively, those that occur quite frequently in the dataset would be highly informative and thus, representative of the qualities of the restaurant. We use the Apriori Algorithm for computing frequent itemsets, and thus frequently occurring word combinations describing a restaurant. As an input to the algorithm, we consider each sentence as a basket and all the features (as identified above) as the itemset present in it. We use the code from [8] in our implementation.

The frequent itemsets generated by this algorithm can't be directly used as features due to the following problems :

- The Apriori Algorithm computes sets of frequent items. Thus, there is no notion of ordering between the words. For eg, the algorithm denoted the set

$$[Alto, Palo, pizza]$$

  as a frequent itemset. But, obviously, the correct feature would be ordered as

$$[Palo, Alto, pizza]$$

- All features in a sentence are put in the same basket for running the Apriori Algorithm. This discards all information about the position of these features in the sentence. Thus, even if a frequent itemsets is such that all its words occur in the sentence of a review, it may happen that the words are very far apart, and thus, don't form a coherent and meaningful feature for the restaurant.
- Due to the nature of the way frequent itemsets are computed, all the words in a frequent itemset containing more than one word would individually be considered frequent as well. In some cases, the existence of all or some of the words independently as features might make sense e.g. *french wine* as well as *wine* may co-exist together as meaningful features since the word *wine* alone occurs sufficiently frequently. In some cases, the existence of some or all of the words independently as features might not make sense at all e.g. *pepperonni pizza* and *pepperoni*. Since whenever *pepperoni* comes, it either comes along with *topping*, *toppings* or *pizza*. Hence, we can conclude that as a standalone feature *pepperoni* is not useful. It is important to identify such features and eliminate them so as to make the summaries more succinct.

To solve the problems listed above, we used two filtering techniques. To achieve desired filtering we required a total order on all the sentences in the dataset. For our current *support threshold* (a parameter in the Apriori Algorithm), the algorithm gives us a list of all one itemsets, two itemsets, three itemsets and no four or higher size itemsets. We ordered all the sentences in all the reviews of all the restaurants, assigning a unique id to every sentence.

1. **Ordering and Proximity Filtering** : For each frequent itemset, we maintain a *Set* of sentence identifiers. For every itemset in the two itemsets list, we consider only those sentences which contains its words within a distance of atmost 2 word. Further, if within these sentences we encounter more cases of second word of the itemset occuring before the first word of the itemset then we consider, *secondword :firstword* as the feature otherwise we consider *firstword :secondword* as the feature.

   For a three itemset, let us denote *'a'*, *'b'*, *'c'* as the first, second and third element of the itemset. We consider only those sentences that either contains *a,b* and *b,c* both within a distance of two words of each other or contain *a,c* and *c,b* both within a distance of two words of each other or contain *b,a* and *a,c* both within a distance of two words of each other. Further, if within these sentences we encounter more cases of a particular ordering of *'a'*, *'b'* and *'c'*, for example in most cases *'b'*,*'c'* and then *'a'* occurs, then we consider *b :c :a* as the feature for this itemset.
2. **Standalone Identity Filtering** : As described above, it is important to remove those features which occur mostly with their supersets. These features by themselves don't mean much and are more useful if considered with their supersets which are frequent. For instance in our entire corpus *pepperoni* mostly comes with either *topping*, *toppings* or *pizza* and hence has no significant existence of its own. For every feature, we compute the number of cases where it occurs but none of its supersets occur. If this number is below a certain threshold, we remove this feature. For features with one, two and three words we use a threshold of 30, 25 and 20 respectively. For all features with one, two and three words we maintain a separate *HashMap* with key as the feature (with spaces between words replaced by colon) and value as a *HashSet* containing a set of sentence identifiers.

For features with three words, since we don't have any four itemsets, there is no superset of this feature and the count of this feature after removing count of all its supersets remains the same. For every feature with two words, we first compute the union of sets of sentence identifiers of all supersets of this feature. We then subtract this *Set* from the *Set* of sentences in which this feature occurs. This results in a *Set* where only this feature occurs and none of its superset occurs. The size of this *Set* gives us the number of sentences after filtering. We do the same for every feature containing only one word. For the features with only one word, we now consider only those features as final features which occur more than the a certain threshold number of times. We now threshold the occurrence of all features discarding those with small counts.

#### 4.1.2 Infrequent Feature Extraction

Although the frequently occurring features give us a good idea of representative features in the dataset, several features present in a restaurant review aren't captured in this fashion. For example, a review for *Applewood Pizza* in Los Altos contains this sentence about their specials : *They have great lunch specials.* Obviously, this is a very important feature for this restaurant. But, since *specials* doesn't occur very frequently all the reviews, it isn't captured in the frequent itemsets. To extract such infrequent but highly informative features, we extended our feature extraction technique.

We start with a list of all descriptors of the frequently occurring features in the dataset. For each sentence in a review, we then look for all words that occur in this list. For each descriptor instance, we use the dependancy tree and locate the words that it is related to via two relations : *nsubj* and *amod*. If any of these relations exist, we then validate if the other word is a noun (having POS tag *NN* or *NNS*). All such words that aren't present in the original frequent itemset are labeled as infrequent feature and added to the feature set. Since these features are highly informative, it is hard to understand their significance for the restaurant without knowing the context in which they were used. Thus, we also extract a snippet of text around the feature and its descriptor.

### 4.2 Feature Classification

#### 4.2.1 Method

As mentioned at the start of this section, one of our main tasks was to classify the features of a restaurant into three categories : *Food*, *Ambience* and *Service*. For eg, the mention of *drinks* in a review would describe the quality of *Food*, *wait* would describe *Service* and *place* would describe the *Ambience* of the restaurant. It is impossible to carry out this classification task by simply considering the word by itself and trying to assign a label to it. There are two sources of information that can be utilized in this situation : (1) context of the word in the sentence and (2) synonyms, hypernyms, description and other linguistic forms associated with the word.

Our approach utilizes the second kind of information using Wordnet (accessed via java api [9]). Wordnet allows us to look at synonyms and hypernyms of a particular query word thus giving us information about the location of the word in the overall Wordnet tree and the edge distance between two words in this tree. Consider the hypernyms (till we reach the hypernym *entity*) of particular synonyms of

- **pizza** : aliment, alimentation, dish, food, matter, nourishment, nutrient, nutriment, nutrition, physical entity, pizza, pizza pie, substance, sustenance, victuals
- **drinks** : abstract entity, abstraction, beverage, component, component part, constituent, drink, drinkable, fluid, food, liquid, matter, nutrient, part, physical entity, portion, potable, relation, substance
- **wait** : abstract entity, abstraction, amount, break, delay, hold, intermission, interruption, interval, measure, pause, postponement, quantity, suspension, time interval, time lag, wait

We can immediately observe that there is significant overlap in the hypernyms between *pizza* and *drinks* (food, nutrient, physical entity, substance etc.) whereas there is little overlap between them and *wait*. Building upon this intuition we implemented a Multinomial Probability Distribution Model over the hypernyms of a word as its features. Following steps summarize our classification method.

1. **Training Data** : We hand annotated 400 words from the reviews in our dataset with one of three labels : *food*, *ambience* and *service*. We used these word, label pairs to train our Multinomial Model.
2. **Features** : Since we don't use any context information for the words in a review, we use Wordnet to compute all its noun synsets (synonym sets). For each synset, we compute its hypernyms. We then

compute the hypernyms of these hypernyms and continue doing this till we reach the hypernym *entity*. Since *entity* is a parent of all words in the Wordnet tree, this gives us all possible hypernyms of a word. We use these as features in our model.

3. **Training** : We use Maximum Likelihood Estimation to compute the probabilities of any word being in a particular class. We also include an *UNKNOWN* tag to handle unknowns and use Absolute Discounting to smooth the counts.
4. **Testing** : While testing and classifying words in a review, we use the same approach as listed above and calculate all hypernyms of the word. Probability of the word being in a class is calculated as the multiplication of the probability of all its hypernyms being in that class. The class with the highest probability is then assigned to the word. If Wordnet doesn't contain synsets for any word, no class is assigned to it.

#### 4.2.2 Discussion

Described above is the final algorithm that we used for classifying words into their labels. In this section, we would present a short discussion on some of the other methods that we tried along the way that lead to the current algorithm.

1. In our first approach, we observed that several words having the same label also led to a group of common hypernyms when generating all possible hypernyms from them. Thus, we manually populated a group of target words for each class such that any word having a hypernym in a particular target set is classified to bear the label of that set. Our target set for the classes were :
   - **Food** : food, crust, topping, drink, rotisserie, slice, container, taste, fruit, foodstuff, ingredient, condiment
   - **Ambience** : ambience, parking, space, crowd, environment, toilet, interior, surrounding, pane, material, furnishing, furniture, area, structure
   - **Service** : activity, waiter, staff, manner, employee, worker, person, period, interval

   Intuitively, all food related words were in the target set of *Food*, all place and decor related words were in the target set of *Ambience* and all staff and person oriented words were in the target set of *Service*. Although this simple approach worked very well for many words, it failed for many others. There also wasn't any good method to determine whether the addition of a word in the target set of a particular class will improve or decrease classification accuracy. Another problem we encountered was that some words had hypernyms in more than one of these target sets. Hence, it was hard to determine which of them is the actual label of the word.
2. We then started using the hypernyms as features and used Maximum Likelihood Estimation to learn the probability of a feature being in a particular class. The hypernyms were calculated till a certain depth going upwards in a tree. We had set the depth to 5. The probability of a word ($W$) being in a class ($C$) was calculated as the number of words labeled as $C$ having $W$ as its hypernym divided by the total count for the class $C$. While classifying a word, we then calculated all its hypernyms and computed the maximum over the probabilities of the hypernyms in a particular class. The word was classified into the class having maximum score.

   This didn't work very well. Due to the fixed depth, several words labeled as *Service* and *Ambience* reached more general hypernyms such as *physical entity* ; but very few words labeled as *Food* reached such hypernyms. This assigned a high and discriminative probability to *physical entity* for *Service* and *Ambience* classes. Since we were taking the maximum over the probabilities of hypernyms of a word, several food related words were also classified into *Service* or *Ambience* simply by the virtue of the fact that they included the more general hypernyms in their feature set. We then experimented with different depths for different classes. That didn't work well either.
3. We then changed our model to a completely sound Multinomial Probability Distribution by estimating the probability of a word being in a class by multiplying the probabilities of its features being in the class and multiplying the prior probability of the class. This worked very well in general for *Food*. This didn't work well for *Service* or *Ambience* because our training data was heavily biased towards food related words (about 75% of the words had the label food). Thus, by multiplying with the prior probability, several words that shouldn't have been classified as *Food* were done so now. To decrease this bias, we also tried assigning prior probabilities from personal experience. This didn't improve the

results. Thus, we decided to stop multiplying with the prior due to lack of a representative training data.

As in the previous case, this method also didn't give us a good measure of the depth till hypernyms should be considered as features for a word.

4. Once we were training a Multinomial Model, we realized that we don't need to fix the depth till we consider hypernyms. Since all words had *entity* as a parent in Wordnet, we considered all hypernyms till we reached *entity*. This helped the results improve significantly.

   This approach also meant that, since most words do end up having the highly general words such as *entity* as hypernyms, the probabilities of these words reflected the prior bias in the training data. There were several such general hypernyms in the feature set. To prevent this, we started applying weights to the counts incremented during training. The farther away a hypernym is to the word in the training set the lesser (exponentially by a factor of 0.8) probability mass does it get. This helped improve the results as well.

5. When dealing with features consisting of more than one words, we independently classified individual words. We observed that if a word that describes a *Service* or *Ambience* comes with a word describing *Food* (eg. *pizza place*, *pizza delivery*), the feature most probably describes either *Service* or *Ambience* and never *Food*. Using this intuition, we assigned the classified label of a word as the label of the whole feature if it was *Service* or *Ambience*. Otherwise, if all words were classified as *Food*, we classified the feature as *Food*.

## 4.3 Descriptor Extraction

We have described the usage of Stanford parser and POS tagger in section 3.2. However, now our features are not all Nouns or Noun Phrases in a review, but are those determined in section 4.1. We now iterate over the review sentences of each restaurant and check for opinion words (adjectives) in a sentence. We then check the dependency relations in which these opinion words occur as one of the two nodes. We check whether relations are either 'amod' or 'nsubj'. If so, we extract the other node (the word that co-occurs with this opinion word in this dependency relation) and check if this word is among the set features determined for this restaurant. If it does, then the opinion word is added as a descriptor for this feature. For example : in the sentence *The pizza was delicious*, the feature *pizza* is being modified by the adjective delicious. This can be identified by the dependency relation *nsubj(delicious, pizza)*. If *pizza* is among the features of this restaurant then *delicious* is added as a descriptor for the feature *pizza*.

## 4.4 Sentiment Classification

In this step we wish to determine the semantic orientation of each descriptor for the features of a restaurant. The semantic orientation can be positive (indicating good opinion about the feature), negative (indicating bad opinion about the feature) or neutral (indicating that the opinion can be either good or bad based on the context used, or non-committal). We make use of WordNet to determine the semantic orientation of an opinion. But, WordNet itself does not give out the semantic orientation of any word. Moreover, we contend that the semantic orientation of any opinion word is subjective, and thus, we needed to incorporate domain knowledge into the system for it to correctly classify the orientation. For example, 'long' and 'large' are two words that appear to have the same orientation. But the sentence "It was a long wait before the pizza arrived" expresses a negative opinion about the wait time and the sentence "The large slices were very filling" expresses a positive opinion about the slices being large (and hence filling). In order to achieve this we populated seed lists of positive, negative and neutral words which were hand-picked from our corpus. Although there were many standard adjective lists available on the internet, several opinion words that occurred in our problem weren't included in them. So, we handpicked opinion words to populate the seed lists. The most interesting classes were service and ambience where we encountered cases like "long wait", "friendly staff", "cute place", "convenient parking".

The above examples show the importance of populating the seed lists with the right words so that the system can unambiguously capture the right sentiment. Our seed list consists of 150 positive words, 80 negative words and 39 neutral words. For a given opinion word we use WordNet to generate its Adjectevial Synonyms sets (synonym sets where the word is used as an adjective satelite), Adjectevial Satellite Synonyms sets (synonym sets where the word is used as an adjective), Adverbial Synonym sets, set of similar words,

set of related concepts and Antonyms. We then check to see if the opinion word or its above generated variants occur in one of the seed lists. If they do, we then classify the words according to class of the seed list that it occurs in. Otherwise, the opinion remains unclassified. This method has given us very high accuracy in opinion classification ; the only flip side being the need for careful handpicking of adjective seed lists. Moreover, we observed that most opinion words that were unclassified were words like "Italian", "second" and so on which trickled in as adjectives in sentences like "I had an Italian pizza" and "I asked for second helping".

# 5 Results

## 5.1 Feature Generation

We used a *support threshold* of 0.0004 for the apriori algorithm ; which means that if an itemset appears more than 0.04% of the time, it will be considered as frequent. Although this *support threshold* is very low, any *support threshold* higher than this value resulted in very low number of three itemsets. Further, after applying the two filtering steps mentioned in section 4.1.1, we didn't want to miss a potential useful but less frequent feature. Any threshold lower than this resulted in some four itemsets as being frequent. We feel that it is highly unlikely for a sentence to contain four nearby NN or NNS that might be representative of a review.

With a *threshold* lower than 0.0004 for the frequent itemset algorithm, we got the following items :

- **Useful** : pineapple, bake, strawberry, hummus.
- **Unimportant** : em, chez, anyhow, knowbut, ho dat, th, imo, put, yea, meh, itbut, bc, char, sans, tatts, ol, ur, hd, aka, pun, knots, bed, left, thurs, hi, form, section, test, st.

It is apparent that by decreasing the *threshold*, we get several unimportant features. Although some of these may be highly useful (such as those described in section 4.1.2), we aim to capture these using our Infrequent Feature Generation Method. Due to the significant increase in the number of useless feature,s as opposed to useful features with decrease in the *threshold*, we decided to against it.

The resuls with a *threshold* higher than 0.0004 for the frequent itemset algorithm are as follows :

- **Useful features that we miss** : worth, sense, art, crispy, interior, consistency, crap, metal, crap, comfort, chewy, cheesecake, patio, traffic, complaints, freshness, aroma, classic, special, entertainment, tax, parties, signature, noise, charm, tortilla, olive, coffee, environment, juicy, ladies, appetizers, crunchy, attitude.
- **Unimportant features that we miss** : son, back, cuase, use, fall, will, can, so, hey, hubby, plan, step, really, ten, level, vu, ive, thats, lil, pa, cus, note, im, mine.

Clearly, we miss a lot of useful features with a higher threshold. Since, we apply post-filtering on these itemsets, we don't want to loose out on a reasonably frequent itemset that might satisfy our filtering criteria. A large fraction of the unimportant features don't satisfy our filtering criteria (section 4.1.1). Thus, we didn't increase the *threshold*.

The thresholds for the *standalone identity* (section 4.1.1) filtering criteria were also decided based on the empricial observation. From the three thresholds, we used the highest threshold for features with one word and the lowest threshold for features with three words.

## 5.2 Feature Classification

Figure 1 shows a snippet of the training data we use and Figure 3 shows a snapshot of the results after classification. In this section, we will briefly describe the intermediate results we got in the process.

Figure 2 shows a snippet of the results with the first approach where we used a static list of words and didn't train on the data. We can see that *bread* has been labeled wrongly as *service*. This is because one of the meanings of *bread* in Wordnet is *Staff of Life*. Since, *staff* is in the list of words associated with *Service*, *bread* gets classified as service as well. We can see that this problem goes away and *bread* is correctly classified as *Food* in Figure 3.

While using the Maximum Likelihood Estimation Model, we initially fixed the depth of the hypernyms considered in the feature set going upwards in the tree as 5. This also resulted in *bread* being classifier as service. This was because, one of the words in its feature set is *abstract entity*. The maximum likelihood

```
ingredients:food
dang:food
piece:food
minutes:service
pizzas:food
ownership:service
minutes:service
sauce:food
```

FIGURE 1 – Training Data

```
[places] : ambience
[experience] :
[garlic] : food
[bread] : service
[atmosphere] : ambience
```

FIGURE 2 – Classification without Multinomial Model

```
[staff] : service
[salad] : food
[lunch] : food
[quality] : ambience
[soda] : food
[taste] : ambience
[flavor] : ambience
[places] : ambience
[experience] : service
[garlic] : food
[bread] : food
[atmosphere] : ambience
```

FIGURE 3 – Classification using Multinomial Model

probabilities of *abstract entity* being in *Service* is 0.0398 whereas the maximum likelihood probabilities of *abstract entity* being in *Food* is 0.000274. This discrepancy is due to the fact that for most words labeled *Food* in the training set, very few reached *abstract entity* at a distance of 5 ; but several labeled *Service* reached it. Due to this observation, we removed the limit on the maximum depth and considered all hypernyms till we reach *entity*.

### 5.3 Sentiment Classification

We have tested the accuracy of our sentiment classifier module on a set of 150 adjectives out of all those present in our data. We processed the results to calculate precision and recall for each semantic orientation. We obtained a precision of 100% and a recall of 98% for Positive adjectives. We obtained a precision of 100% and a recall of 95% for Negative adjectives. We obtained a precision of 96% and a recall of 100% on neutral words. Our system performed quite well in identifying prominent opinions about the features of each category present in reviews.

### 5.4 Tag Cloud Generation

To visually see the results, we use a standard tag cloud representation (Figure 4). We split the page in 3 panes for the 3 categories. For each category, we show positive features (as *opinion feature*) in green, negative features in red, neutral features in blue and all others in black. The size of the text for each feature represents its count in the review. We also extract snippets of text around the infrequent features that are shown below the tag cloud. We used the Open Cloud API [10] to generate the tag cloud. Figure 4 shows the summary for *Pinks Pizza* in *Houston*.


"This has to be my favorite pizza joint in town - and I"

"of all the menus of the local pizza joints on the interwebs."

"through the rough waves for the perfect Chinese spices and dumplings."

"(especially the cheese which is a fine mix of cheese of a lovely"

"way and balanced nicely with the fresh vegetables and the tang of the"

" Look out Star Pizza there's a new gourmet in town."


"and it's nice to have a non-chain option for pizza delivery."

" They have a few stools to sit on and watch"

"noticed atop their space in a small strip center at Heights Blvd."


" They honestly have some of the best combinations of toppings I've ever seen"

FIGURE 4 – Restaurant Summary

# 6 Future Work

There are several avenues of improvement to better the results of our method. In this section, we describe several problems that can be attacked to extend this project.

1. Currently, we used simple heuristics to compute boundary between various sentences in the review. This resulted in several problems due to issues like missing punctuation, slang language, various punctuation etc. Several methods can be applied to solve this problem. One approach can be to split the sentence at various words and use the parse trees of the two sentences to recognize whether the sentence is a combination of two sentences or not. This is a standard sentence boundary detection problem.
2. We have used reviews from only 120 restaurants in this project. This is a very small corpus for tackling this problem. If we increase the corpus size, we would be able to extract several features that would be frequent on the increased corpora but, unfortunately aren't in the current small one.
3. To train the classifier for classifying words into the 3 categories, we have only used its hypernyms as features. We haven't used any context information from the sentence. We could train a HMM with several such features to improve classification accuracy.
4. Our current training set for classifying features into the 3 categories was very small and biased. We need to increase and improve the training set for better accuracy.
5. Gleaning through our final results we feel that it would very useful to populate separate adjective seed lists for each of the categories (food, service, and ambience). This is because a "late delivery" indicates a negative opinion about the service while "late night" possibly indicates a positive opinion about the ambience ("the place is open till late night"). We have also observed that although it causes no harm, our seed lists are over-populated, especially the positive seed list. We feel that we could obtain the same results with a smaller seed list. However, as explained in a previous section we need to cover all kinds of opinion senses that might occur with each category.
6. Another important issue we could not explore is incorporation of context information around an opinion word. This is useful for two reasons : 1) handling of negative opinions expressed with a positive adjective along with a negative word contrasting the positive opinion. For example : "The pizza wasn't so good". One approach to incorporating this could be searching for contrast words like "not, no, n't" around an opinion. 2) Consider the phrases "additional wait but a delicious pizza", "made the pizza unbearably spicy by putting additional sauce". In the former sentence we can see how the word 'additional' which is generally used as a positive adjective ('additional toppings') has been used to express a negative opinion. However, had we used context information to capture the sentiment in the entire sentence, we would have been able see that "delicious pizza" is a positive opinion and 'but' contrasts the previous opinion with this one. Since "delicious pizza" can be easily classified as a positive opinion, 'but' can be used transfer this positive opinion as a negative opinion for "additional toppings". Same holds true for the second sentence where the phrase "unbearably spicy" can be used to infer a negative opinion about the sentence.

# Références


[1] M. Hu and B. Liu, Mining and summarizing customer reviews, in *KDD '04 : Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 168–177, New York, NY, USA, 2004, ACM.

[2] M. M. Patrick Nguyen and G. Zweig, Summarization of multiple user reviews in the restaurant domain, Technical Report (MSR-TR-2007-126), September 2007.

[3] I. Titov and R. McDonald, (2008).

[4] P. D. Turney, Thumbs up or thumbs down ? : semantic orientation applied to unsupervised classification of reviews, in *ACL '02 : Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, pp. 417–424, Morristown, NJ, USA, 2001, Association for Computational Linguistics.

[5] Jobo, http ://www.matuschek.net/jobo/.

[6] Htmlparser, http ://htmlparser.sourceforge.net/.

[7] The stanford parser : A statistical parser, http ://nlp.stanford.edu/software/lex-parser.shtml.

[8] Frequent itemset mining implementations repository, http ://fimi.cs.helsinki.fi/src/.

[9] Java api for wordnet searching (jaws), http ://lyle.smu.edu/ tspell/jaws/index.html.

[10] Opencloud, http ://opencloud.sourceforge.net/.