Summarization Miner & Sentiment Classifier on Yelp Reviews

Iqra Shahid
Dept. of Computer Science
New York University
is1293@nyu.edu

Rachit Mehrotra
Dept. of Computer Science
New York University
rachit.mehrotra@nyu.edu

Shubham Jain
Dept. of Computer Science
New York University
Shubham.jain@nyu.edu

Abstract—

We made an automated system that uses machine learning and natural language processing to generate a human-understandable, and browsable summary of the opinions expressed in a corpus of Yelp reviews (taken as input) about a particular restaurant. Applying aspect-based opinion mining, sentiment summarization/aggregation is done to generate a summary that aims to provide the user with an at-a-glance understanding of a restaurant's features or *aspects* (fries, burgers, sweet potato) as well as reviewers' attitudes towards these features. Our project takes a two-stage supervised modeling approach for opinion/sentiment classification, first attempting to classify a given sentence as Opinionated vs. Not Opinionated, and only subsequently attempting to classify Opinionated sentences as being Positive or Negative. The Opinion Model is an SVM-Support Vector Machine, and the Sentiment Model is a Logistic Regression. Both models are trained using an extremely lightweight feature set (only 16 total features) that includes primarily lexicon- and syntactic-features (e.g. number/fraction of positive/negative words, number/fraction of various parts of speech). These models are then checked for their performance (precision, recall, F-1) on cross validation. The system returns aspects of a restaurant selected. On selection of an aspect, returns an aggregate sentiment score for that aspect and some textual evidence to support it.

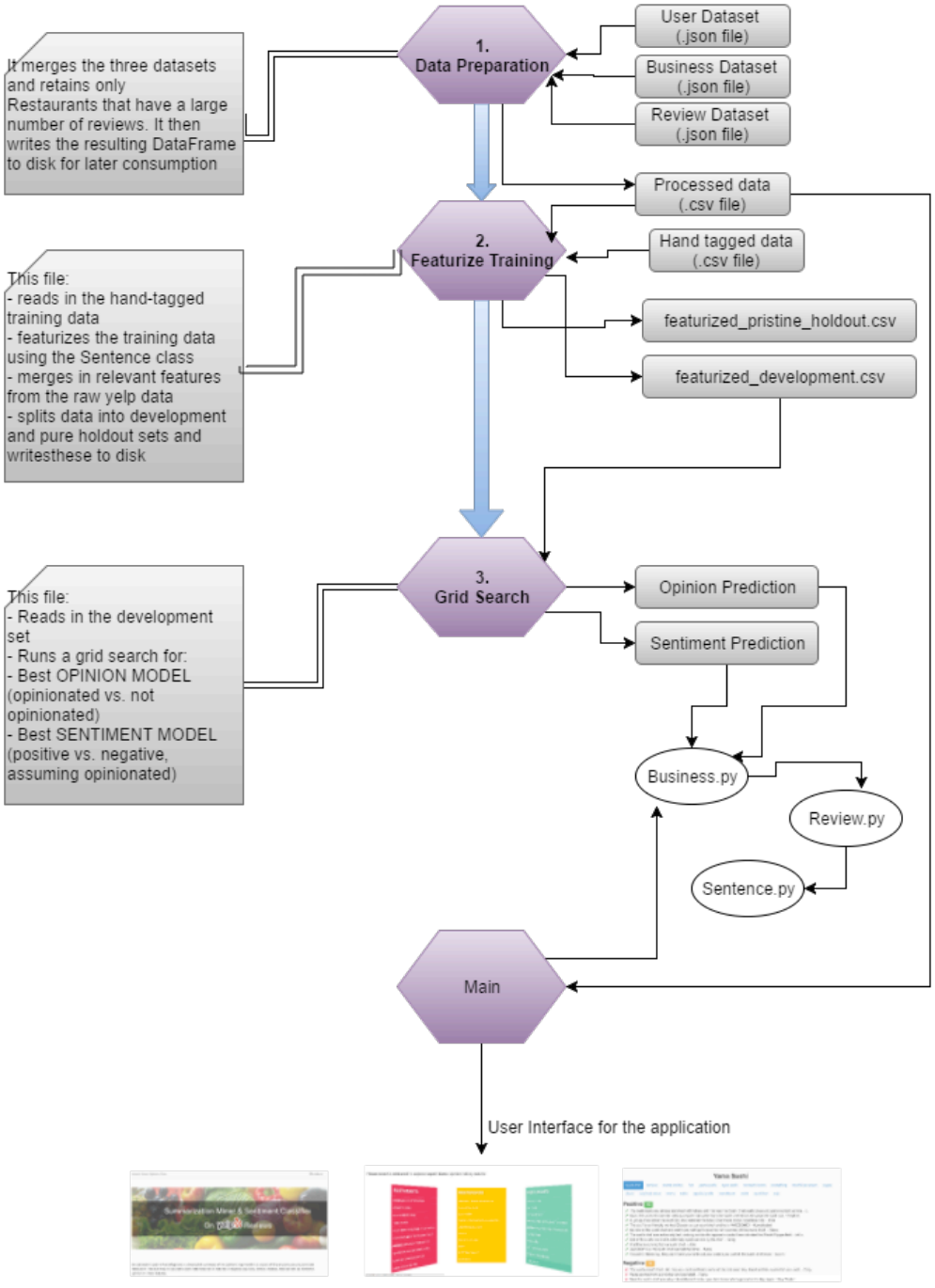
Keywords— summary, Yelp, aspects, opinion, reviews, sentiment, mining, classifier, SVM, Regression

I. INTRODUCTION

Restaurant selection has become really difficult because of the number of restaurants now. Yelp provides us with the restaurant reviews to understand the qualities of the restaurants but we cannot get a whole picture of the restaurant without reading all of its reviews. Yelp only tells us the absolute rating of a restaurant and lacks to provide us with fine-grained reviews and ratings in individual categories (e.g. food, ambience, and service). This is an obstacle in comparing restaurants and making a quick decision. Neither can we sit and read all reviews listing all the pros and cons of a restaurant as this would be too time-consuming, neither can a user rate a restaurant on all the metrics as that would be too cumbersome. What one is really looking for is the degree of adjectives describing a restaurant and then maintaining a model for comparison with other restaurants. If we consider a scenario in which you can read a summary of any restaurant reviews with its pros and cons on food, ambience and service. This would help to efficiently compare two restaurants in a matter of seconds.

In this project we aim to generate an aspect based sentiment summary of a particular restaurant in a completely automated fashion from the raw text (and metadata) of Yelp reviews about that restaurant taken as an input. Once a restaurant is selected, our program extracts the "aspects" from the reviews ("service", "food", "sweet potato fries", “ambience” etc.) of that restaurant through Part-Of-Speech tagging and shallow chunking. These are the salient features of the restaurant that reviewers often comment on. After determining the most important features in a restaurant’s reviews, our system looks for opinion words occurring in close proximity to the feature. The aspects are displayed in our Web Application and when one of the aspects is selected, using our sentiment/opinion classifier which is a supervised two stage machine learning approach, we classify the opinion words as positive or negative, thus generating a visually appealing and quickly readable summary of reviewers' attitudes towards the aspect. A Positive/Negative score on top of the sentiment (positive and negative) summary displays the overall balance of sentiment that our program uncovered in the data with respect to the chosen aspect. The program provides supporting evidence in the form of snippets from reviews.

II. DESIGN



III. METHODOLOGY AND IMPLEMENTATION

Our system is a fine-grained approach. We focus on extracting sentiments about the features in a set of reviews of a restaurant rather than summarizing the reviews. We start with identifying significant features of all restaurants like food, service and ambience from the reviews of the restaurant and extract them. We then determine the sentiment of each feature of the restaurant. We perform two-stage machine learning algorithms and then visually summarize the opinions about each restaurant. We first perform training/optimizing on the machine learning models that currently power our sentiment summarization miner. Our Modeling pipeline is described below. Three kinds of data were needed for the project; Yelp Data, Sentiment Analysis Data, Aspect Extraction Data. Each dataset is acquired in stages.

III.1 Data Extraction and Parsing

We've used Yelp dataset as our input primary data. We scraped data from Yelp website and downloaded the business.json, reviews.json, users.json files. We collected data of 3GB and then prepared it for modeling process i.e. reading in the raw data provided by Yelp from file, converting to usable form, and filtering unnecessary information. This step reads in the following Yelp datasets: review data set, business data set, and user data set. Using the pandas library in python we create a dataframe. Pandas.DataFrame is a two-dimensional, size-mutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns). Arithmetic operations align on both row and column labels. Can be thought of as a dict-like container for Series objects. We read raw data from each json file, add it to the dataframe and finally output this dataframe. We first process the business dataframe. The raw data from business.json is cleaned to keep only those columns that are necessary to us i.e. 'name, 'attributes', 'stars', 'categories', 'review_count' and 'business_id'. Then we further clean the categories in business data to keep only restaurants as our business. We've set a threshold of 300 to filter and retain only those restaurants that have number of reviews equal to or greater than this threshold. Then we process our Users dataframe. The raw data from users.json is cleaned to keep only these columns 'user_id', 'average_stars', and 'name'. Lastly, we process our Reviews dataframe, read and clean the raw data to keep only 'business_id', 'review_id', 'stars', 'text', 'user_id'. At the end of this step, we merge the three datasets and write the resulting DataFrame to disk. The processed csv we get at the end of this step is used in later steps.

III.2 Featurizing Training Data

This step is responsible for reading in the manually tagged training data and featurizing it for model training. We first featurize the training data, then merge in relevant features from the raw yelp data and finally split data into development and pure holdout sets writing these to disk. To train sentiment classifier, we need some tagged sentiment data. We start by loading in the training data to our dataframe which we hand-tagged ourselves with positive, negative, objective and neutral sentiments. We load by using the readcsv function from pandas and clean up the dataframe columns for merging, keeping 'review_id', 'sentence', and 'sentiment'. We then merge our training data to the remaining dataframe. In order to fix the stray values we change all the neutral sentiments to negative. Then we read in the yelp data from the processed csv generated in the previous step and keep only the columns 'business_id', 'review_id', 'user_id', 'review_stars', 'user_avg_stars'. After this, we merge the training and Yelp data frames on review id by dropping few unmatched values. In order to featurize the training data frame we take the sentences in the final dataframe as objects and set the star property for the object to the stars for that review. We extract features from this object and add it to a dataframe. We know our sentence object has the following 4:-

(i) ***A word tokenizer*** for converting a raw string (sentence) to a list of strings (words) which we get from the algorithm MyPottsTokenizer by Chris Potts.

(ii) ***A lemmatizer*** we get from the algorithm WordNetLemmatizer() which we imported from NLTK.

(iii) ***A featurizer*** we get from the algorithm Metafeaturizer by combining the two featurizer objects; SubjFeaturizer(), LiuFeaturizer() by Minqing Hu & Bing Liu.

(iv) ***An aspect extractor*** - a chunk tree we get by passing our grammar (Nouns and Adjectives, terminated with Nouns. And this connected with in/of/etc.) we set for NP chunking to the regex parser of nltk and then parsing this on POS tags achieved from applying POS tagging on our sentence. The aspects are the noun phrases we get from this chunk tree. If the aspects are valid that is it's not a stopword or forbidden word then only it is returned. We created our own stopword list and forbidden words along with the one imported from nltk.

Therefore we get an ordered feature dictionary for a sentence by passing our sentence to the featurize function of the featurizer above-(iii)

The sentiment category in this dataframe was set to the sentiment values from the final dataframe. We then adjust the sentiment labels by giving '1' to positive sentiments, '-1' to negative features, '0' to objectives. Finally in this step we create an opinionated column in the dataframe we created for features extracted. For all the sentiments that weren't objective, we classify them as opinionated in this column by assigning value of 1. We take out a random sample from our dataframe. In order to make sure sample size remains the same no matter what data size is, we provide a random seed. We write this random sample to our disk in two parts. Using 'ix' in pandas which is a primarily label-location based indexer with integer position fallback, we pass random sample to ix. The result we get is a property of our featurized dataframe, which we copy and write to disk as pristine holdout using function to_csv in pandas. Similarly we use 'drop' (drop (labels [, axis, level, inplace, errors])) in pandas, which returns new object with labels in requested axis removed. Passing random sample to drop, the result we get is a property of our featurized dataframe, which we copy and write to disk as featurized development csv.

III.3 Opinion and Sentiment Grid Search CV using Support Vector Machine and Logistic Regression

This step runs a grid search to optimize hyper parameters for both the opinion and sentiment models. The final models tuned by the grid search are ultimately pickled for later use in the project's main summary-generation pipeline. This step first reads in the development set we got from previous step, prints out its size, prints the class breakdown of our sentiment and opinionated column of dataframe giving count for positive, negative and neutral for sentiment model and count for subjective, objective for opinion model. This is shown in our results section. Then we run a grid search for:
- Best OPINION MODEL (opinionated vs. not opinionated)
- Best SENTIMENT MODEL (positive vs. negative, assuming opinionated)
For our Opinion model we use SVM (Support Vector Machine) grid search. Support vector machines (SVMs) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Given a set of training examples, each marked for belonging to one of two categories, an SVM training algorithm builds a model that assigns new examples into one category or the other, making it a non-probabilistic binary linear classifier. We used the sklearn library and its cross_validation.train_test_split function, which splits arrays or matrices into random train and test subsets. We used this function on the opinionated column of our development set to get our train size and test size. Then we call the StandardScalar function of the sklearn pipeline class to standardize feautures by removing the mean and scaling to unit variance by computing the relevant statistics on the samples in the training set. We pass the scaled result to Pipeline, which sequentially applies a list of tranforms and a final estimator that implements fit. The purpose of the pipeline is to assemble several steps that can be cross-validated together while setting different parameters. Using the grid search cv algorithm of the sklearn grid search class over the estimator, we implement fit and score method, predict, predict_proba and best_estimator to return the fit for the best estimator. We print the results of our grid search i.e. best params found, best estimator and grid scores on the train set we get from score method. We also print classifier results for this model by generating standard metrics for binary classification evaluation -our classification report. Using metrics class of sklearn we do this.
Similarly for our Sentiment model we use Linear Regression grid search. Logistic regression is a statistical method for analyzing a dataset in which there are one or more independent variables that determine an outcome. The outcome is measured with a dichotomous variable (data coded as 1). The goal of logistic regression is to find the best fitting model to describe the relationship between the dichotomous characteristic of interest and a set of independent variables. As in the above model, we use cross_validation.train_test_split function on the sentiment column of our development set to get our train size and test size. Using the grid search cv algorithm, we implement fit and score method, predict, predict_proba and best_estimator to return the fit for the best estimator. We print the results of our grid search i.e. best params found, best estimator and grid scores on the train set we get from score method. We also print classifier results for this model.
Finally, we store the final models to our disk.

III.4 Aspect-based sentiment summary using the opinion and sentiment models- Highest-level analytical method

To run the project, there should be Mongo Database set up and running with DB as Yelptest2 and collection called summary. In this step we first create a MongoClient connection. Then we read data from processed csv we generated in the modeling step III.1 and from this panda data frame we extract unique business ids. For each business id we get its review ids and text of all reviews for that business to make a Business object with all reviews as Review Objects. Then we perform summarization. Taking this Business object as input, we output a dictionary; a json object encoding the aspect-based sentiment summary for the given business, ready to be written to MongoDB, and containing everything (in correct orders) that will be displayed by the front end. This step effectively runs the full analysis for this Business. We start by extracting aspects of the business by maintaining a single word threshold: how common does a single-word aspect need to be in order to get included in the summary? And a multiple word threshold: - how common does a multi-word aspect need to be in order to get included in the summary? The former is higher as it is noisy (and so higher threshold is needed for high precision). The aspects are a list of lists of strings - e.g. [['pepperoni', 'pizza'], ['wine'], ['service']]. These aspects are most often commented on in a business/restaurant. Currently, aspect extraction is based on frequent noun-phrase counting. That is, inclusion in the summary is determined by frequency of occurrence across sentences, as the former tends to be much more. Firstly, we get all candidate aspects in each sentence and create a single-word and multi-word aspect list. Then we get sufficiently common single and multi word aspects. We filter redundant single-word aspects. The full aspect list is sorted by frequency. Filter out those one-word aspects that are subsumed in a multi-word aspect. E.g. filter out "chicken" if "pesto chicken" is a multi-word aspect and return these aspects. For each aspect, we make a dictionary of aspect summary with keys 'pos' and 'neg', which map to a list of positive sentences (strings) and a list of negative sentences (strings) correspondingly. We get summary for a *particular* aspect that includes primarily the sorted positive/negative sentences mentioning this aspect. Mathematic formulas and probability is applied and each sentence is given a weightage based on whether it's positive or negative using our opinion and sentiment model. Then we do final filtering of this dictionary and print it. Filtering is done to make sure sentences with number of positive or negative words are greater than 5, otherwise its omitted from the summary. This summary is inserted into Mongo and displayed.

III.5 Web Application

Using techniques in Jinja templates, python, JS and CSS, we create a beautiful and very informative UI where we show a list of restaurants. Once we click on a restaurant, we are taken to a page where we show all our features on which a user can see an opinion. Once a feature is selected, we show all the positive and negative reviews of that feature.
We host our app on http://0.0.0.0:80. Before running applications for displaying UI, we need to have a MongoDb (database) setup with summaries column in place, which holds all the summaries for all the restaurants. Then we get list of business names/ids from Mongo and keep it in a data structure. We also keep summaries of each and every business id stored in a data structure. Once we have these stored, we call our jinja templates to display all this data in the form of opinions (positive & negative) and show it as shown in results. Justin Windle's CSS style sheet was very useful to create our UI.
For example the pseudo code for displaying the list of businesses (using unique business ids) is:

```
<dl class="list nigiri">
<dt>Resturants</dt>
        {% for business in businesses %}
            <dd>
                <a href="/summaries/{{ business['business_id'] }}">{{ business['business_name'] }} </a>
          </dd>
        {% endfor %}
</dl>
```

IV. RESULTS

System Evaluation in terms of precision

Once we created our *opinion model* and *sentimental model* using the methodologies explained in section III, we ran cross validation using classification report function of the sklearn metrics library, to evaluate our system, so that we can get an estimate as to where does the system stands in terms of precision, recall, f1-score & support.

Opinion_Model Classification Report:

	precision	recall	f1-score	support
0	0.70	0.79	0.74	669
1	0.93	0.85	0.88	4882
avg / total	0.90	0.84	0.86	5551

Fig a. Opinion Model Evaluation Report

In Fig a. we can see the performance report for the tuned Support Vector Machine that basically generates our Opinion Model based on the training data and the test data. 0 signifies the objective sentences (reviews) and 1 signifies the subjective sentences (reviews). The precision for the objective sentences is comparatively less than the subjective ones as we have hand tagged the training data for the opinion model and thus had fewer amounts of training data for the same.

Sentiment_Model Classification Report:

	precision	recall	f1-score	support
-1	0.80	0.75	0.77	402
1	0.95	0.79	0.86	4446
avg / total	0.93	0.78	0.85	4848

Fig b. Sentiment Model Evaluation Report

In Fig b. we can see the results of the same evaluation of the positive/negative reviews, which are extracted from the subjective reviews tagged by the Sentiment Model. -1 signifies the negative sentences and 1 signifies the positive sentences. As we can see we are achieving a 93% average precision for tagging reviews as negative/positive, which makes the final application accurate and reliable.

```
Size of complete development set: 27752
Target class breakdowns:
 1    22199
 0     3516
-1     2037
```

Fig c. Complete Development Set

Fig c shows the development set we used to train the system for positive, neutral and negative sentences. As we can see that the amount of negative examples are less in comparison to positive ones, and thus the precision for negative sentences is less than the others. We still try our best to account for this imbalance by heavily weighting negative examples. So that while running it on the testing data negative words gets weighted heavily to compensate for the less amount of training data. The amount of development data available was majorly non-negative.

Sample Results: Aspect Based Opinion Miner

Aspect Based Opinion Miner is the graphic based front-end representation of the system, which will be used by the user to view the summarized reviews of the said restaurants. It is built using python and jinja templates and connects to mongo database running on the server to get its data. It comprises of a landing page, which contains the lists of restaurant's to choose from, upon selecting a restaurant you can see the categories of the specific restaurant extracted by the aspect extractor. And then in those categories find the summarized reviews classified as negative or positive with their rating as well and the user who wrote them.

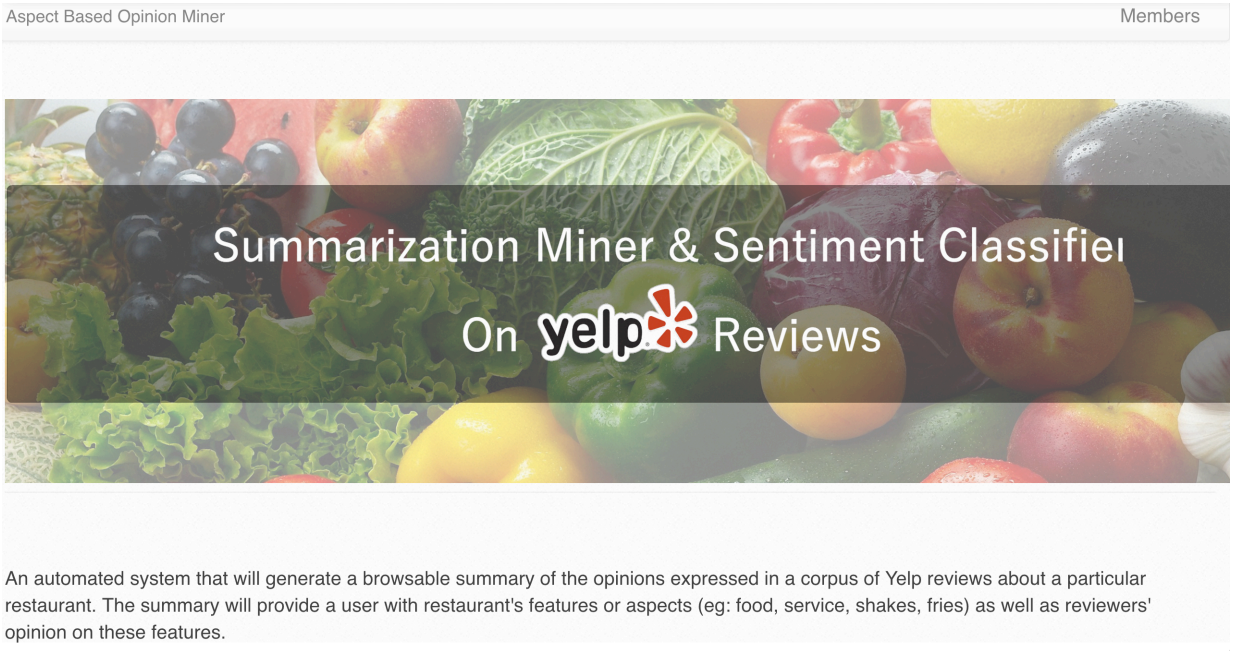


Fig d. Landing Page of the application

Please select a restaurant to explore aspect based opinion mining results:

Fig e. List of Restaurants to choose from

For showing some sample results we took the first 30 restaurants from the list of the results which consists of more than 500 restaurants, the user can select any restaurant he wants and view the categories of the restaurants and summarized reviews of all the categories. Fig e shows the style in which the list of restaurants is displayed and Fig f depicts a sample restaurant with categories and a category 'sushi chef' selected and the positive/negative reviews (summarized) that contain sushi chef followed by the user who wrote them. And also the number of positive and negative comments, whereas the display of reviews is limited to top 10.

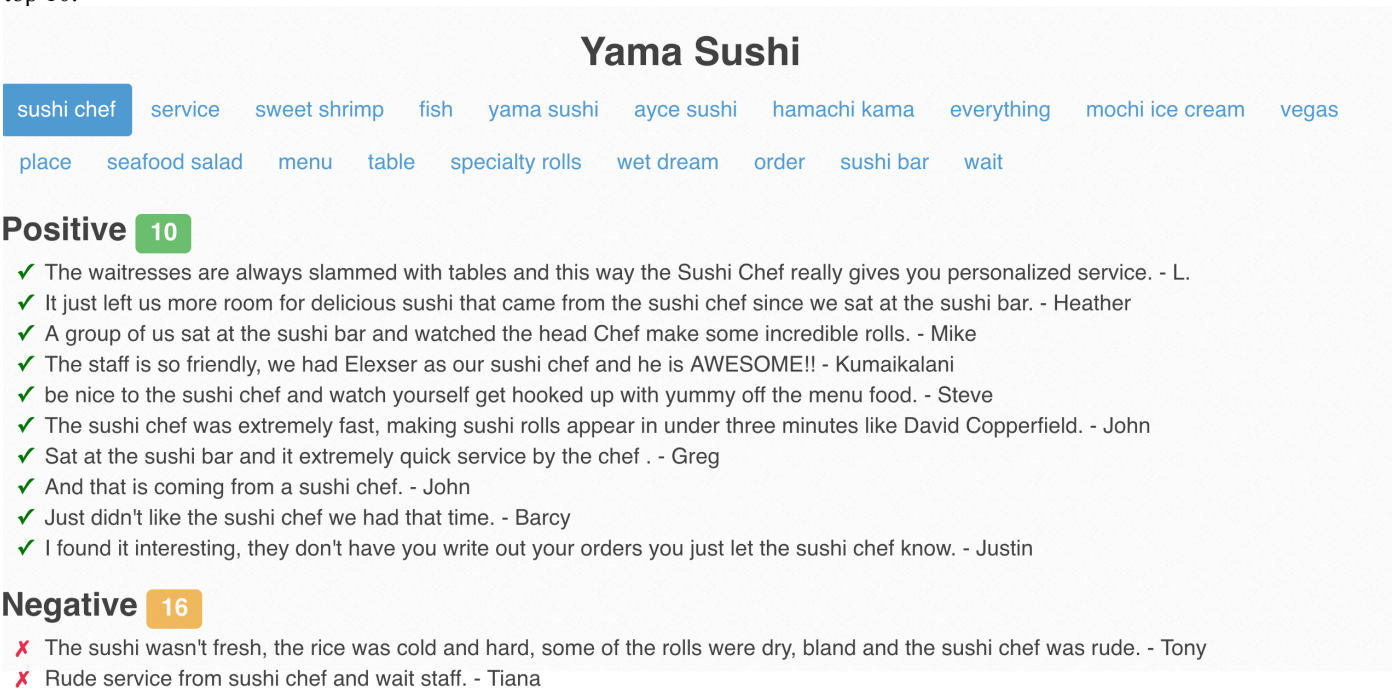


Fig f. Yama Sushi Restaurant with Positive/Negative Reviews for category 'sushi chef'

V. Conclusion

The goal of this project was to develop an aspect-based summarizer for sentiments expressed in Yelp reviews i.e. take as input a set of user reviews for a particular restaurant and produce, a set of aspects, an aggregate sentiment score for each aspect and supporting textual evidence. The user attitudes towards food vs. décor vs. service is extracted, then aggregated across users to get a clearer picture of what users like or dislike about a particular restaurant.

VI. Acknowledgment

We would like to thank Yelp for providing us with the data without which we couldn't have had run the analytics to get the desired results. We would like to thank Ralph Grishman for giving us the opportunity to explore these new technologies in the industry and equipping us with proper skillset to use these technologies.We would also like to extend thanks to Christopher Potts for writing an open source tokenizer and Miging Hu & Bing Liu for providing the opinion lexicon and a featurizer which helped us to classify words as positive & negative. Lastly, we would thank the graders for regularly helping out whenever we were stuck.

VII. References

[1] http://sentiment.christopherpotts.net/

[2] http://sentiment.christopherpotts.net/lexicons.html#wnpropagate

[3] Blair-Goldensohn, Kerry Hannann,Ryan McDonald.Building a Sentiment Summarizer for Local Service Reviews.

[4] Svetlana Kiritchenko,NRC-Canada-2014: Detecting Aspects and Sentiment in Customer Reviews

[5] https://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html

[6] Abhishek Gupta, Sentiment based Summarization of Restaurant Reviews, June 2009

[7] http://textminingonline.com/dive-into-nltk-part-iv-stemming-and-lemmatization

[8] http://www.nltk.org/api/nltk.stem.html

[9] Minqing Hu and Bing Liu. Mining and Summarizing Customer Reviews.