Yelp Explorers

Final Project Report

Parul Singh, Abhinava Singh, Ilakya Palanisamy, Mukund Chillakanti

Background/Motivation

Restaurants are constantly getting feedback. Yelp is one channel, but many restaurants document verbal feedback, gather feedback from questionnaires, or even from other review sites. When faced with a huge amount of text, restaurants need a way to objectively interpret their reviews. By building a way to relate text based feedback to a star rating, we can help these restaurants understand how they would rate on Yelp.

While we focus on restaurant rating prediction, there has been related work focused on review rating prediction for Yelp reviews.[1] They use similar sentiment analysis methods such as bag of words and part of speech tagging.[2] There has also been some analysis of Yelp reviews using Doc2Vec methodology, but such work is limited and does not exist for specifically restuarant rating prediction.[3]

Problem Statement

Can we predict a restaurant's Yelp rating from just the text of its reviews? We want to learn the best mapping from a review word vector to a restaurant rating.

Data

The data we used comes from the following link: http://www.yelp.com/dataset_challenge. This data has been made available by Yelp for the purpose of the Yelp Dataset Challenge. In particular, the challenge dataset contains the following data:

- **1.6M** reviews and **500K** tips by **366K** users for **61K** businesses
- **481K** business attributes, e.g., hours, parking availability, ambience.
- Social network of **366K** users for a total of **2.9M** social edges.
- Aggregated check-ins over time for each of the **61K** businesses

Businesses reviewed in this dataset are located in the following cities around the world:

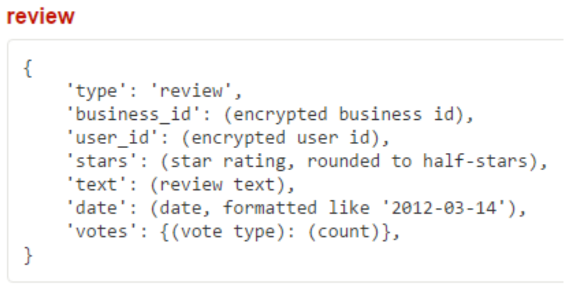
- U.K.: Edinburgh
- Germany: Karlsruhe
- Canada: Montreal and Waterloo
- U.S.: Pittsburgh, Charlotte, Urbana-Champaign, Phoenix, Las Vegas, Madison

[1] https://cs.uwaterloo.ca/~nasghar/886.pdf

[2] http://cs229.stanford.edu/proj2014/Chen%20Li,%20Jin%20Zhang,%20Prediction%20of%20Yelp%20Review%20Star%20Rating%20using%20Sentiment%20Analysis.pdf

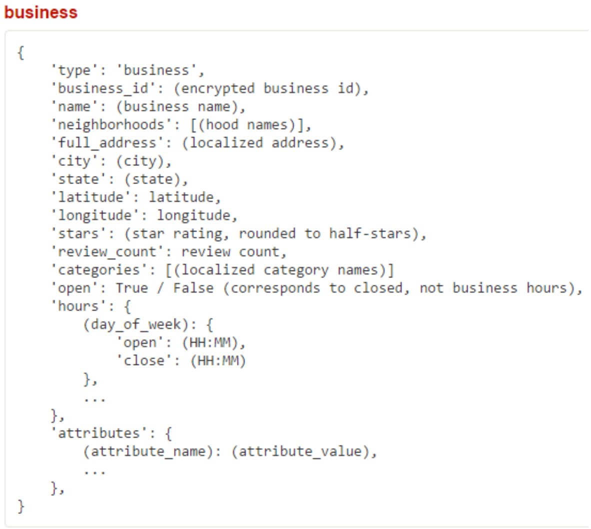
[3] https://github.com/KenDooley/Tast.io/blob/master/README.md

The data is provided in the form of five JSON files, one for each object type; the object types are businesses, check ins, reviews, tips, and users. Each JSON file consists of several JSON objects of the same type (e.g. each review is represented as a JSON object in the review.json file). See below for examples of the formatting used for these JSON
objects in the review.json and business.json files:



review

```
{
    'type': 'review',
    'business_id': (encrypted business id),
    'user_id': (encrypted user id),
    'stars': (star rating, rounded to half-stars),
    'text': (review text),
    'date': (date, formatted like '2012-03-14'),
    'votes': {(vote type): (count)},
}
```



business

```
{
    'type': 'business',
    'business_id': (encrypted business id),
    'name': (business name),
    'neighborhoods': [(hood names)],
    'full_address': (localized address),
    'city': (city),
    'state': (state),
    'latitude': latitude,
    'longitude': longitude,
    'stars': (star rating, rounded to half-stars),
    'review_count': review count,
    'categories': [(localized category names)]
    'open': True / False (corresponds to closed, not business hours),
    'hours': {
        (day_of_week): {
            'open': (HH:MM),
            'close': (HH:MM)
        },
        ...
    },
    'attributes': {
        (attribute_name): (attribute_value),
        ...
    },
}
```

Informal Success Measures

The way we will measure the success of our problem is through the accuracy on cross-validated data of predicting restaurant rating from review text.

Methods

Data Collection

As previously specified, we used Yelp review data made public for the Yelp Dataset Challenge. We simply downloaded all JSON files from the following link: http://www.yelp.com/dataset_challenge.

Additionally, we obtained a comprehensive list of categories for businesses on Yelp as a csv file from the following source http://www.localvisibilitysystem.com/2013/07/19/yelp-business-categories-list/

Transformation

All data pertaining to our project (since we are interested in rating restaurants purely based on text reviews) is contained in the business.json and review.json files, so we extracted the data from these files and stored them in two Pandas DataFrames.

In addition, we extracted all valid restaurant categories from the business category csv file and stored them in a python list for quick access. We ended up with about 110 potential restaurant categories, including categories such as "American", "Burgers", "Breakfast & Brunch", to name a few.

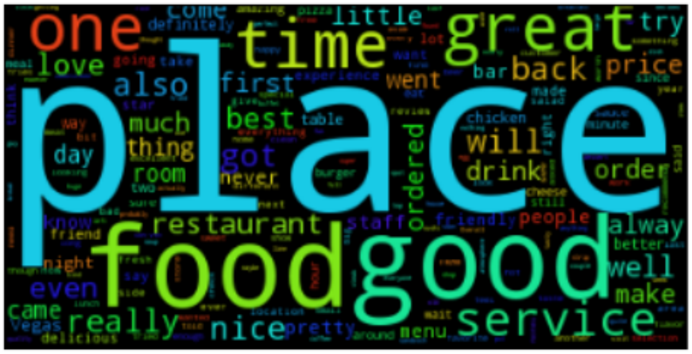
Cleaning and Repair

Since we are only interested in restaurants, we created a restaurant dataframe consisting of all businesses that are restaurants. We did this by filtering businesses in the business DataFrame based on their ‘category’ field; we categorized a business as a restaurant if at least one of its listed categories matched at least one of our restaurant categories. Additionally, we filtered restaurants that had no text reviews associated with it by ensuring that every business in the restaurant DataFrame had a value greater than 0 for the ‘review_count’ field.

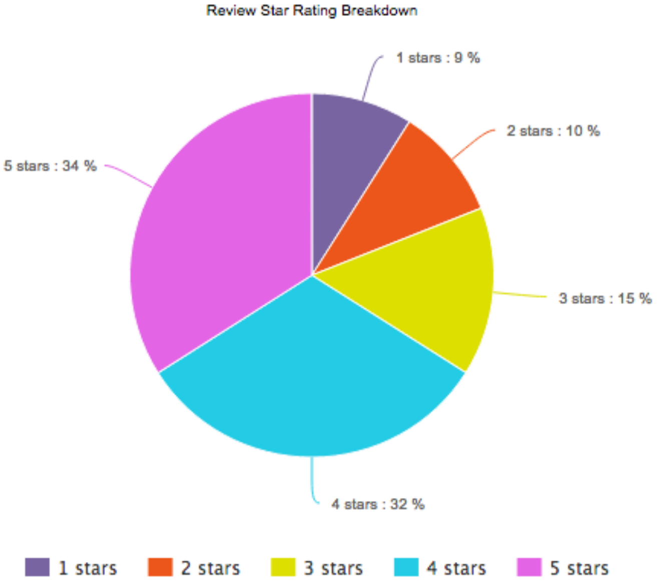
Finally, before working on predicting a rating for a restaurant, we wanted to consider the binary case of predicting of the overall sentiment towards a restaurant (positive or negative) as a baseline for our classifier performance. To do this, we had to do some additional data preparation, specifically adding a sentiment column for each restaurant in the restaurant dataframe. Restaurants with a rating of 4 stars or greater were assigned a positive sentiment; the rest were assigned negative.

Initial Data Analysis & Visualization

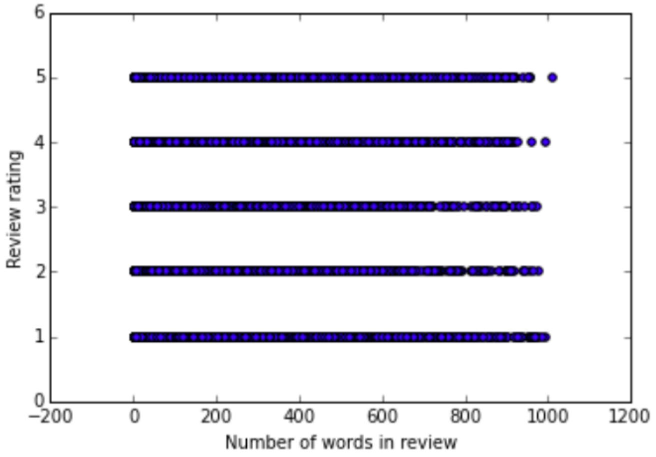
We then sought to understand our data better through visualizations.



We generated a word cloud of the 100 most frequently appearing words across all restaurant review text, to get an idea of the vocabulary people tend to use in their reviews. We noticed a lot of the bigger words seemed positive/neutral (such as “good”, “great”, “love”, “nice”), which seemed to indicate that most of the reviews in our dataset were positive. It was also interesting to see words such as “service”, “time” and “price” appear large in this cloud, as it gave us an initial indication of things that might matter a lot to reviewers.



We generated this pie chart that shows the distribution of star ratings of all reviews. Most reviews had positive ratings (4 or 5 stars).



We also explored whether there was any pattern between the number of words in reviews to the rating of review. We generated a scatter plot of reviews, with the X axis being the number of words in that review and the Y axis being the rating for that review. The scatter plot seemed to show no general pattern or correlation between size of text and rating. At each rating there seem to be reviews across a wide range of sizes. This was an example of a feature that we ended up not using, after our analysis showed it would not be a strong indicator.

Featurization Algorithms

We used two approaches to featurize our Yelp data, Bag of Words and features generated by Doc2Vec. The reason we also explored Doc2Vec was because we realized that the Bag of Words was missing many linguistic contexts of words that would be needed to improve our classification accuracy. We used Doc2Vec as a 2 layer Neural Network that relied on a continuous bag of words (found from the Yelp Corpus) to create word embeddings which provided us a richer feature set than Bag of Words.

Bag of Words

Our first approach was to use a bag of words model of the most frequent unigrams, bigrams, and trigrams appearing in text reviews. The bag of words model was most appropriate for the problem we are trying to solve, since we want to use only the feedback text provided by a restaurant in making a rating prediction for them.

Before extracting the n-grams, we first apply a word stemmer and a word lemmatizer to reduce related forms of a word to a common base form to improve our bag of words. We also filter out common stopwords from the text.

Then we find the 2000 most frequently occurring unigrams by constructing a dictionary mapping all unique words in the pre-processed review text to a count. To capture the relationship between words and the effect of phrases such as “great service” or “will come back”, we perform a similar process for bigrams and trigrams. In the end, we have a feature vector consisting of the 2000 most frequently occurring unigrams, 1000 most frequently occurring bigrams, and 500 most frequently occurring trigrams.

Using these features, we construct a feature matrix consisting of a feature vector for each of our 17109 restaurants. To avoid memory issues in the generations of bigrams and trigrams, we realized that a dictionary that contains the most common n-grams will cover most of the n-grams we’ll see. Thus for bigrams and trigrams, we first generate a vocabulary using a random subset of the data, and loop through subsections of restaurants to compute feature vectors.

Doc2Vec

Our second approach was to use a trained Doc2Vec model to generate our features. We first split up our restaurant review texts into test, training, and validation sets for each class. This means we had biz_{test | train | validation}_{label}.txt files, which stored the reviews for the corresponding label,type pair. Then we assigned the words of each file to a tag which was the label of that file. We used Genesim’s LabeledLineSentence object to do this. The next step was to train the model which looked like so:

```
model = Doc2Vec(min_count=1, window=10, size=100, sample=1e-4, negative=5,
workers=8)
model.build_vocab(sentences.to_array())
for epoch in range(10):
        model.train(sentences.sentence_perm())
```

Once the model was trained, getting the vector for a particular label was just a matter of typing model[{tag string here}]. Finally we aggregated the training vectors for various tags and we had our feature set that was generated by Doc2Vec.

Learning Algorithms

For binary classification of a restaurant sentiment, we predict whether a restaurant has a rating higher or lower than 3 as a baseline classification problem. We compared the performance of Logistic Regression, SVM, Random Forest classifiers. We then tackle the prediction of 1-5 star rating of a restaurant, again comparing Logistic Regression, SVM, and Random Forest Classifiers.

In both cases, we also use Singular Value Decomposition (SVD) and Chi-squared tests to observe change in accuracy of the classifiers with varying feature set size and to identify the most informative features.

Finally, we perform addition tuning for each of our classifiers using 5-fold cross-validation. We tune for the optimal C value for logistic regression corresponding to regularization strength, the optimal norm used in penalization (l1 vs l2) and penalty parameter (C value) for SVM, and the number of trees, max_depth, and use of bootstrapping for Random Forests. Further down in our report, we list the exact parameter choices we used for each method that worked best on the data.

Brief Discussion of Some Methods That Didn’t Work or Perform Optimally

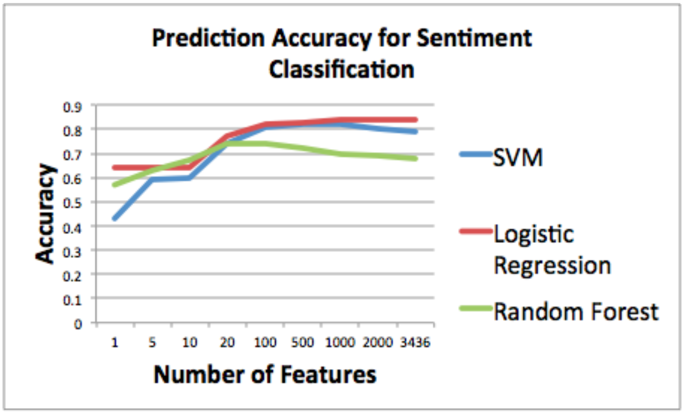
We mentioned earlier how some of our initial data exploration led us to realizing that features such as average number of words in reviews for a business, or distance to city center showed little correlation to the actual rating of a business. We could see this from scatterplots and histograms of our review data. In addition, in applying learning algorithms, we tried a Random Forest classifier for this rating prediction problem, but we did not expect it to perform as well as Logistic Regression or SVM given a bag of words feature model, since our feature vectors are quite large and also very sparse, and Random Forest is usually not the best choice for high-dimensional, sparse data. We thought using SVD to reduce the feature set size to the most impactful features might help Random Forest performance, so we still include these results in our next section. Furthermore, we include results of both Bag of Words and Doc2Vec to compare the performance of these two models of featurizations, although we expected Doc2Vec to produce a slight improvement over the traditional bag of words due to its richer feature set of word vectors.

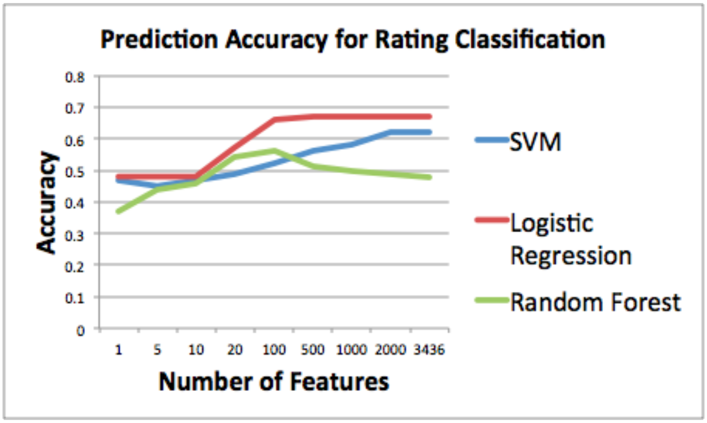
Results

The below graphs are using our bag of words (BoW) feature model consisting of unigram, bigrams, and trigrams. They show prediction accuracy as number of features increases for sentiment classification and rating classification of a restaurant using Logistic Regression, SVM, and Random Forest prior to tuning hyperparameters. Feature reduction was done using Singular Value Decomposition (SVD).

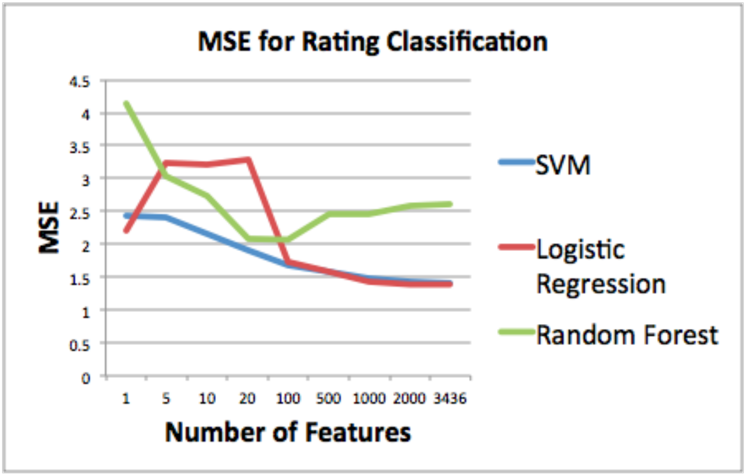
If we take a look at the rating classification graph, Random Forest consistently performs a little worse than SVM and Logistic Regression and achieves peak accuracy at N=100 features. SVM and Logistic Regression perform comparably, with Logistic Regression achieving a slightly higher peak accuracy before tuning. However, as you can see on the graph, the accuracies for both SVM and Logistic Regression both do not increase much when going above 1000 features.

In fact, in the sentiment classification graph SVM peaks at 100 features before dipping lower. Without hyper-parameter tuning, logistic regression consistently achieves slightly better prediction accuracy than SVM in these graphs.





We use MSE to quantify our error, as shown in the below graph. We used MSE to quantify our error since it tells us about the average magnitude of our error in rating prediction compared to a restaurant's true rating. This gives us more information than just true prediction accuracy as to how well our classifier is performing. As the number of features increases, we can see the MSE drops initially but stays around the same after 1000 features. Thus, using all 3500 features instead of just 1000 doesn't impact the quality of our classifier. After looking at these graphs we decide to tune these SVM and Logistic Regression at 1000 features and Random Forests at 100 features because that is where their accuracies peaked.



Tuning

After tuning SVM at 1000 features, and in particular using the optimal C value (1e-5) and penalty (L2) that we found, SVM leads to our best performance on both sentiment and rating classification, for our bag of words as well as our Doc2Vec models. The following are the optimal hyperparameters we found for rating classification, by tuning using cross-validation. For finding the best combination of hyperparameters, we made use of sckit-learn’s GridSearchC, and used 5 fold cross-validation. These were our optimal hyperparameters for rating classification:

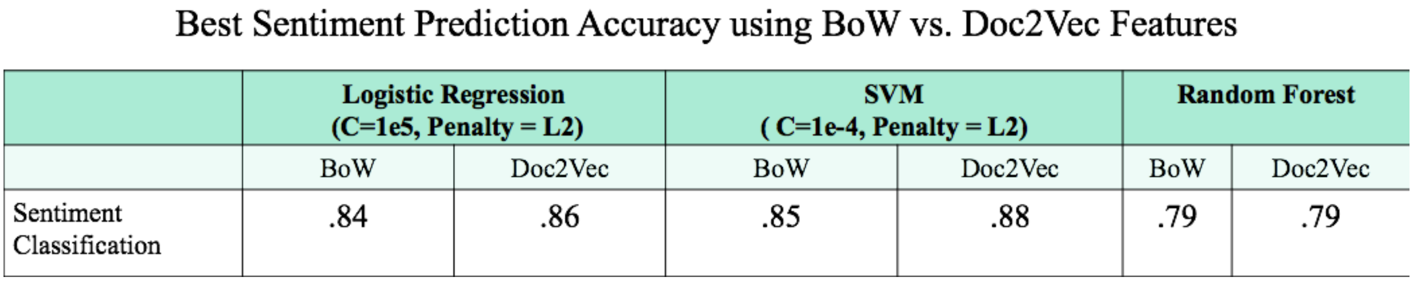
Parameter Choices

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
        max_depth=None, max_features='auto', max_leaf_nodes=None,
        min_samples_leaf=1, min_samples_split=2,
        min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
        oob_score=False, random_state=None, verbose=0,
        warm_start=False)

LogisticRegression(C=100000.0, class_weight=None, dual=False,
       fit_intercept=True, intercept_scaling=1, max_iter=100,
       multi_class='ovr', penalty='l2', random_state=None,
       solver='liblinear', tol=0.0001, verbose=0)

LinearSVC(C=1e-05, class_weight=None, dual=True, fit_intercept=True,
   intercept_scaling=1, loss='squared_hinge', max_iter=1000,
   multi_class='ovr', penalty='l2', random_state=None, tol=0.0001,
   verbose=0)
```

The following 3 tables show the best prediction accuracy and RMSE we were able to achieve for each of our three classifiers, using BoW and Doc2Vec. For sentiment classification, SVM with Doc2Vec features performs the best with an 88% accuracy after hyperparameter tuning. [4]



Best Sentiment Prediction Accuracy using BoW vs. Doc2Vec Features

	Logistic Regression (C=1e5, Penalty = L2)		SVM (C=1e-4, Penalty = L2)		Random Forest	
	BoW	Doc2Vec	BoW	Doc2Vec	BoW	Doc2Vec
Sentiment Classification	.84	.86	.85	.88	.79	.79

[4] In the graphs, rating classification was done by rounding down decimal ratings to the nearest integer, making a correct prediction one that predicts the true rating after rounding. Accuracies on the graphs are slightly lower than those shown as the best results on our table, in which ratings were estimated within .5 of true rating.

For rating classification, SVM again performs the best with 88% accuracy in predicting a rating within .5 of the actual rating, using Doc2Vec features. This was a slight improvement over the Bag of Words model for rating classification, which gave 87% accuracy with SVM.

Best Rating Prediction Accuracy using BoW vs. Doc2Vec Features

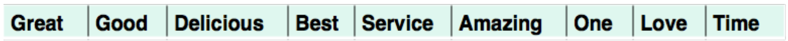
	Logistic Regression (C=1e5, Penalty = L2)		**SVM (C=1e-5, Penalty = L2)**		**Random Forest**	
	BoW	Doc2Vec	BoW	Doc2Vec	BoW	Doc2Vec
Rating Classification* *within .5 of actual rating	.84	.87	.87	.88	.77	.78

We can see from the RMSE table that using Logistic Regression with Doc2Vec features with a C value of 1e-5 and L2 penalty, gave our lowest RMSE of .28, with SVM just behind with an RMSE of .29. Just as with prediction accuracy, Doc2Vec gave slight improvement in RMSE for both Logistic Regression and SVM. Random Forest again did not perform as well as the other two classifiers, with a best RMSE of .47, using the hyperparameters shown earlier, and the optimal feature size for Random Forest of 100 features. This is to be expected given that Random Forests tend to not do well on high dimensional sparse data.

Best RMSE using BoW vs. Doc2Vec Features

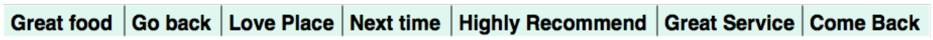
	Logistic Regression (C=1e5, Penalty = L2)		**SVM (C=1e-5, Penalty = L2)**		**Random Forest**	
	BoW	Doc2Vec	BoW	Doc2Vec	BoW	Doc2Vec
Rating Classification* *within .5 of actual rating	.35	.28	.31	.29	.47	.49

We also found the highest scoring bigrams and trigrams using a chi-squared test. Below are some of the n-grams that scored highly. You can see that some of the most informative features for classification are positive adjectives like "great", or words referring to wanting to return to a restaurant like "come back" or "go back", or descriptions of the service such as "great service.



Great	Good	Delicious	Best	Service	Amazing	One	Love	Time

Highest scoring unigram features from a chi-squared test



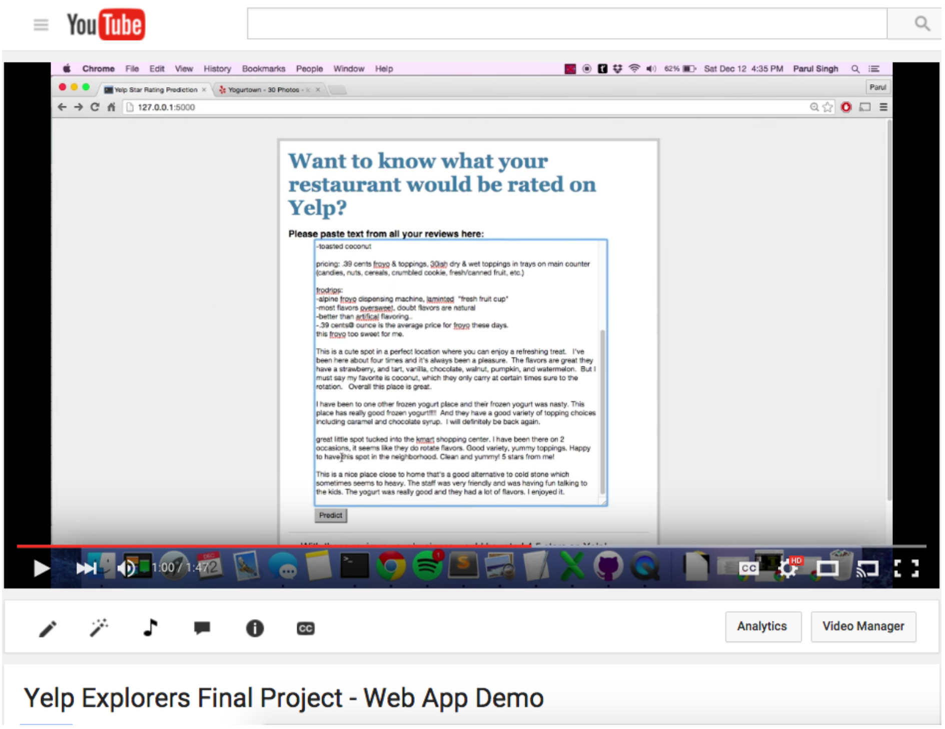
Great food	Go back	Love Place	Next time	Highly Recommend	Great Service	Come Back

Highest scoring bigram features

Visualizations

We used Flask to create a single page web application, which uses our pickled classifier to illustrate the potential benefits our classifier could have for restaurants that are not on Yelp. We were unable to host the website so we created a short demo YouTube video going through the functionality of the app.

Link to Demo for App: https://www.youtube.com/watch?v=LZgQLQNnzh4&feature=youtu.be
Github Repository for web app: https://github.com/parulsingh/FlaskAppCS194


You Tube
Chrome File Edit View History Bookmarks People Window Help
62% Sat Dec 12 4:35 PM Parul Singh
Yelp Star Rating Prediction
Yogurtown - 30 Photos
127.0.0.1:5000
1:00 / 1:47
Analytics
Video Manager

Want to know what your restaurant would be rated on Yelp?

Please paste text from all your reviews here:

-toasted coconut

pricing: .39 cents froyo & toppings. 30ish dry & wet toppings in trays on main counter (candies, nuts, cereals, crumbled cookie, fresh/canned fruit, etc.)

frodrips:
-alpine froyo dispensing machine, laminted "fresh fruit cup"
-most flavors oversweet, doubt flavors are natural
-better than artifical flavoring..
-.39 cents@ ounce is the average price for froyo these days.
this froyo too sweet for me.

This is a cute spot in a perfect location where you can enjoy a refreshing treat. I've been here about four times and it's always been a pleasure. The flavors are great they have a strawberry, and tart, vanilla, chocolate, walnut, pumpkin, and watermelon. But I must say my favorite is coconut, which they only carry at certain times sure to the rotation. Overall this place is great.

I have been to one other frozen yogurt place and their frozen yogurt was nasty. This place has really good frozen yogurt!!!! And they have a good variety of topping choices including caramel and chocolate syrup. I will definitely be back again.

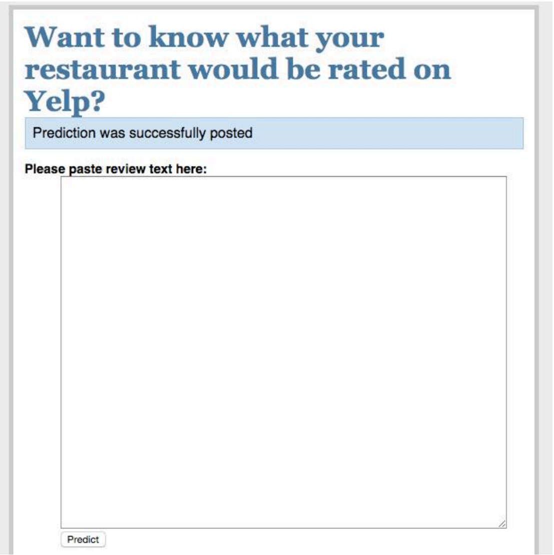
great little spot tucked into the kmart shopping center. I have been there on 2 occasions, it seems like they do rotate flavors. Good variety, yummy toppings. Happy to have this spot in the neighborhood. Clean and yummy! 5 stars from me!

This is a nice place close to home that's a good alternative to cold stone which sometimes seems to heavy. The staff was very friendly and was having fun talking to the kids. The yogurt was really good and they had a lot of flavors. I enjoyed it.

Predict

Yelp Explorers Final Project - Web App Demo

Screenshot of YouTube Video Demo of Web App



Want to know what your restaurant would be rated on Yelp?

Prediction was successfully posted

Please paste review text here:

Predict

Screenshot of Web App UI

Tools

To generate the features from Doc2Vec, we used Genesim's implementation of the doc2vec model because it had a lot of nice wrapper classes (Like LabeledLineSentece and Doc2Vec) which made feeding data into the Doc2Vec model easier while also maintaining a solid performance time (15 min to train the entire Yelp Corpus to do labeling).

To process all textual review data, we used Python's NLTK library, which provides a convenient suite of text processing libraries for classification, tokenization, stemming, and more. In particular, we used NLTK's SnowballStemmer, WordNetLemmatizer, and Stop Words Corpus for improving the power of our bag of words model. We used scikit-learn for SVM, Logistic Regression, and Random Forest Classifiers. Furthemore, we made use of sklearn GridSearchCV for finding the best hyperparameters for each classifier. This tool is particularly helpful in dealing with classifiers like SVM or Random Forest, which have multiple hyperparameters, since it will find the optimal combination. We also used scikit-learn's CountVectorizer for generating ngrams and the feature matrix for our bag of words model. Scikit-learn's CountVectorizer comes with optimizations for computation time that made it a much more suitable option when dealing with large amounts of review data than what we initially tried, which was implementing every step of bag of words manually.

We also used scikit-learn for feature selection. We used sklearn's TruncatedSVD for dimensionality reduction as seen in our results, and sklearn.feature_selection's SelectKBest library for selecting the best scoring features of a chi-squared test. We used pickle for serializing classifiers to avoid unnecessary recomputation. We also use a pickled classifier in making our web application, so that it can return results to the user quickly. We made use of EC2 for computing the large feature matrix. We used pandas dataframes for data cleaning, preparation, and exploration on initial datasets. We used Flask to create a single page web application which used the pickled classifier to illustrate the potential benefits our classifier could have for restaurants which are not on Yelp.

Lessons Learned

The aim of our project was to be able to accurately predict the star rating of a restaurant based purely on its textual reviews. After obtaining, cleaning and preparing our data, we tried two approaches for generating feature sets for restaurants; the first approach was to use a BoW model of the most common unigrams, bigrams, and trigrams (across all restaurant reviews), and the second was to use a trained Doc2Vec model. Then, using first the BoW feature model and then the Doc2Vec feature model, we trained a Logistic Regression classifier, SVM classifier, and Random Forest classifier to 1) predict the general sentiment towards a particular restaurant (as a sanity check of classifier performance), and 2) predict the star rating of that restaurant.

We found that using the Doc2Vec feature model generally lead to improved accuracy scores for all classifiers for both the sentiment prediction and rating prediction problems. For example, for the sentiment prediction problem, SVM's highest accuracy was .85 using the BoW feature model (when using 1000 features), and .88 using the Doc2Vec feature model. In addition, for both sentiment prediction and rating prediction, we found that SVM and Logistic Regression outperformed Random Forest (irrespective of feature model choice). For the rating prediction problem, SVM achieved an accuracy of .87, Logistic Regression achieved .84 and Random Forest achieved .77 using the BoW model. Using the the Doc2Vec model, SVM achieved an accuracy of .88, Logistic Regression achieved .87, and Random Forest achieved .78, where we define an accurate prediction as being within .5 of the actual rating. When we look at the BoW model, SVM also outperforms Logistic Regression in rating prediction, but when using Doc2Vec, they both perform comparably. We also use RMSE to quantify how far off our predictions are, and Logistic Regression with Doc2Vec interestingly gives the lowest RMSE of .28 (compared to .29 for SVM using Doc2Vec).

Ultimately, using our SVM and Logistic Regression classifiers with the Doc2Vec feature model, we were able to achieve our initial goal of accurately predicting restaurant star ratings based off of text reviews. Our high prediction accuracy and low RMSE values indicate that we can give a restaurant looking to objectively interpret feedback, whether it's text from questionnaires or reviews from other websites, a good idea of how they would rate on Yelp. To make this capability accessible to restaurant owners, we built a web app that takes as input any amount of text reviews for a particular restaurant, and using our SVM classifier, outputs a star rating prediction. A demo of this app can be viewed at this link: https://www.youtube.com/watch?v=LZgQLQNnzh4&feature=youtu.be.

Team Contributions

We all contributed equally to this project, with most of the work done through pair programming in group meetings.
Mukund - 25% - Worked on data exploration with word cloud, bag of words featurization, applying various machine learning classifiers, parameter tuning, generating visualizations.
Parul - 25% - Worked on initial data exploration/analysis, featurization, created Flask web application, visualization of results, and poster design.
Abhinava - 25% - Worked on Doc2Vec featurization, machine learning classifiers, generating visualizations
Ilakya - 25% - Worked on initial data exploration, analysis, and initial TF/IDF/bag of words featurization, helped with visualization of results, report write-up, and poster