# Project Report

Daniel Loi Kevin Arellano Claudio Sangeroki

December 2019

## 1 Abstract

Using nltk for preprocessing and scikit-learn for our training and testing, we were able to achieve around above 60% accuracies using different classifiers from a train set and test set that we split off on our own. After further experimentation and optimization, we settled on using sklearn's SVM classifier to train and predict our data as it yielded the highest accuracies from our train set and test set.

## 2 Approach

We used scikit-learn's library and its classifiers to test which classifier will yield us the most accuracy. Since there is no labelled test set given, we decided to derive our own by splitting the given train set into a train set and test set. After that, it was plenty of experimentation and optimization.

## 3 The Problem

Our task is to predict the star rating of a user review given the review itself and some meta-data about the review.

## 4 The Data

Our dataset came from yelp.com, a popular business rating platform where users can talk about their experience at a certain business and give it an overall 'star' rating from 1 to 5. Every instance in our dataset consisted of 6 attributes: stars, review, funny, cool, useful, and date. Where stars was the rating, review was the user review, date was the date that the review was posted and the rest were meta-data tags about the review that other uses could give a review if the found it funny cool or useful. We also noticed that the data was highly skewed towards 5 star reviews due to the realism of the reviews.

## 5 Thought Process

To decide on a classifier to use we had to look at the data available to us as well as the type of problem we were trying to solve. We decided that the only useful attribute in predicting the star rating of a review was the review itself because the other attributes were *about* the review and did not give us any information indicative of whether the user felt like they had a good experience at a certain business or not. Thus, since we were going to be working with text we knew we were going to have a lot of features because reviews range from at least a few sentences to multiple paragraphs.

## 6 Feature Abstraction

We first extracted the stars rating and reviews from the given train file. As we decided on only using two attributes from the set, we created a loop where it will take in the file and store those values in two different arrays for us to use. We decided on a bag-of-words approach to model our data because this approach helped quantify the vocabularies of the reviews in terms of weights and numbers making it easier for us to understand and create a model based off of it. We also preprocessed the data before we did any training which will be discussed in the next section.

# 7 Data Preprocessing

Our text review is initially just one long string of text. To fit our review into bag-of-words used the tfidfVectorizer from sklearn and created our own tokenizer for it. Below is a rough description of a tokenizer.

- In order for our bag-of-words data model to be effective we need to normalize our text as much as possible. We can do this by making each review lowercase and removing punctuation, special characters and numbers so that 'trouble', 'Trouble', 'trouble!' are all treated equally. Then we need to "stem" words so that different variations of the same words are treated the same. We did this using Porter's stemming algorithm which just cuts off the prefix and suffix of a word to obtain a 'root' so 'troubled', 'trouble' and '' would both be made into 'troubl'. Finally, we also removed stop words from reviews because they are considered non-informative words and only cause extra overhead. There are many lists of stop words out there and we decided to choose one of a moderate size.

Now we can pass our tokenizer to the tfidfVectorizer which will apply it to every review and then create 'vocabulary' from the given dataset that gives each word/feature a value based on its frequency and importance of the word to each instance. We also passed a minimum document frequency parameter X to the vectorizer because we only want word with frequency higher than X to be part of our vocabulary.

# 8 Tools Used

- scikit-learn library for machine learning
- Nltk for the porter stemmer
- json and csv to extract data from json and write predictions in csv
- sys to insert command line arguments (for files)
- Re(regex) for removing special characters, punctuation and numbers from text
- Other python libraries to help such as math

# 9 Experimentation

1. Count Vectorizers and TfIdf Vectorizers

   Initially we used a count vectorizer which only takes into account the word frequency and not the importance of each word to each instance. We found better performance with the tfidfVectorizer because it modeled our data better. This helped the data we were going to use for training immensely.

2. Classifiers

   We tested training the data with SVMs, Gaussian Naive Bayes, Multinomial Naive Bayes, Logistic Regression, K-Nearest Neighbor, and Decision Tree Classifiers. Out of all these classifiers, only two were worth noticing namely SVMs and Logistic Regression Classifiers. We would only choose between these two for final results but it was interesting to see how the others fared during training. Finally, we decided on using SVMs as it was consistently giving us the best results for our own training set and test set that we split from the given train set.

   **Highest results for each classifier:**

   | | |
   |---|---|
   | SVM | : 0.81764705882352942 |
   | Gaussian Naive Bayes | : 0.30588235294117646 |
   | Multinomial Naive Bayes | : 0.6882352941176471 |
   | Decision Tree | : 0.46294117647058826 |
   | KNN | : 0.51176700058823529 |
   | Linear Regression | : 0.6588235294117647 |

   We also had plenty of times where the accuracies of SVM and Logistic Regression would be exactly the same depending on the train data provided.

3. Lemmatization

   Lemmatization is essentially a better more sophisticated version of stemming and will try to return the actual root of a word instead of the pseudo-root that stemming finds. After replacing our stemming step with lemmatization instead we saw next to no improvement in accuracy and a lot more overhead so we stuck with stemming instead.

4. Min df

   We experimented with many values for min df for the tfidfVectorizer and found that 20 yielded the best results.

5. Grid search CV

   As a last effort to improve the accuracy of our classifier sklearn offers functionality to determine the best hyperparameters for our SVC such as the regularization value, the kernel, gamma etc. and evaluates them using cross fold evaluation. Ideally one would run all the parameter combinations offered sklearn but this is a very costly operation so only a few subsets of parameter values were analyzed and gridsearch CV determined default parameters had the best performance.

## 10 Challenges

We had trouble approaching the subject and had to learn new libraries and tools to solve this problem. It was a tiring process in the beginning but we picked up our pace and was able to solve the problem. Debugging using the new tools was tiresome and time consuming. We couldn't train our data using ALL of the given dataset since it had around 330000 values so we settled with using only 30000 values (which was the maximum amount allowed by our machine's memory). This would have yielded a higher accuracy in predicting the correct label for the test set in our opinion.

## 11 Results

Since SVM ended up with a consistent and high accuracy from our training model (averaging at around 70%), we ended up using that classifier to train our model. We had two CSV files that were outputs from different training sets and when compared, some of the values for 2,3,4 ratings were different but they both almost consistently predicted the 1 star rating and the 5 star rating the same way.

## 12 Conclusion

Ultimately, we chose to use an SVM with a linear kernel as our classifier for several reasons. First, it was returning the highest accuracy for our data-set. We believe that this can be attributed to the fact that we used a linear kernel, which works well on data-sets that are easily separable. We believed that this is the case for this data-set. By using Tf-idf as our feature selection method, we were better able to make our data-points more distinct, and consequently, more linearly separable. Furthermore, linear kernels can be trained quicker than other kernels, especially when there is a large number of features and data points. Due to the constraints of our machines as well as time, we decided that this was the best option moving forward.

## 13 Ideas for Future Work

We could have used better and more complex set of tools to help visualize the model further. We could have also trained with many other different sets such as an evenly distributed set where the data is not very skewed towards the 5 star rating.