Sentiment Analysis for Yelp Reviews: Star Prediction

Daniel Tang*
dt19@rice.edu
B.S. Computer Science, Rice University
Houston, Texas

Patrick Han*
pyh1@rice.edu
B.S. Electrical Engineering, Rice University
Houston, Texas

ABSTRACT

Yelp is an online business directory and crowd-sourced review forum on which customers can write about their experiences with various businesses ranging from restaurants to electricians. A business' popularity and growth can be greatly affected by the reviews on Yelp, and consumers are interested in a platform that paints an "honest" picture of businesses. Therefore it is highly desirable for both parties to have access to reliable and honest reviews. The star rating of a business is particularly important. In many cases, consumers may only consider businesses with star ratings above a particular threshold, rather than consulting the reviews in detail. Unfortunately, consumers can potentially leave misleading reviews in which the star rating does not reflect the sentiment of the review content itself. Whether as a joke, or as a mistake, these misleading ratings have the potential to impact a business negatively. Furthermore, it can be difficult for business owners to have these types of reviews removed. Therefore, in this paper we devise a system to evaluate review sentiment and predict the appropriate star rating. We utilize several classical methods from text analysis to build features including a bag-of-words approach and TFIDF (term frequency–inverse document frequency). We then apply several algorithms to predict star ratings based on the features built and produce relatively successful results especially for the more common label classes (1 and 5 star reviews). Finally, we discuss some of the issues we faced in our approach and outline how this work can be further extended.

CCS CONCEPTS

• **Information systems** → **Sentiment analysis**.

KEYWORDS

Sentiment Analysis, Machine Learning, Regression, Classification, TFIDF, Bag of Words, Random Forest, Naïve Bayes, Logistic Regression, KNN, Neural Network

ACM Reference Format:
Daniel Tang and Patrick Han. 2019. Sentiment Analysis for Yelp Reviews: Star Prediction. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

In their simplest form, Yelp reviews consist of a written portion accompanied by a star rating and extra rating buttons "Useful", "Funny", and "Cool" as shown in **Figure 1**. There is a 5000 character limit on the written portion and users may choose to add

*Both authors contributed equally to this research.

Unpublished working draft. Not for distribution.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
Conference'17, July 2017, Washington, DC, USA
© 2019 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

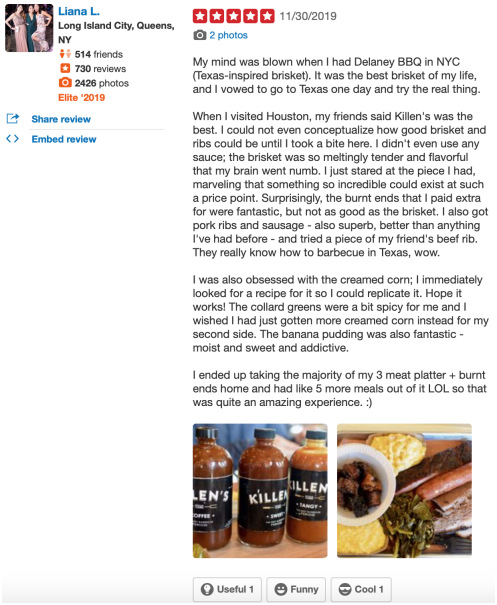

Figure 1: Example Yelp review for Killen's BBQ, via Yelp.com

accompanying pictures. An example of the consumer user interface for leaving a review is depicted in **Figure 2**. Once the written

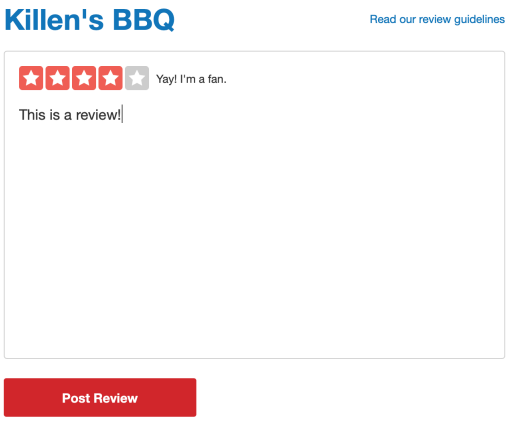

Figure 2: Example review prompt, via Yelp.com

portion is finished, users simply select a star rating from one of five choices to complete and post the review. As discussed earlier, we are particularly interested in the star rating as it correlates to the sentiment of a review. Sometimes the review sentiment does not necessarily match the star rating, and low star ratings have the potential to greatly impact a business in terms of popularity. An example of such a rating is shown in **Figure 3**. In this case, the user's sentiment clearly implies a 4 or 5 star review.

2 PRIOR WORK

There have been previous attempts to classify by grouping words into "positive" and "negative" associations [5]. Additionally, there

Figure 3: Example misleading review, via Yelp.com

has been interest in identifying restaurant features using sentiment analysis [3]. Many attempts seem to group reviews into the 2 categories of "positive" and "negative" rather than the 5 distinct star categories.

3 DATA

The data used for this project was officially provided by Yelp [6]. The sentiment analysis dataset contains several JSON files of interest:

- `business.json`: Info about business being reviewed, avg star rating
- `review.json`: Info about review itself, with star rating + user ID
- `user.json`: Information about user who wrote the review
- `tip.json`: Short tidbits that reviewers post about a business

Of course, `review.json` is the file of primary interest to us. The JSON file is approximately 5 GB uncompressed and contains about 6.6 million reviews. Each review contains the following features:

- `business_id`
- `cool, useful, funny`
- `date`
- `review_id`
- `stars`
- `text`
- `user_id`

The notable features for our use case include the cool, useful, funny ratings, the body of review text, and of course our label, the star rating.

3.1 Data Exploration

Before any sort of heavy pre-processing and modeling, we decided to explore and analyze the data as is to see if we could find any patterns. For this project, we decided to use a subset of the entire dataset (150k reviews), mostly for practical computing reasons. In doing so, we made sure that our chosen subset reflected the label distribution of the full data set (**Figure 4 and Figure 5**). Those distributions are (In order from 1 to 5 stars):

- **Subset:** 14.99%, 8.11%, 11.06%, 21.97%, 43.87%
- **Full:** 14.64%, 8.0%, 11.05%, 21.99%, 44.32%

Notably, the distribution of reviews is highly imbalanced. Over half of all reviews result in a 4 or 5 star rating. The number of 1 star reviews follows in frequency. A possible explanation for this is the tendency for only passionate users to leave reviews. I.e. a very satisfied or very dissatisfied customer.

Another avenue of interest was the possibility of relating the review

Unpublished working draft. Not for distribution.

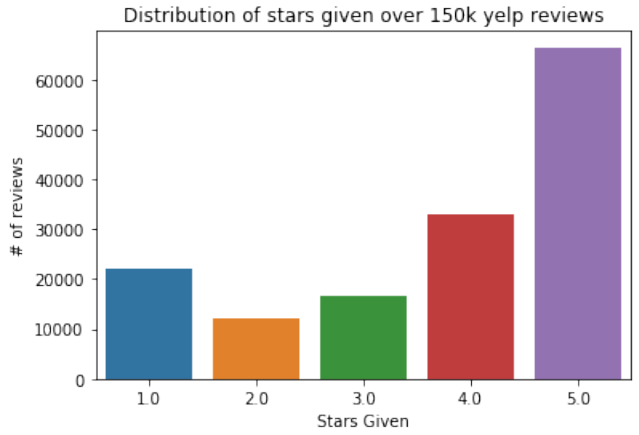

Figure 4: Distribution of data subset

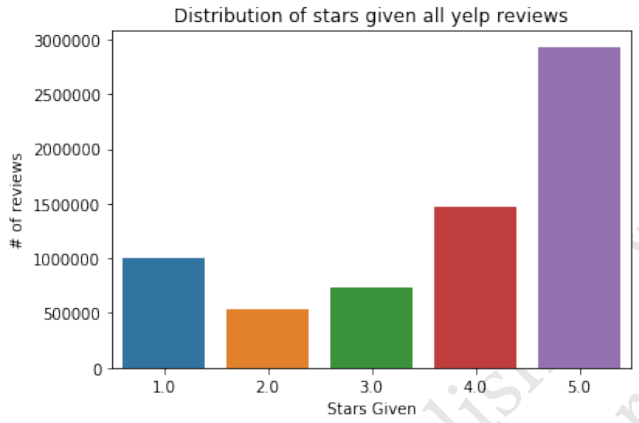

Figure 5: Distribution of all data

length (word count) to the star rating (**Figure 6**) As seen, there

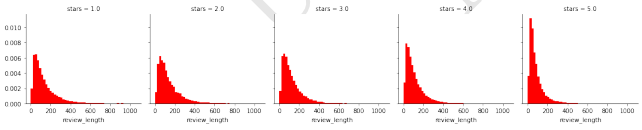

Figure 6: Review length distribution per star rating

doesn't seem to be a relationship between how long a review is, and what star rating a user gives. The distribution is similar among all label categories. Next, we explored the distribution of the "funny, cool, useful" ratings in relation to stars (**Figure 7**) Again, there doesn't seem to be a clear relationship between the star rating and the "funny, cool, useful" ratings. We also constructed a correlation (Pearson) matrix (**Figure 8**) The matrix demonstrates a strong correlation among all three of the "funny, cool, useful" counts, but no relationship with the star rating. Based on this analysis, we decided not to use the "funny, cool, useful" ratings for our project.

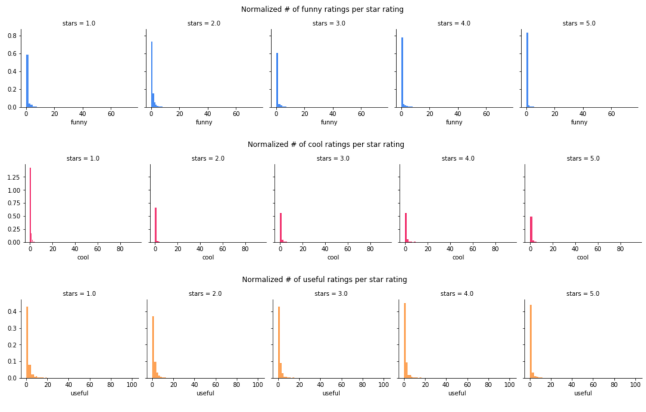

Figure 7: Rating of funny, cool, useful per star rating

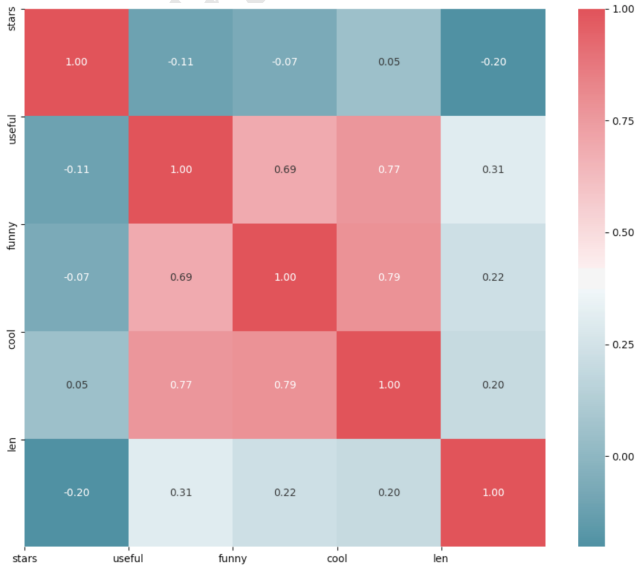

Figure 8: Correlation matrix among labels and potential features

3.2 Preprocessing

Before vectorizing our words, we did some standard preprocessing. We used Python's Natural Language Toolkit (NLTK), a suite of libraries and programs for symbolic and statistical natural language processing for English. Specifically, we used *stem.PorterStemmer*, *tokenize.word_tokenize*, and *corpus.stopwords*. Stop words are the most common words in the dictionary, like "The", "and", and "a". These words were filtered out of our documents because they provide no explicit meaning. Uppercase letters were transformed into their lowercase variants, numeric characters were transformed into their string variants, and punctuation was filtered out. Stemming simplifies words to their roots, or stems. The Porter Stemming Algorithm (**Figure 9**) was applied on each word in our documents to remove the commoner morphological and inflectional endings in English words.

Unpublished working draft. Not for distribution.

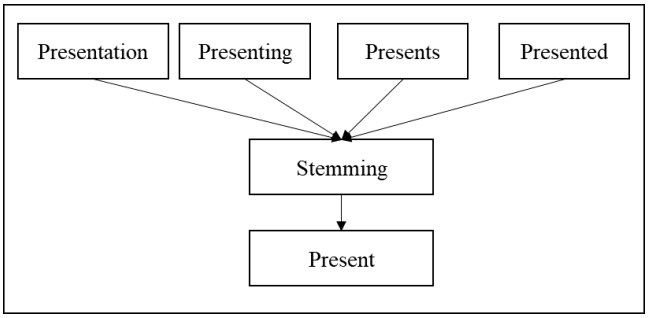

Figure 9: Blackbox of the Porter Stemming Algorithm [1]

4 BAG-OF-WORDS VECTORIZATION

One of the simplest ways to vectorize paragraphs is to use a "bag-of-words" approach. Essentially, each paragraph is represented as an n-length vector where n is the number of unique words in the corpus. Each entry of the vector denotes the number of occurrences of that word in that document (**Figure 10**). Taking this approach,

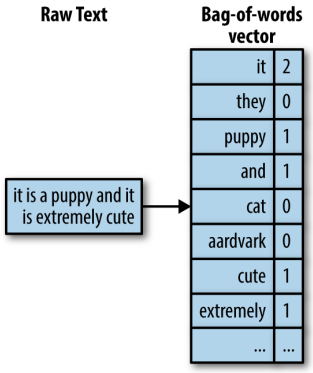

Figure 10: Bag-of-words approach, via Zheng & Casari [2]

we vectorized the pre-processed and cleaned data in order to test several predictive algorithms. Those results are shown here:

- **Random Forest (110 est):** CV avg acc: 58% | Test acc: 58%
- **KNN (k=20):** CV avg acc: 51% | Test acc: 51%
- **SVM (elastic net)** CV acc: 64% | Test acc: 64%
- **Naive Bayes** Test acc: 60%

Cross-validation tuning was performed over 8-10 folds. The train/test split used was 80/20. Equally important are the precision, recall, and F1 scores for these classifiers (**Figures 11,12, 13, 14**) Analysis of these metrics reveals great performance on 1 and 5 star reviews. There is a strong reason to believe this is related to the imbalanced class distribution. Additionally, KNN performs noticeably poorer compared to the other algorithms. This doesn't seem to be a surprise given how high-dimensional our data is (nearly 60k features). The Curse of Dimensionality seems to be at play here. Relevant cross validation tuning plots are also depicted in (**Figures 15, 16**). We essentially chose parameters that maximized average cross-validation accuracy while minimizing model complexity.

Star	Precision	Recall	F1-score
1.0	0.71	0.71	0.71
2.0	0.51	0.02	0.04
3.0	0.44	0.06	0.11
4.0	0.41	0.21	0.28
5.0	0.59	0.96	0.73

Figure 11: Random Forest results (110 est)

Star	Precision	Recall	F1-score
1.0	0.66	0.34	0.45
2.0	0.18	0.03	0.04
3.0	0.35	0.03	0.06
4.0	0.38	0.15	0.22
5.0	0.52	0.95	0.67

Figure 12: KNN results (k=20)

Star	Precision	Recall	F1-score
1.0	0.70	0.83	0.75
2.0	0.42	0.17	0.24
3.0	0.46	0.23	0.31
4.0	0.48	0.38	0.42
5.0	0.71	0.90	0.80

Figure 13: SVM results (elastic net loss)

Star	Precision	Recall	F1-score
1.0	0.62	0.71	0.66
2.0	0.32	0.24	0.27
3.0	0.38	0.32	0.35
4.0	0.44	0.60	0.51
5.0	0.80	0.69	0.74

Figure 14: Naive Bayes results

5 TFIDF VECTORIZATION

TFIDF stands for "Term Frequency - Inverse Document Frequency". Essentially, the features it develops are a measure of how important a word is to a document in a given corpus. Like our bag-of-words approach, each feature corresponds to a unique word from the corpus. The formula for calculating TFIDF is:

$$W_{x,y} = tf_{x,y} * \log\left(\frac{N}{df_x}\right)$$

Where:

- $W_{x,y}$: for term x in document y

Unpublished working draft. Not for distribution.

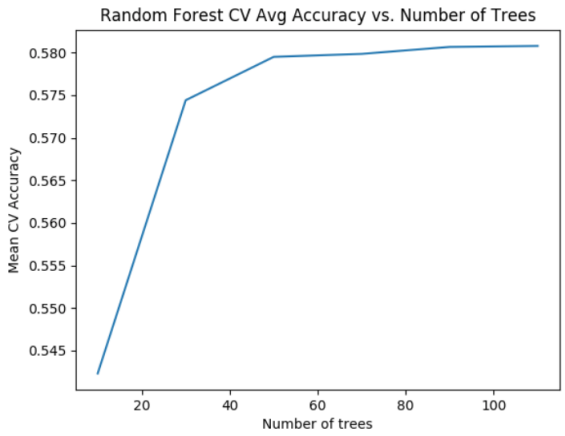

Figure 15: Random Forest CV curve

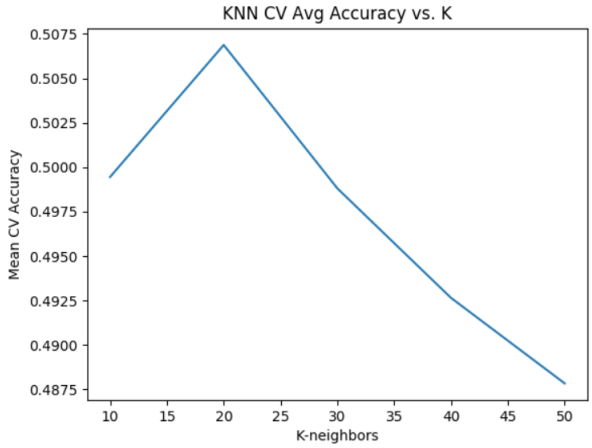

Figure 16: KNN CV curve

- $tf_{x,y}$: frequency of x in y
- df_x: number of documents containing x
- N: number of documents

The dimensionality of a TFIDF representation is the same as our bag-of-words approach. The difference is of course, the relative weighting of each vector. Again, we tested several predictive algorithms on the vectorized data. We opted to try Logistic Regression instead of KNN due to the performance issues related to dimensionality in the bag-of-words approach. Our cross validation for Logistic Regression tuned the max number of iterations used. Those results are shown here:

- **Random Forest (60 est):** CV avg acc: 58% | Test acc: 58%
- **Logistic Regression (400 iter):** CV avg acc: 66.23% | Test acc: 66.58%
- **SVM (L2)** CV acc: 63% | Test acc: 63.04%
- **Naive Bayes** Test acc: 54.27%

Again, corss-validation tuning was performed over 8-10 folds and the train/test split was 80/20. Our other performance metrics are detailed in **Figures 17,18, 19, 20**. The results indicate relatively strong performance from the Logistic Regression model across all classes, and especially with 1 and 5 star reviews. The SVM model comes in second, with a fairly reasonable F1-score in all classes. Both the Naive Bayes and Random Forest models perform poorly, with very little predictive power for 2 and 3 star reviews. Overall, the TFIDF vectorization and corresponding algorithms had higher accuracies than those found in our bag-of-words approach.

Star	Precision	Recall	F1-score
1.0	0.70	0.69	0.70
2.0	0.46	0.03	0.05
3.0	0.45	0.08	0.13
4.0	0.4	0.21	0.27
5.0	0.59	0.95	0.73

Figure 17: Random Forest results (60 est)

Star	Precision	Recall	F1-score
1.0	0.73	0.82	0.77
2.0	0.46	0.28	0.34
3.0	0.47	0.36	0.41
4.0	0.52	0.46	0.49
5.0	0.76	0.86	0.81

Figure 18: Logistic regression results (400 max iter)

Star	Precision	Recall	F1-score
1.0	0.64	0.88	0.74
2.0	0.42	0.09	0.15
3.0	0.51	0.20	0.29
4.0	0.48	0.21	0.29
5.0	0.66	0.96	0.78

Figure 19: SVM results (L2 loss)

Star	Precision	Recall	F1-score
1.0	0.81	0.53	0.64
2.0	0.00	0.00	0.00
3.0	0.15	0.00	0.00
4.0	0.33	0.14	0.19
5.0	0.54	0.98	0.69

Figure 20: Naive Bayes results

Unpublished working draft. Not for distribution.

6 DISCUSSION OF RESULTS

Overall, the results we got were promising for relatively simple vectorization and modeling. SVM and Logistic regression seem to have the best performance in general. As discussed earlier, KNN suffers heavily due to the dimensionality of this project. Additionally, the relatively poor performance of Naive Bayes suggests that the independence assumption is not necessarily sound in this context. Class imbalance heavily affects performance on individual classes. Across nearly all algorithms, the F1-score was superior in the case of 1 and 5 star reviews.

When it comes to our vectorization techniques, one of the biggest problems that arises is a lack of semantics. Neither the bag-of-words approach nor TFIDF capture any sort of meaning among groups of words. For example, this becomes a problem with the phrase "not bad". While a "not bad" might seem like a 4 star rating, our techniques only measure frequency. Thus, the algorithm may see the word "bad" and decide to classify the review as 1 star.

One big assumption made in this project that should be addressed is the presence of "misleading reviews" in our dataset. In short, it's highly likely that the type of reviews we are trying to combat exist somewhere in the training set. Of course, there is no way we can go in and manually verify each sample. Instead, we assume that these misleading cases only represent a small portion of the dataset, and that on average, we can expect honest reviews.

7 EXTENSION OF THIS WORK

We built a web application to provide a working prototype for our project. The backend was built in Python using a Flask microframework. The frontend was built using HTML/CSS. The web app allows the user to input any arbitrary review and a choice of 3 of our best performing models, or the average of the 3 models. Upon submitting, our web app will predict the classification of the review and display that rating. A demo of our web application is shown in **Figures 21,22, 23**.

Figure 21: Random Review from Yelp

Additionally, we discussed earlier than the methods we use currently do not capture semantics or meaning between groups of words or paragraphs as a whole. There are more sophisticated ways of representing our reviews as vectors. For example, word and document embeddings are powerful tools to perform this type of task

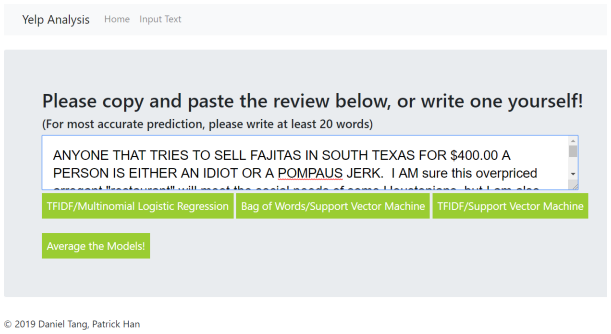

Figure 22: The Web Application

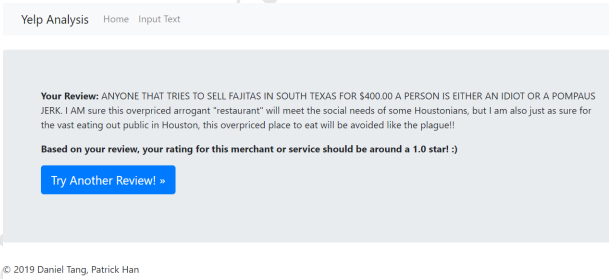

Figure 23: Average Prediction for our Random Review

[4]. However, a great amount of computation is needed up front in order to compute these embeddings from a given corpus.

8 CONTRIBUTIONS

In terms of modeling, Patrick tested using the bag-of-words features while Daniel used TFIDF. Daniel preprocessed the data and developed the web application. Patrick performed the exploratory analysis of data.

9 PROJECT CODE

https://github.com/dtang5/DSCI303_Final_Project

ACKNOWLEDGMENTS

Thank you Dr. Souptik Barua and Dr. Akane Sano for teaching this course.

REFERENCES

[1] Amol C Adamuthe. 2018. *Hierarchical Text Classification using Dictionary-Based Approach and Convolutional Neural Network.* https://www.researchgate.net/figure/Stemming-process-Algorithms-of-stemming-methods-are-divided-into-three-parts-mixed_fig2_324685008/
[2] Amanda Casari Alice Zheng. 2018. *Feature Engineering for Machine Learning* (1st ed.). Principles and Techniques for Data Scientists, Vol. 1. O'Reilly Media. (book).
[3] Yi Zhang Yunong Cao Boya Yu, Jiaxu Zhou. 2017. Identifying Restaurant Features via Sentiment Analysis on Yelp Reviews. *arXiv* (2017).
[4] Tomas Mikolov Quoc V. Le. 2014. Distributed Representations of Sentences and Documents. *arXiv* (2014).
[5] David Robinson. 2016. *Does sentiment analysis work? A tidy analysis of Yelp reviews.* http://varianceexplained.org/r/yelp-sentiment/
[6] Yelp.com. 2019. *Yelp Dataset Challenge.* https://www.yelp.com/dataset/challenge

Unpublished working draft. Not for distribution.