Yelp Rating Prediction From Reviews

Yezhen Gong
yezheng@andrew.cmu.edu

Ziyan Liu
ziyanl@andrew.cmu.edu

Wenna Qin
wennaq@andrew.cmu.edu

Abstract

Sentiment analysis is an important natural language processing technique used for interpreting and classifying emotions in subjective data, yet there has been few works on multi-class classification in sentiment analysis. This study illustrated a multi-class classification experiment of ratings on Yelp Review Dataset based on review's comment text. In the paper, we compared Bag of Words and Word Embedding for feature extractions, and applied them to Logistic Regression, SVM and LSTM models to optimize the performance on accurate predictions.

1 Introduction

Yelp is convenient for human beings to read reviews from other customers and have a preliminary idea of a restaurant. We can easily distinguish which comments are positive and which are negative. It is actually a process of "receiving" the texts, analyzing the sentence structures, interpreting the meaning of each word, and finally making a judgement. In this project, we want to build a prediction model that outputs the ratings given the review texts. From real-life experience, a person does not need to read every single word of a paragraph, but can still understand the central meaning. Based on this idea, and in order to achieve the goal, we first need to find some attributes that contribute the most for people to making judgements.

Predicting Yelp scores based on review texts is important because texts are more revealing than simple numeric ratings. There can be mis-rating cases where the review is very negative while rating is high, due to the carelessness of the user. We hope that our prediction model could help correct these cases, and better inform the companies/restaurants of consumer attitudes toward their service.

2 Data

Our dataset contains three sets of data - training, testing and validation sets. However, only the training data has the label `stars`. Therefore, we decided to use the training data only, and divided it into three sets to do validation and test on. In the dataset, there are 36693 samples in total, of which 60% were used for training, 20% for validation, and the rest 20% for final testing. Each contains stars, the name of restaurant, the review text, date of review, categories of the restaurant, and 8 more numerical attributes, which are `useful`, `funny`, `cool`, `longitude`, `latitude`, `nchar`, `nword` and `sentiment_score`. Further, the dataset also provides us with a list of vocabulary and counts of the words in each review text. It is worth noting that the dataset is skewed towards higher ratings, as over 60% of samples have `stars` to be 4 or 5.

Since we want to find a subset of attributes that contribute most to the prediction label, we did a correlation analysis on the data. It turns out that `sentiment_score` has the highest correlation with `stars`. It is calculated based on AFINN lexicon, and ranges from -5 (very negative) to +5 (very positive). `nchar`, `nword`, and `cool` also have relatively high correlations scores with the label. We also explored the correlation matrix among the numerical attributes, and found that `nchar` and `nword` are highly correlated as expected, and that `useful`, `funny`, `cool` also have pairwise high correlation.

Preprint. Under review.

3 Related Work

The idea of using texts to perform multi-class classification has been previously explored by Gayatree, Yogesh and Amélie[1].In their study, they set out to model the setup - they proposed the idea of learning to predict ratings based on review's text alone, because free-text reviews are difficult for computer systems to understand, analyze and aggregate. In addition, they manually annotated a training set of approximately 3400 sentences with both category and sentiment information. They were able to make better predictions on multiple sentiment classes based on the review texts.

Other researches have been focusing on binary(positive and negative) classifications utilizing different methods. Hemalatha and Ramathmika[2] experimented tokenization of texts, speech tagging, in which token is tagged to its part of speech and only the adjectives are considered for the classification purpose, and investigated several classification algorithms, including SVM, Naive Bayes and Logistic Regression.

However, very few studies focused on comparing the effectiveness of combinations of different algorithms and feature extraction techniques in a multi-class classification context. Therefore, we decided to explore this topic thanks to the inspiration by related works mentioned above.

4 Methods

4.1 Features

4.1.1 Bag of Words

To extract numerical features from raw texts, we chose to use the Bag of Words (BoW) representation due to its simplicity and flexibility. Five sets of features were generated based on the BoW model with different design choices.

BoW 1 A basic unigrams model was created using the TF-IDF (Term Frequency – Inverse Document Frequency) Vectorizer. All characters are converted to lower case, all punctuations and extra whitespaces were removed, and strings were tokenized. The TF-IDF scoring measure was chosen over simple token counts because it assigns higher scores to informative words that appear frequently in one review but rarely across the corpus. Also, it brings all the features to the same range, facilitating later model training. This method generated over 30,000 unique features in total, out of which the top 1000 words ordered by term frequency across the corpus were selected.

BoW 2 To further reduce the noise and dimensionality of data, we expanded contractions and performed stemming with Snowball Stemmer in addition to the preprocessing steps in the basic unigrams model, and terms with document frequency strictly lower than 0.001 were excluded from the vocabulary, which produced 3451 features. Since terms that are more correlated with the target variable, stars, should be more relevant, we computed the Pearson correlation coefficient between each term and stars and selected the top M (100, 500, 1000) unigrams with the highest correlations.

BoW 3 Since a significant portion of the unigrams most correlated with stars were adjectives and verbs, we attempted to retain only adjectives and verbs and applied the same preprocessing and tokenization methods as BoW 2, which resulted in a vocabulary of size 1785. The top 500 features most correlated with stars were selected based on Pearson correlation coefficients.

BoW 4 Because stemming is a crude heuristic process that reduces words to their root forms by chopping off the ends, we also replaced stemming with lemmatization in the hope of obtaining better features. Besides the regular preprocessing and tokenization, we obtained part-of-speech tags for the tokens through the NLTK part-of-speech tagger, converted those tags to WordNet tags, and applied the WordNetLemmatizer to each of the words. Similar to BoW 2, Pearson correlation coefficients were calculated and the top 500 unigrams ordered by correlations with stars were selected.

BoW 5 One drawback of unigrams models is that they completely ignore the order and structures of words in texts. While using the same preprocessing procedure as BOW 2, we allowed the TfidfVectorizer to extract both unigrams and bigrams to capture more semantics and ranked all the

produced features by their correlations with stars. The top 500 features that are most correlated with stars were selected.

4.1.2 GloVe Word Embedding

Since the BoW models retains little semantics and structure, we decided to look into word embedding. Word vectors provide information about the relationship between words through arithmetic calculations with vectors. We first considered training our own word vectors, as what we did for Bag of Words to get our own important features, but due to the limit of time and considering the mass amount of text data required to achieve this goal, we decided to use some appropriate pre-trained word vectors.

From all available options there, we chose GloVe Embedding, as it provides direct access to the vectors and thus easy to apply to our project. We experimented with two different vector sets, the first one is Wikipedia+Gigaword5, and the second one is Twitter texts. It turns out that vectors from twitter had better performance, and it is understandable because in reviews, it is likely that people will use non-standard English and colloquial expressions in their Yelp reviews, while Wikipedia is usually more formal compared to Twitter. To be specific, we choose twitter embedding that is 100 dimensional in order to balance between performance and time.

4.2 Models

4.2.1 Logistic Regression

A multinomial logistic regression model with L2 regularization was fitted using all sets of BoW features. The built-in Stochastic Average Gradient (SAG) solver in the Sklearn library was chosen to minimize the L2-regularized cross-entropy loss defined as:

$$\mathbf{L}(\boldsymbol{\theta}, \mathbf{X}, \mathbf{y}) = \frac{1}{2}\|\boldsymbol{\theta}\|^2 - C\sum_{i=1}^{n}\sum_{k=1}^{K}\mathbf{1}\{y^{(i)} = k\} logP(y^{(i)} = k \mid x^{(i)}; \theta),$$

where K is the number of classes, $\mathbf{X}$ represents the $n \times p$ input data matrix, $\boldsymbol{y}$ the vector of observed values of stars, $f_{\boldsymbol{\theta}}(\mathbf{x}) = \sigma(\hat{\boldsymbol{\theta}} \cdot \mathbf{x})$ the logistic model where $\hat{\boldsymbol{\theta}}$ is the vector of current parameter values, and the positive scalar C controls the strength of regularization. The inner summation represents the loss at one data point $(\boldsymbol{X_i}, y_i)$:

$$l(\boldsymbol{\theta}, \boldsymbol{X_i}, y_i) = \sum_{k=1}^{K} 1\{y^{(i)} = k\} logP(y^{(i)} = k \mid x^{(i)}; \theta)$$

SAG optimizes finite sum of smooth convex functions and was preferable than other methods because it can obtain the convergence rates of the Full Gradient (FG) method while preserving the low iteration complexity of Stochastic Gradient (SG) methods (Schmidt et al., 2016, p. 86) [3]. Define $g_i(\boldsymbol{\theta}) = \frac{1}{2}\|\boldsymbol{\theta}\|^2 + \frac{C}{n}\sum_{i=1}^{n} l_i(\boldsymbol{\theta})$. The SAG iterations take the form

$$\boldsymbol{\theta}^{k+1} = \boldsymbol{\theta}^k - \frac{\alpha_k}{n}\sum_{i=1}^{n} y_i^k,$$

where, at iteration k, α_k is the step size, and a random index i_k is selected and

$$y_i^k = \begin{cases} \nabla_{\boldsymbol{\theta}} g_i'(\boldsymbol{\theta}^k), & \text{if } i = i_k \\ y_i^{k-1}, & \text{otherwise} \end{cases}$$

4.2.2 Linear SVM

A linear support vector machine in scikit-learn implemented with LIBLINEAR was applied as well for classification using squared hinge as the loss function. The model was fitted using all sets of BoW features. To save computational costs, we chose the One-vs-Rest strategy that trained five binary classifiers for each of the five classes. The primal can be formulated as the following optimization problem:

$$\min_{\boldsymbol{w}} \left[\frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^{n}\max(1 - y_i\boldsymbol{w}^T\boldsymbol{x}_i, 0)^2\right],$$

and the corresponding dual problem is:

$$\min_{\boldsymbol{\alpha}} f(\boldsymbol{\alpha}) = \frac{1}{2}\boldsymbol{\alpha}^T Q \boldsymbol{\alpha} - e^T \boldsymbol{\alpha}$$
$$\text{subject to } 0 \leq \alpha_i \leq U, \forall i$$

LIBLINEAR solves SVM with a dual coordinate descent method that was shown to be much faster for large datasets than LIBSVM (Hsieh et al., 2008, p. 411) [4], thus suitable for our case. For L2-SVM, $U = \infty$ and $D_{ii} = \frac{1}{2C}$.

4.2.3 LSTM

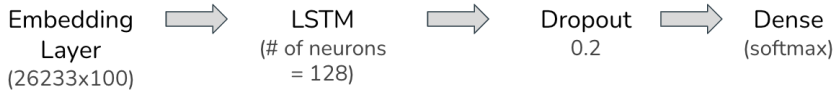

Figure 1: LSTM Model Layers

We further fitted a long short-term memory (LSTM) model, which is a recurrent neural network architecture. We map each Yelp review to a vector, so that the texts are encoded as real-valued vectors in high dimensional space, and the similarity between words can be measured by a distance metric in the vector space. After normal preprocessing and lemmatization, we tokenized the corpus and kept the most frequent 5000 words using Keras' tokenizer, and padded or truncated sequences to the same length of 100. An embedding matrix was constructed based on the GloVe word vectors pre-trained on Twitter texts. The first layer was an embedding layer seeded with the GloVe embedding weights, which was followed by an LSTM layer with 128 hidden units, where we make the dropout rate to be 0.2 in order to reduce overfitting. Finally, a Dense output layer with softmax activation function was applied to make predictions.

Since we are solving a multi-class classification problem, we chose to use the categorical cross entropy loss function. We used softmax as activation function, and the loss is defined as follows:

$$CE = -\sum_{i}^{} \sum_{k=1}^{5} \mathbf{1}(y_i = k) \log(\sigma(z)_i)$$

Because we are working with sparse representations, we chose Adam to be the solver. It uses the first and second moments of the gradient to update the parameters (Kingma et al., 2015)[5]. Denote the first moment at time t by m_t, and second moment by v_t. With default step size $\alpha = 0.01$, exponential decay rates $\beta_1 = 0.9$ and $\beta_2 = 0.999$ (Keras)[6], and let g_t denote the gradient with respect to the objective function, the first and second moments update as follows:

$$m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$$
$$v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t$$

Then we correct the bias of the moments by

$$\hat{m_t} = \frac{m_t}{1 - \beta_1^t}$$
$$\hat{v_t} = \frac{v_t}{1 - \beta_2^t}$$

and the final update rule for the parameters is given by

$$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \frac{\hat{m_t}}{\sqrt{\hat{v_t}} + \epsilon}$$

Figure 2 shows the loss progression in 30 epochs, using Twitter texts as the embedding layer.

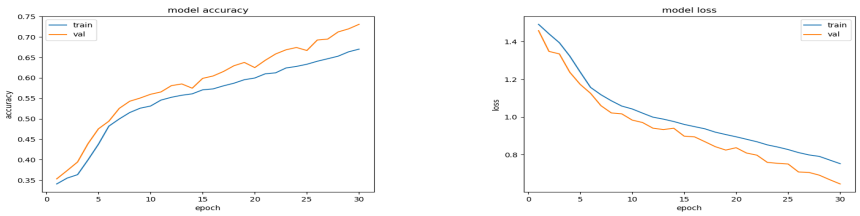

Figure 2: Accuracies(left) and Loss Progression(right) of LSTM with Twitter Texts in 30 epochs

4.3 Model & Metric Selection

The evaluation metrics we choose are accuracy calculated by $\frac{\#ofMatches}{\#ofSamples}$ and running time. Running time is an important factor to consider when dealing with large datasets, so efficiency becomes an important part of performance evaluation.

In addition to the three models discussed above, we performed multi-class classification for the Yelp ratings with random forest and Naive Bayes classifier. Random forest was chosen initially because it can offer insights into the importance of features as well as make predictions. Besides the BoW features, we also included sentiment score, nchar, longitude, and latitude in the model as the correlation matrix showed that they were associated with stars. Although these four features were among the most relevant features according to feature importance for random forest, their presence added computational burden and failed to considerably improve models' performance. Hence, we chose to focus on generating features from texts for our subsequent experiments.

After training all models except for LSTM on the seven sets of BoW features, we compared their performance based on validation accuracy and running time and determined that the multinomial logistic regression and linear SVM were significantly better than the others. We trained the LSTM model using twitter texts as embedding, but the training cost was extremely high, and it did not converge after 30 epochs.

5 Results

Figure 3: Top 100 common unigrams in raw texts (left) and in preprocessed texts (right)

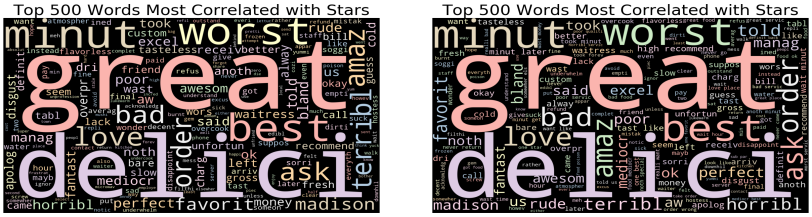

Figure 4: Top 500 unigrams (left) and unigrams/bigrams (right) most correlated with stars

5.1 Features

Figure 3 shows the importance of preprocessing step as the most common words in raw texts are stopwords that carry little meaning. It is worth noting that no negative words appear on the left

plot, and the right plot exhibits a similar pattern as only one negative word, "bad", made it to the list of most common words even after text preprocessing, which matches the previous finding that the distribution of stars is skewed towards higher ratings. Besides the apparently positive words, the presence of nouns such as "food", "place", "service", "time", and "taste" indicates the factors that customers tend to consider when writing a review. Figure 4 presents the features generated from BoW2 and BoW5 mentioned previously in 4.1.1. Although the majority of most correlated words were unigrams, the bigrams were able to capture more semantics. For example, the presence of "great service", "poor service", "bad service", and "horrible service" is likely to be more informative than a single word "service". Also, bigrams like "never return" and "one star" contains information that cannot be discovered with unigrams and should ideally be captured by the models as indications for low ratings.

Table 1 and 2 below presents the performance of logistic regression and linear SVM on all seven sets of BoW features. The results for random forest and Naive Bayes classifier are not presented since they all follow a similar pattern. Both the random forest and Naive Bayes classifiers reached the highest validation accuracy with BoW 1, the basic unigrams model with top 1000 most frequent terms, whereas the validation accuracy using BoW 2 (M=1000) was slighly higher than that using BoW 1 for logistic regression and linear SVM. As expected, more information were preserved with a larger number of features, thus yielding higher accuracy but also a higher risk for overfitting. The random forest model overfitted significantly as it classified all training samples perfectly with 1000 features, and the training accuracy remains above 99% for the other smaller sets of features. Although limiting the maximum depth helped to reduce overfitting, the validation accuracy dropped as well. In contrast, the models' validation accuracies were the lowest with BoW 2 (M=100) features, suggesting that 100 features were not sufficient to generate good results even if they were most correlated with the target variable. Similarly, the 500 features with only adjectives and verbs were not able to capture the sentiments underlying reviews very well, which demonstrated that seemingly neutral word classes such as nouns did contribute to models' understanding of sentiments in this case.

We expected that lemmatization (BoW 4) would improve the performance of the models, but the results showed the contrary. Keeping all other procedures the same, the validation accuracy dropped by a small amount after changing from stemming to lemmatization for all the models. Moreover, obtaining the part-of-speech tags and lemmatizing almost tripled the time for preprocessing. Lastly, BoW 5 that allowed for both unigrams and bigrams failed to improve the performance as well. We also tried to include only bigrams, which led to poorer performance. Although the bigrams highly correlated with stars should contain more information in theory, they might be specific to the training set and hinders the models' ability to generalize. Considering all the factors mentioned above, we decided to use BoW 2 with M=500 or 1000 as the feature representations for running subsequent experiments depending on specific purposes since the former runs faster while the latter gives better results.

	Training Accuracy	Validation Accuracy	Training Time
BoW 1	0.6169	0.5542	18.6497
BoW 2 (M=100)	0.5120	0.5066	1.8204
BoW 2 (M=500)	0.5763	0.5473	8.4428
BoW 2 (M=1000)	0.6110	0.5597	23.3478
BoW 3	0.5621	0.5247	9.3310
BoW 4	0.5757	0.5420	8.2699
BoW 5	0.5707	0.5435	7.9307

Table 1: Logistic Regression with Different BoW Schemes

5.2 Models

For logistic regression, using BoW2 with 1000 features gave us the best performance. Although the training time is longer than the other methods, the training accuracy reaches 0.6110, and validation reaches 0.5597. Then, we use it to train models with different regularization strength C. The smaller C is, the stronger the regularization becomes. In general, the more weight the regularization term has, the less overfitting there will be. When we tried the model with $C = 0.1$, i.e. a very strong regularization, it converged within 20 iterations and uderfitted with poor performance that the training

	Training Accuracy	Validation Accuracy	Training Time
BoW 1	0.6142	0.5476	0.8090
BoW 2 (M=100)	0.5125	0.5046	0.2452
BoW 2 (M=500)	0.5771	0.5456	0.4940
BoW 2 (M=1000)	0.6199	0.5508	0.6354
BoW 3	0.5594	0.5123	0.4079
BoW 4	0.5759	0.5399	0.4844
BoW 5	0.5782	0.5374	0.4628

Table 2: Linear SVM with Different BoW Schemes

accuracy was 0.5071 and validation accuracy was 0.4918. It turned out that when $C = 0.5$, we received the highest validation accuracy, while keeping it not overfitted.

Although the training and prediction time of the Naive Bayes model were the lowest out of all the models, its performance was roughly at a similar level as random forest with the best possible validation accuracy at 0.5119 achieved with BoW 1 features, which was not as promising as the logistic regression model. This aligned with our expectations because the assumption of conditional independence between features apparently did not hold in such a sequential data context.

In general, the performance of linear SVM was extremely close to that of logistic regression, and training linear SVM with LIBLINEAR on our training data takes less than one second even with 1000 features. Similar to logistic regression, linear SVM reached the highest training accuracy and validation accuracy when it was trained on data with 1000 features generated by the BoW 2 scheme. We also tried other kenels such as RBF and polynomial whose implementations were based on LIBSVM, which took significantly longer time to run and did not yield better accuracies. Since we were working with high dimensional datasets, it was very likely that the datasets were linearly separable, which could potentially explain why using kernel functions did not improve SVM's performance. We also compared different regularization strength for linear SVM, and the model overfits when $C > 0.3$.

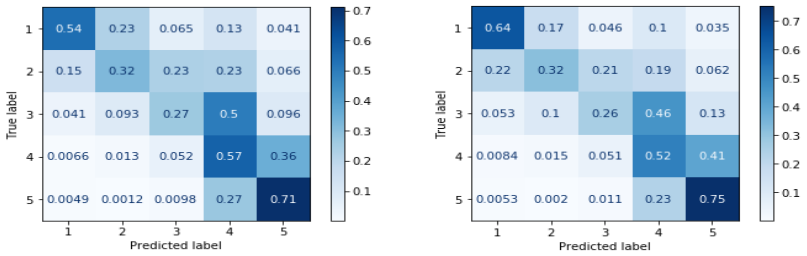

Figure 5: Confusion matrix for logistic regression (left) and linear SVM (right)

With the tuned hyperparameters, we compared the linear SVM with $C = 0.3$ to logistic regression with $C = 0.5$ using BoW 2 (M=500) features on the validation data. Figure 5 shows the confusion matrices of our final logistic regression and linear SVM models. Each row of a matrix sum to 1. The row and column labels correspond to classes, with class 1 being the most negative and class 5 being the most positive reviews. The numbers on the diagonal are rates of correctly predicted examples, and off-diagonal numbers are the rates of mis-classified examples. From the matrices, we can see that linear SVM model reached higher accuracy in classes 1 and 5, while logistic regression did slightly better in relatively neutral classes. For class 1, linear SVM greatly outperformed logistic regression as the LR model mislabeled 13% of class 1 samples as class 4. Both classifiers did not perform well in predicting examples from class 3. Table 3 presents the words associated with the largest positive weights for each class in the logistic regression model. The words for the two extremes, class 1 and 5, show very strong sentiment, and the words for class 4 are mostly positive, so the error of mislabelling class 1 as 4 might have been caused by the inherent bias in the data since it contains more positive reviews, which also explains why the models performed better for class 4 than class 2 even though both are neutral classes. Furthermore, the model struggled with classes 2 and 3 as evidenced by ambiguity reflected from the words, consistent with the findings from confusion matrices. Such

results are understandable because neutral cases can be hard to classify even for a human, not to mention that the dataset contains much more positive reviews.

Due to its fast training and competitive performance, we decided that linear SVM implemented with LIBLINEAR was the best one to use in practice among the models that we explored. Using the linear SVM with $C = 0.3$, we predicted the labels of testing dataset and got test accuracy 0.5502 with BoW 2 (M=1000) and 0.5354 with BoW 2 (M=500), which was a significant improvement compared to the initial models fitted with attributes given in the original data that yielded accuracies lower than 40%.

Class 1	Class 2	Class 3	Class 4	Class 5
disgust	maybe	alright	always	fantastic
never	lack	fine	quick	excel
waste	better	though	enjoy	love
bad	okay	nothing	excel	awesome
poor	meh	however	perfect	great
awful	disappoint	average	tasty	favorite
rude	bland	decent	nice	perfect
terrible	ok	pretty	definite	delicious
horrible	overpriced	okay	delicious	best
worst	mediocre	ok	great	amazing

Table 3: Words with largest positive weights in logistic regression model

6 Discussion and Analysis

One pitfall we fell into at the beginning of the project was that we did not leave a hold-out dataset to evaluate the performance of the selected model on unseen data at the end. Thanks to Roger, our project mentor, we were able to correct this mistake at an early stage. This experience has definitely reinforced the importance of differentiating between training, validation, and test datasets in our minds, and we will not make such mistakes in the future.

We tried multiple bag of words and embedding schemes when trying to find appropriate features for training. However, while the performance gets better by a small amount, the training time increases significantly. Especially for logistic regression, when we used BoW2 scheme with top 1000 features, the training time was almost 3 times of that using top 500 features, while the validation accuracy did not increase much.

Another drawback of the experiments is that we were not able to perform K-fold cross validation for all the models and features due to high computational costs. We performed 10-fold cross validation for the basic models with BoW 1 features since the default setting of TfidfVectorizer runs very fast. After building our own preprocessor and tokenizer in an effort to enhance feature extraction, the increased computational costs make it time-consuming to run all the combinations of features and models using cross validation. In order to have comparable results, we needed to train on the training data and test on the validation data, but such an arbitrary split could actually be non-random and cause the models to overfit and unable to generalize to unseen data, especially given the highly skewed distribution of stars in the original training dataset.

In terms of features, we were not able to generate a good set of features that significantly improves model performance due to limited experience with NLP prior to this project. A closer look into the current set of features suggests that correcting misspelling and reducing synonyms into a single word (such as "okay" and "ok" in table 3) has the potential to deliver better results. Specifying better word embeddings such as combining GloVe and FastText also holds great promise for better features.

With regard to models, we did not have the chance to take a deep dive into the LSTM model due to time constraint, but we believe that experimenting with neural networks will lead to improvements that are worth the complexity. For example, combining LSTM and GRU layers, stacking multiple LSTM layers, and replacing standard layers with bidirectional layers are few of the possible directions for further exploration. New architectures from image processing might also work for NLP tasks.

References and Citations

[1] Ganu, G., Kakodkar, Y., & Marian, A. (2013). Improving the quality of predictions using textual information in online user reviews. *Information Systems*, 38(1), 1–15. https://doi.org/10.1016/j.is.2012.03.001

[2] S, H., Ramathmika, R. (2019). Sentiment Analysis of Yelp Reviews by Machine Learning, *International Conference on Intelligent Computing and Control Systems (ICCS), Madurai, India, 2019*, 700-704. https://doi.org/10.1109/ICCS45141.2019.9065812.

[3] Schmidt, M., Le Roux, N., & Bach, F. (2016). Minimizing finite sums with the stochastic average gradient. *Mathematical Programming*, 162(1–2), 83–112. https://doi.org/10.1007/s10107-016-1030-6

[4] Hsieh, C., Chang, K.-W., Lin, C.-J., Keerthi, S., & Sundararajan, S. (2008). A dual coordinate descent method for large-scale linear SVM. *International Conference on Machine Learning*, 408–415. https://doi.org/10.1145/1390156.1390208

[5] Kingma, D. P., & Ba, J. L. (2015). Adam: A Method for Stochastic Optimization. ICLR. https://arxiv.org/pdf/1412.6980.pdf

[6] Keras documentation: Adam. (n.d.). Keras. https://keras.io/api/optimizers/adam/