Self-Attentive Classification Network for Sentiment Analysis on Yelp Reviews

Jonathan Hurwitz
University of California, Los Angeles
hurwitz@cs.ucla.edu

ABSTRACT

Yelp is widely used for finding quality food and well respected store-front businesses. Yelp encourages users to write reviews by offering a "Yelp elite" program, where selected special users get discounts at restaurants. This paper studies the effectiveness of a self-attentive classification network on predicting the star rating of a Yelp review based solely on the text. Baselines were established using random forest classifier and single layer neural network models with average pooling on the word embeddings.

1 INTRODUCTION

Yelp is an American internet company that enables users to crowd-source review data for businesses. It's primarily used for restaurants, but other storefront-type businesses exist on the site as well. Yelp reviews are structured in the form of a text response with a star rating, as well as some optional photos. Readers typically use a business' star rating as a signal of the business' quality. In fact, it has been shown empirically by [5] that a one-star increase led to a 59% increase in revenue of restaurants.

Due to the sheer amount of readable text in the corpus of aggregated review data, it's unlikely that a typical reader would read all reviews in depth. The star rating is a much faster indicator of aggregated quality, but it is also a faster indicator at the review-level granularity. Readers often skim the review text but pay extra attention to the star rating that the reviewer has assigned. Yelp allows users to assign integer-valued star ratings between one and five. This complicates the free-market signaling process as there is inherent ambiguity between subsets. For example, 1 star and 2 star ratings are both bad, but the difference is not immediately clear. Similarly, 4 and 5 star ratings are both good, but analysis of reviews that garner 4 and 5 star ratings shows dramatic similarity. Oftentimes, a 4 star rating has text that is so positive it could be a 5 star rating. This ambiguity results in potentially inaccurate signals to readers, as a reviewer's text may be overwhelmingly positive yet their star rating may be sub-par. The opposite is also possible: a reviewer's text may be negative but the star rating is positive. Here is an example of a 3-star review with super negative sentiment:

> "I never write reviews. But then I went to Armando's and felt the need to share. I had a breakfast burrito, and i'll admit it was good until an uneasy feeling hit my stomach a couple hours later. Food poisoning. One of the worst experiences of my life. Even my dog, who sniffs her own shit couldn't stand to be around me for 2 days. I will never go back to this place."

This could easily be a 1 star review. While the text is indicative of the issues with Armando's, quick readers will only be looking at star ratings, both individual and aggregate. Since this star rating is inaccurate, it is a misguiding market signal.

The approach discussed in this paper is based on the assumption that the majority of reviewers are rational, and that there is an implicit "mean" bijection between a reviewer's sentiment and a star rating. Concretely, the belief is that the majority of reviewers will think similarly. For example, an overwhelmingly positive review would be 5 star. A super negative review such as the one discussed above would be 1 star. The ratings in between are harder to accurately map. The aim is to learn this bijection by learning the underlying text features that correspond to more or less positive ratings. Labels consist of the star ratings, and the assumption of a mean bijection is used to justify the acceptance of outliers in the data. The model will take review input as text and output its predicted star rating. This mapping could be used to suggest a star rating to a user based on the sentiment and features of their text prior to posting a review. If the assumption were that yelp reviewers are irrational, then some method of transfer learning would be appropriate in order to preserve a text sentiment to integer output class bijection.

Additionally, there were several objectives defined after understanding the dataset breakdown. There were two datasets used, a severely imbalanced one from kaggle and a perfectly balanced one. These datasets are further explained in the following section. Additional project goals were defined as follows:

- Effectively deal with a class imbalance.
- Study the effects of pre-trained word embeddings vs. embeddings trained at each training step.
- Use part-of-speech (POS) tags as an additional feature.

2 DATASET

Two datasets were used to study different aspects of the problem.

2.1 RecSys2013 Dataset

The first dataset dataset was sourced from the yelp-recsys2013 kaggle competition [1] and consists of 229,898 reviews. The data consists of over 10,000 businesses, 8,000 check-in sites, and 40,000 users. All reviews were sourced from the Phoenix, Arizona metropolitan area.

The actual RecSys2013 competition has different problem setup than the problem discussed here, and it was only used as a data source. In the original problem discussed on Kaggle, users were also tasked with being able to predict a business' star rating, but the testing setup was different. Rather than learning purely from the review text, this problem provides features including business details, user details, and check-in information. The testing set withholds the actual review text and only provides user and business information. Therefore, the RecSys2013 testing set is unsuitable for

use in this project. Instead, the training set was separated into a train, test, and dev set. Each set is disjoint since unique businesses were partitioned into each set rather than having crossover.

Set	Number of Examples
Train	166101
Dev	29312
Test	34485

Table 1: Imbalanced dataset train, dev, and test breakdown.

The total dataset has a severe class imbalance. As such, this dataset was used to see how effective an imbalance mitigation technique such as loss weighting would be. Shown below are the class labels and the number of examples with that label:

Class	Number of Examples
1	17567
2	20615
3	35436
4	79708
5	76572

Table 2: Number of examples in each class.

The histogram shown below illustrates this severe class imbalance, with 69.97% of examples in classes 4 and 5.

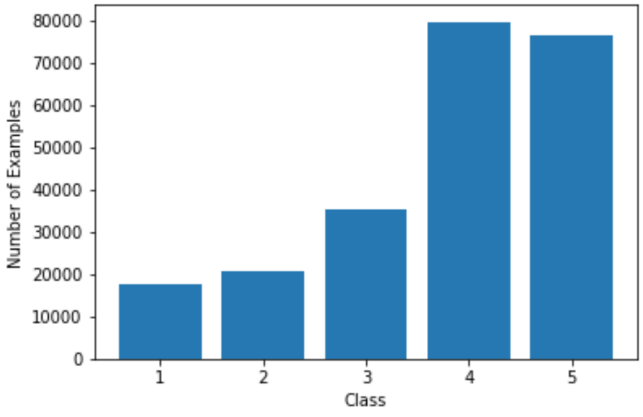

Figure 1: Class imbalance for the RecSys2013 dataset.

For the purposes of the RecSys2013 competition, this class imbalance was acceptable since the goal was not to predict a rating based on review text. Such a classifier trained on this dataset without any type of augmentation would result in memorization and outputting of the highest probability class. RecSys2013 involved other features, such as business characteristics and the actual user profiles in order to predict the ratings. Methods for dealing with such a class imbalanced are discussed in the models section.

2.2 Balanced Dataset

The other dataset was obtained from the authors of [2]. This dataset has 700,000 total reviews, and each class is perfectly balanced. Table 3 shows the breakdown of train, test, and dev sets.

Set	Number of Examples
Train	600000
Dev	50000
Test	50000

Table 3: Balanced dataset train, dev, and test breakdown.

3 BACKGROUND

In language, words are the building blocks upon which larger structures, such as phrases and sentences, are created. Words have inherent meanings based on their dictionary definitions, but their part of speech provides additional information into how the word can be used to assemble more complicated structures. Humans are very skilled at reading a sentence and determining the sentiment. The steps that occur behind the scenes include: per-word dictionary lookup for definition, linking from sequence (and subsequence) to other sequences (and subsequences) in order to form connections between subjects, objects, and actions, and finally special attention is paid to certain words such as adjectives, which help denote how someone is feeling about something. The first step to building a model that can emulate human sentiment analysis is to map a vocabulary of words to a continuous vector space that can be used as input to a model. This is referred to as word embeddings.

3.1 Word Embeddings

Common word embedding methods include word2vec and GloVe [7]. In the following discussions, GloVe pre-trained embeddings were used. Matrix factorization techniques such as latent semantic analysis (LSA) can capture global text statistics effectively but are deficient when it comes to actually capturing word to word meaning. GloVe combines matrix factorization with context. The GloVe method can be summarized as follows:

(1) Create a word co-occurrence matrix X. Some element X_{ij} represents a count of how often word i appears in the context of word j. Context is determined by a window size before current term and after current term. In some instances, these may be two separate hyperparameters. Words that are closer are given higher weight and words that are farther are given lower weight with a penalty function:

$$penalty = \frac{1}{distance} \tag{1}$$

(2) Using ratios of co-occurrence probabilities rather than actual probabilities, create a model:

$$F(w_i, w_j, \tilde{w_k}) = \frac{P_{ik}}{P_{jk}} \tag{2}$$

In this equation, $w \in R^d$ consist of word vectors and $\tilde{w} \in R^d$ is a context word vector. The next step is to formalize the definition of function F. Given a context k, we need to encode

information regarding the probability ratio. Using linearity of the vector space we restrict the function F to a difference between the two words i and j that we care about:

$$F(w_i - w_j, \tilde{w_k}) = \frac{P_{ik}}{P_{jk}} \tag{3}$$

Arguments on the left are vectors but the right hand side is a scalar. We take the dot product of the vector difference and the context, and enforce that F be a homomorphism as follows, in order to enforce labeling invariance:

$$F((w_i - w_j)^T \tilde{w_k}) = \frac{F(w_i^T \tilde{w_k})}{F(w_j^T \tilde{w_k})} \tag{4}$$

Numerator and denominator follow the same derivation, but the numerator on the right hand equates to P_{ik}, which is the co-occurrence of word i, context k, divided by count of word i. Concretely:

$$F(w_i^T \tilde{w_k}) = P_{ik} = \frac{X_{ik}}{X_i} \tag{5}$$

Solving for the above equation by applying log to both sides:

$$w_i^T \tilde{w_k} = log(P_{ik}) = log(X_{ik}) - log(X_i) \tag{6}$$

The original paper moves the term $log(X_i)$ into a bias term b_i and rrearranges as follows:

$$w_i^T \tilde{w_k} + b_i + \tilde{b_k} = log(X_{ik}) \tag{7}$$

This equation has several problems. The logarithm diverges when X_{ik} is zero. The simplest solution is to shift the function additively, but co-occurrences are still weighted equally. We want occurrences that are rare to not be underweighted and occurrences that are common not to be overweighted. Thus, a weighting function $f(X_{ij})$ is used as a coefficient in consideration of word i and word j's relation.

(3) The cost function can be summarized as a least-squares problem given the weighting function:

$$J = \sum_{i,j=1}^{V} f(X_{ij})(w_i^T \tilde{w}_j + b_i + \tilde{b_j} - log(X_{ij}))^2 \tag{8}$$

The original authors chose the following weighting function to satisfy the expected requirements regarding divergence and appropriate weighting for rarity:

$$f(x) = \begin{cases} (\frac{x}{x_{max}})^\alpha, & \text{if } x < x_{max}. \\ 1, & \text{otherwise.} \end{cases} \tag{9}$$

3.2 LSTM

Since text reviews consist of sequences of words, and sequences of larger hierarchical structures, it is appropriate to use models that can effectively learn sequence-to-sequence and chain dependencies. LSTMs are a class of RNN that solves some of the classic problems, namely the vanishing gradient problem in long-chain RNNs.

The LSTM can allow information to pass through easily, and can also add cell state regulated by gates. Each gate consists of a sigmoid activation and a pointwise multiplication. Smaller activations result in less information being let through, and vice versa. The overall LSTM architecture can be described as follows (images courtesy of Christopher Olah [6]):

(1) Determine how much information to keep or forget from the cell state via a forget gate. This is a simgoid layer that considers current input as well as state of the previous time increment. Since the output is multiplied into the current cell state, output of 1 means that this information is kept, and zero means it is forgotten.

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{10}$$

Shown below is a figure illustrating the inputs for the forget gate:

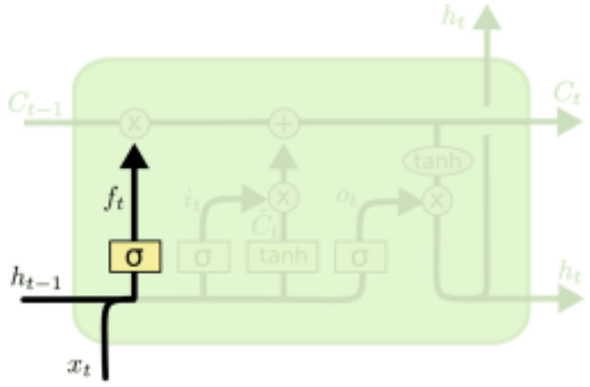

Figure 2: LSTM cell forget gate.

(2) Determine what information gets stored in teh cell states. The value i_t is an input vector to the multiplication unit which determines which values are updated. It has the ability to zero-out entries. The $\tilde{C}_t$ vector consists of the entire set of candidate values to be added to the state. The multiplication unit combines these two.

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{11}$$

$$\tilde{C}_t = tanh(W_c \cdot [h_{t-1}, x_t] + b_C) \tag{12}$$

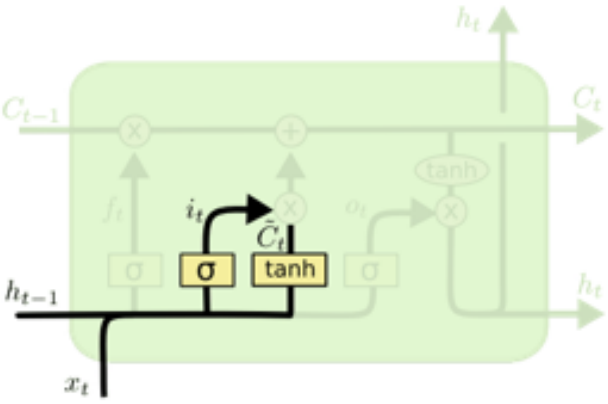

Figure 3: LSTM cell input and candidate values.

(3) Update the cell state. This is done by multiplication and addition blocks. First, the forget gate is multiplied by the cell state of the previous iteration in order keep and discard elements. The input selector vector is also multiplied by the candidate value vector to perform a state update:

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{13}$$

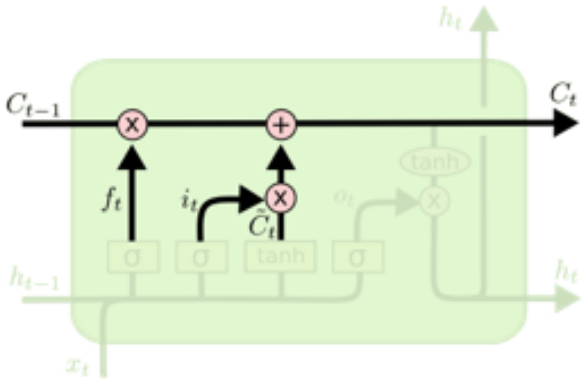

Figure 4: LSTM update cell state.

(4) The final step involves passing current input and history multiplied by weights through another sigmoid activation to create a filtering vector. This vector is then multiplied by the tanh activation output of the current cell state in order to pick and choose which values contribute to the output, and to what degree:

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \tag{14}$$

$$h_t = o_t * tanh(C_t) \tag{15}$$

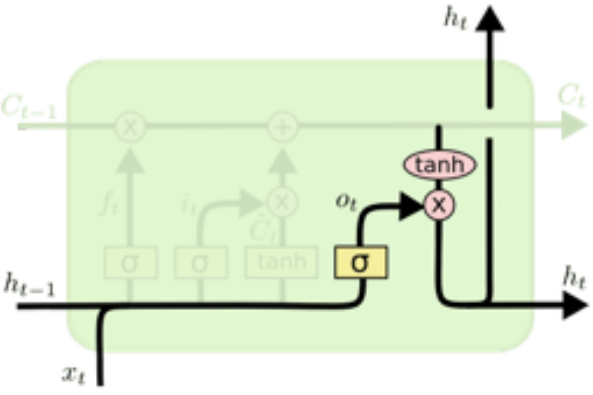

Figure 5: LSTM output.

4 MODELS

Random forest and a single-layer neural network were used to establish baselines. Random forest was chosen since it is a simple ensemble method, and the basic neural net was also used to establish a baseline slightly more relevant to the more complicated, sequence based model. Since this is a 5-class classification problem, the true random classifier is expected to have an accuracy of 20%.

In all models, pre-trained 300d GloVe embeddings were used from Stanford NLP. This consists of 6B tokens and 400k size vocab. For both the random forest and feed-forward neural network, mean pooling was used to create a 300d representation of a sentence. If a word was not in the embeddings dictionary, a vector of zeros was used in its place. Since these models were intended to provide baseline estimates, more complicated word vector aggregation methods were not used.

4.1 Random Forest Classifier

The default random forest classifier from the sklearn package was used. This model has 10 estimators.

4.2 Single-layer Feed-forward Neural Network

The default MLPClassifier from the sklearn package was used. This is a single-layer network with 100 hidden nodes, ReLU activation function, Adam solver, and Nesterov momentum.

4.3 Self-Attentive Classifier

This classifier is based off of a bi-attentive classification network designed by Wasi Ahmad at UCLA. The LSTM encoding mechanism is adapted from Facebook research's InferSent model [3]. The affinity matrix represents the similarity between sentences based on contextualized word representations. It is calculated by doing the matrix multiplication of the LSTM encoded values of sentence 1 with the transpose of the encoded values of sentence 2. The cosine similarity is calculated during the Euclidean dot product, ignoring the scaling factor in the denominator:

$$similarity = cos(\theta) = \frac{v_1 \cdot v_2}{||v_1|| * ||v_2||} \tag{16}$$

When sentences 1 and 2 are different, then the LSTM state vector representation will be different. However, when these are the same then the state should be nearly the same (ignoring weight initialization differences). Attentive (conditioned) representations are produced by first taking a softmax of the affinity matrix and then performing a matrix multiply between the output and the encodings. These are further passed to the biattentive encoding layers, pooled, and then passed through a maxout network. The overall architecture is shown in the figure below.

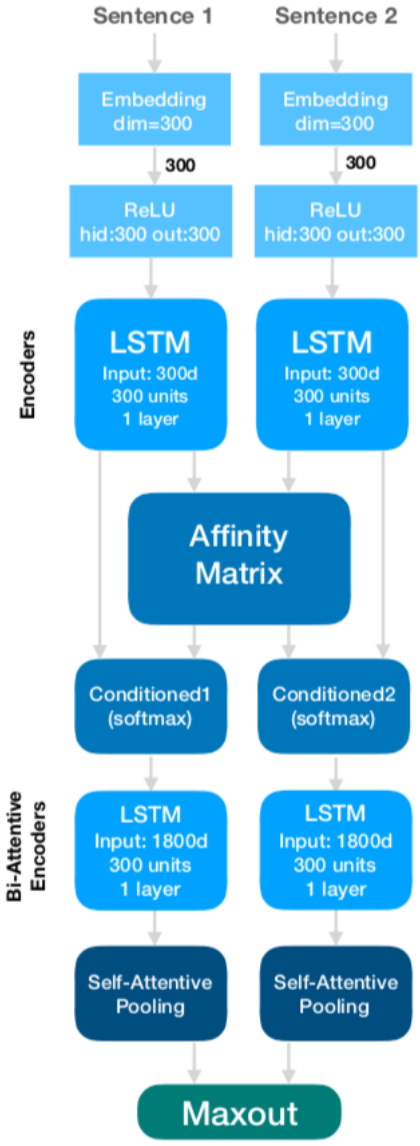

Figure 6: Architecture for bi-attentive classification network.

The basic architecture was augmented by using part of speech (POS) tags as features. Given enough data, the LSTM should be able to learn sequence-to-sequence relationships and implicitly learn parts of speech and their importance. However, the motivation for using POS tags as an additional feature came from observing the way humans learn. For example, adjectives carry a lot of weight with regards to sentiment, but they cannot be taken alone. They must be considered in context as parts of a greater structure. Knowledge of their presence and their POS can indicate a more important structure. The expectation was that by adding explicitly POS tags, the model would be able to more quickly learn which structures are important and which are not.

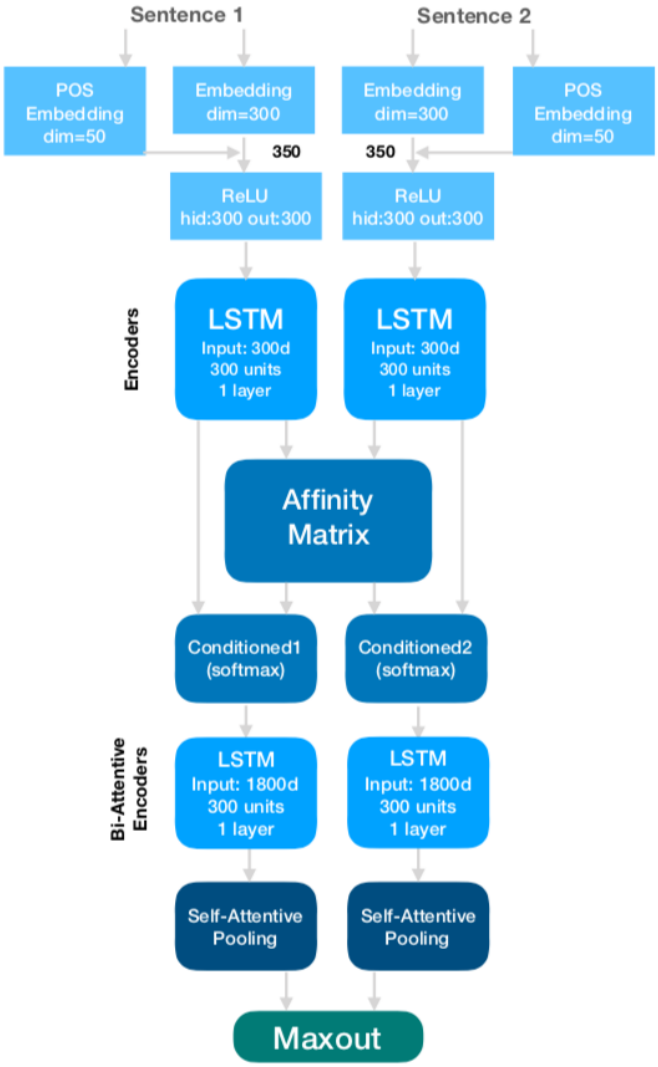

Figure 7: Architecture for bi-attentive classification network with part of speech as a feature.

The above below shows how this modification was implemented. NLTK tagger was used to tag tokens into one of 37 possible tags, with tags 1-36 being the standard Penn Treebank tags and tag 37 being <unk>. The POS embedding weights were randomly initialized using Xavier initialization [4] and then updated via backpropagation during training time. The POS embedding vector is then concatenated to the word embedding vector before being passed through the ReLU layer. The downstream parts of the network are the exact same in both instances.

4.3.1 ***Loss Weighting.*** When training data exhibits severe class imbalances, such as in the case of the RecSys2013 dataset, a model will "learn" to output the most probable class without actually building an expressive and generalizable mapping from input feature space to output classes. One technique to solve this is loss weighting, wherein a penalty is multiplied backward in the network

to adjust weights. Classes with more members are penalized by multiplying their weights with a smaller number to keep overall weight magnitude relatively lower. Classes with fewer members are penalized significantly less or not at all. The downside of this method is that training time increases as the weights are rescaled at each iteration. There are two simple ways of doing loss weighting:

(1) **Inverse Class Size** In this method, the weighting vector is created by iterating through each class, finding the total number of examples in the class, and then inverting this value. For example, if the class:size distribution were as follows: 1:10, 2:10, 3:5, 4:50, 5:25, then the weighting vector would be:

$$weights = [\frac{1}{10}, \frac{1}{10}, \frac{1}{5}, \frac{1}{50}, \frac{1}{25}] \tag{17}$$

(2) **Inverse Probability** In this method, the weighting vector is created by iterating through each class, finding the total number of examples in the class, dividing each instance by the total number of examples in the entire set, and then taking the inverse probability by subtracting this value from 1. Taking the same distribution from before, the raw class probabilities are:

$$raw\_probabilities = [\frac{10}{100}, \frac{10}{100}, \frac{5}{100}, \frac{50}{100}, \frac{25}{100}] \tag{18}$$

After subtracting each term from 1 to take the inverse, the weighting vector is:

$$weights = [\frac{90}{100}, \frac{90}{100}, \frac{95}{100}, \frac{50}{100}, \frac{75}{100}] \tag{19}$$

In both cases, the smallest class (class 3) is penalized the least amount. The largest class (class 4) is penalized the most. However, the penalization is much larger in method 1. Method 2 was used instead in order to balance the weighting with training speed.

5 APPROACH

The approach was to train two main sets of models in addition to the baseline models. In the following descriptions, the balanced sets come from the Yelp full dataset and the imbalanced sets come from the RecSys2013 dataset. Performance metrics include raw accuracy, F1-score, mean-squared error, and a confusion matrix. Mean-squared error gives us an indicator of how close the model predictions were to ground truth, and is applicable here due to the inherent ambiguity between the subsets within the low and high rating categories.

Evaluation on the balanced dataset will provide an indicator of overall classifier performance. Evaluation on the imbalanced dataset and analysis of the corresponding confusion matrices will test how effectively the model learned the implicit mapping between sentiment and star rating. A model that has effectively learned this mapping should not only have a saturated diagonal in the confusion matrix, but also relatively balanced F1-scores for each class.

5.1 Baseline Models

The random forest and feed-forward neural network models were partitioned into two sets: trained & tested on balanced data as well as trained & tested on imbalanced data.

5.2 Set 1: Models Trained on Balanced Data

The first set of models were trained on the balanced dataset and tested on both a balanced and imbalanced test set. Since all models in the first set were trained on balanced data, none of these models were run with the loss weighting mechanism. There were six configurations of this first set, each with a single variation on the control model. The control model was trained with no POS features, pre-trained embeddings without weight updates, and no limit on vocabulary size. These configurations are summarized in Table 4. Model 2's configuration included training word embeddings on the full vocab size, but this model was not trained because of the memory requirements. Similarly, model 5 involved training both POS embeddings and word embeddings, but was not run due to memory requirements.

5.3 Set 2: Models Trained on Imbalanced Data

The second set of models were trained on the imbalanced data and tested on both a balanced and imbalanced test set. There were nine configurations of this second set. The control model was run with no loss weighting, no POS features, pre-trained embeddings without weight updates, and no limit on vocabulary size.

5.4 Hypotheses

There were several hypotheses, summarized below:

(1) Limiting the vocab size to the top 80,000-100,000 or so frequent words would not drastically reduce classification performance, while at the same time reducing training time.
(2) Part of speech features will result in higher classification performance and potentially faster training time.
(3) Training word embeddings on the specific training corpus will increase classifier performance.
(4) Loss weighting will improve classifier performance and result in more diagonal entries in the confusion matrices.

6 RESULTS

The full set of results across all models and all metrics can be found in the console output dumps on the project GitHub. Core metrics for the self-attentive classifier, such as accuracy, F1, and MSE are shown in tables 8 and 9. Recall that models 1-8 were trained on balanced data and models 9-17 were trained on imbalanced data.

6.1 Baseline Models

Table 6 shows the results for the baseline models trained on the imbalanced dataset and tested on a held out portion of data from the imbalanced dataset. Since there are 79708 examples in class 4 and a total of 229898 examples, and since the testing set has nearly the same class proportion as the training set, a model could simply output class 4 and achieve 34.67% accuracy. F1 score would be low. Since most examples are grouped into classes 4 and 5, a model outputting primarily 4 or 5 would not have terribly high MSE. In fact, the models trained and tested on the imbalanced dataset show a relatively lower MSE than those trained and tested on the balanced dataset for this very reason. Thus, MSE is not the most effective metric when studying results of models trained & tested

Model	Training Set	Testing Set 1	Testing Set 2	Loss Weighting	POS	Custom Embeddings	Max Words
1	Balanced	Balanced	Imbalanced	No	No	No	All
3	Balanced	Balanced	Imbalanced	No	Yes	No	All
5	Balanced	Balanced	Imbalanced	No	No	No	80000
6	Balanced	Balanced	Imbalanced	No	No	Yes	80000
7	Balanced	Balanced	Imbalanced	No	Yes	No	80000
8	Balanced	Balanced	Imbalanced	No	Yes	Yes	80000

Table 4: Model configurations for set 1.

Model	Training Set	Testing Set 1	Testing Set 2	Loss Weighting	POS	Custom Embeddings	Max Words
9	Imbalanced	Balanced	Imbalanced	No	No	No	All
10	Imbalanced	Balanced	Imbalanced	Yes	No	No	All
11	Imbalanced	Balanced	Imbalanced	Yes	No	Yes	All
12	Imbalanced	Balanced	Imbalanced	Yes	Yes	No	All
13	Imbalanced	Balanced	Imbalanced	Yes	Yes	Yes	All
14	Imbalanced	Balanced	Imbalanced	Yes	No	No	80000
15	Imbalanced	Balanced	Imbalanced	Yes	No	Yes	80000
16	Imbalanced	Balanced	Imbalanced	Yes	Yes	No	80000
17	Imbalanced	Balanced	Imbalanced	Yes	Yes	Yes	80000

Table 5: Model configurations for set 2.

on the imbalanced dataset. The neural network model achieved better accuracy, F1 score, and MSE.

Model	Accuracy	F1 Score	MSE
Random Forest	37.94%	0.2088	1.5352
FF Neural Network	39.79%	0.3241	1.4241

Table 6: Results for baseline models trained & tested on the imbalanced dataset.

Table 7 shows the results for the baseline models trained on the balanced dataset and tested on a held out portion of data from the balanced dataset. In this case, it's much more difficult to achieve a low MSE because of the perfectly balanced data. A low MSE would indicate that the model is outputting class predictions close to their true value. Since the data is balanced, MSE is an effective metric here. One again, the neural network performed better than the random forest in all metrics. Both models show lower accuracy than in 6 because the data is balanced, thus the model cannot "learn" to output a high probability class. The F1 scores of both models were higher, indicating that training on the balanced dataset allowed the models to learn the mapping from text features to rating more effectively.

Model	Accuracy	F1 Score	MSE
Random Forest	30.94%	0.3014	3.1237
FF Neural Network	35.17%	0.3402	2.6607

Table 7: Results for baseline models trained & tested on the balanced dataset.

6.2 Set 1

The first point of comparison would be between a model trained with pre-trained embeddings vs. one with embeddings trained at each iteration. This comparison could not be done with the full vocab size. Given the models that were trained with the full vocab size, the question to be answered is the effect of POS tags as a feature on model performance. Interestingly enough, model 3 (with POS features) performed worse in accuracy, overall F1-score, and MSE than model 1 for both balanced and imbalanced evaluation sets. However, model 3 boasted higher F1-scores for minority classes in both balanced and imbalanced evaluations.

The next expectation was that limiting the vocab size to the 80,000 most frequent words would not drastically reduce classifier performance while significantly improving training time. This was confirmed and can be shown in the comparison of performance between models 1 and 5. In fact, model 5 narrowly outperformed model 1 in terms of mean-squared error for evaluation on the balanced dataset, scoring 0.59912 vs. 0.60518. Model 5's minority class F1-scores were also marginally better than model 1's scores. Performance of model 5 was slightly worse than model 1 on the imbalanced dataset for all categories, including minority class F1-score.

Another expectation was that training word embeddings would increase classifier performance. Model 6's word embeddings were trained whereas model 5 used the pre-trained GloVe embeddings. Model 6 outperformed model 5 in every single evaluation metric, for both balanced and imbalanced evaluation sets.

Both model 7 and model 8 had a vocab size of 80,000, but model 7 used POS features and pre-trained embeddings whereas model 8 uses both POS features and trains its embeddings. Model 7's performance is worse across the board than model 6, but model 8 outperforms both of them.

Of set 1, model 8 performed the best with model 6 performing second best.

Model	Accuracy	F1 Score	MSE
1	64.52%	0.6396%	0.6051
3	63.67%	0.6347%	0.6155
5	64.48%	0.6393%	0.5991
6	65.68%	0.6586%	0.4868
7	63.75%	0.6343%	0.6239
8	66.01%	0.6589%	0.5012
9	60.51%	0.6011%	0.7021
10	60.46%	0.6022%	0.7139
11	59.68%	0.5874%	0.6548
12	59.05%	0.5845%	0.7440
13	59.93%	0.5953%	0.6812
14	59.23%	0.5914%	0.7391
15	59.23%	0.5804%	0.6913
16	60.35%	0.6044%	0.6685
17	59.55%	0.5855%	0.7030

Table 8: Results for set 1: Models tested on balanced data.

6.3 Set 2

The first step was to determine if loss weighting via inverse probability is effective enough to mitigate a severe class imbalance problem. Model 9 does not use loss weighting while model 10 does. The loss weighting marginally improves the F1-scores of minority classes 2 and 3 (corresponding to 2 star and 3 star ratings), but every other metric decreases by a small amount. Despite only this small improvement, loss weighting was used for the remainder of the experiments involving training on the imbalanced set.

Models 10 and 12 differ only in the introduction of POS features. Both use the full vocab of the imbalanced dataset. The results here further support the results from set 1: adding POS features to a model training on a full-sized vocab without training word embeddings reduces performance across all categories. However, in this situation, minority class F1-scores did not improve.

Models 9 and 14 differ in maximum vocab size. Models 14-17 are limited to the 80,000 most frequent words. The results here also support hypothesis 1 and the results obtained from set 1: limiting vocab size marginally reduces performance while decreasing training time. Model 9 performed worse in every category but each difference was less than 1%.

When trained on the imbalanced dataset with vocab limit, adding in POS features actually improved performance in nearly every way, with the exception of a small MSE increase for the imbalanced evaluation set. Unlike set 1, the best performance was not exhibited by a combination of POS features and trained embeddings on the limited vocab size, but rather by just adding POS features and training on the limited vocab size. The best performing model from this set was model 16.

Model	Accuracy	F1 Score	MSE
1	61.41%	0.6084%	0.7177
3	59.60%	0.5960%	0.8505
5	60.53%	0.6053%	0.7823
6	61.50%	0.6150%	0.6078
7	60.02%	0.6002%	0.7792
8	62.27%	0.6227%	0.6461
9	59.41%	0.5941%	0.8562
10	58.93%	0.5893%	0.8994
11	55.44%	0.5544%	0.9317
12	57.82%	0.5782%	0.8735
13	59.28%	0.5928%	0.7899
14	59.70%	0.5970%	0.7902
15	53.35%	0.5335%	1.0368
16	60.25%	0.6025%	0.8011
17	57.16%	0.5716%	0.8531

Table 9: Results for set 2: Models tested on imbalanced data.

7 DISCUSSION

Due to ambiguity and some blurred lines between the classes, mean-squared error in conjunction with F1-score proved to be a good evaluation metric. The self-attentive classifier labeled the negative Armando's review (discussed in the introduction) as 1-star, whereas the ground truth was 3 star. This, combined with the performance increases across all metrics when compared to the baselines suggests that the model was able to learn a basic bijection between the text features and the star rating. However, it's worth noting that ambiguity does not exist only at the extremes. For example, here is a 3 star review that the self-attentive classifier labeled as a 4-star review:

> "I went to Z Tejas with a bunch of people (about 25) last night. We had a great experience from start to finish. At the beginning my lady and I were early so the waiter got us our drinks and continued to make sure we were good to go. We wanted to wait for our friends so food wasn't necessary at this time, but the drinks were flowing and the waiter (Jeff) was very cool, friendly, and down to earth. All of our friends showed up and we let the good times roll. Chips and salsa, drinks, water, and the delicious cornbread, they all came very fast and it didn't take any begging on our part. I got a chicken dish which was really good, I stuffed myself silly. My lady got a side salad which was huge, she couldn't even finish it and she said it was really good. Oh! and the food came fast by the way (surprising when you have 25 people.The overall vibe of the place is really cool, very trendy and fancy but reasonable prices. Big screens everywhere, gotta love that since there were some great College FB games on last night. Great experience, thanks Jeff & Z Tejas."

This review has many positive words and phrases: cool, friendly, 'down to earth', delicious, 'very trendy', 'great experience'. This could pass as a higher rated review, and the classifier has chosen to assign a higher value. Without MSE, this example would be

considered a complete misclassification, and there would be no notion of "closeness".

7.1 Hypothesis 1

As expected, limiting the vocab size to the 80,000 most frequent words did not drastically reduce classifier performance. In some instances, performance was actually increased. Training time was reduced by several hours.

7.2 Hypothesis 2

While adding POS tags as a feature did not significantly improve performance across the metrics defined, it did result in an interesting performance boost in terms of training time. The following figure plots the training and validation accuracies across three epochs for models 5 and 7. Both models have vocab limited to 80,000 words. Model 7 uses POS features whereas model 5 does not. The plot below shows that model 7 (blue) trains significantly faster than model 5 (red). In these early epochs, overfitting is not as much of an issue and the validation accuracies track both training curves well.

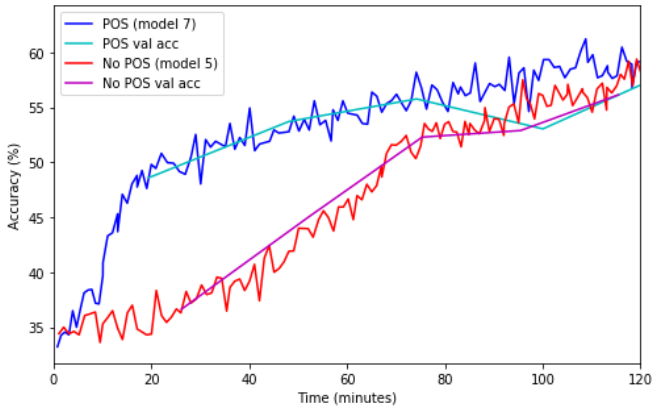

Figure 8: A comparison of training speed and validation accuracy for models 5 and 7.

These models both saturated in terms of validation accuracy. However, model 7 reached this saturation point nearly an hour before model 5. At the 120 minute mark, model 5's training and validation accuracy finally intersect those of model 7.

Across the board, the POS features did not dramatically improve classifier performance metrics but they did reduce the time it takes to reach a saturation point in training. Since this is a sentiment analysis task, the POS tagging may not be extremely useful for determining class since both positive and negative sentiment can be related to the same tag. One example might be the words "delicious" and "horrible". In terms of sentiment, these belong in different classes, but they share the same POS tag. During training, explicitly showing the model the POS tags and training this embedding layer may help the model learn where to look but it does not show the model the underlying meaning of the structure or word.

7.3 Hypothesis 3

In general, training word embeddings on the specific training corpus marginally increased classifier performance when coupled with a vocab limited to the most frequent words.

7.4 Hypothesis 4

Inverse probability loss weighting provided marginal improvement in the minority class F1-scores for the imbalanced data, namely the 1, 2, and 3 star review categories. Other metrics were not improved and some worsened, likely due to the impact penalization has in terms of how quickly the model can learn. A more significant penalization like inverting the class size may improve classifier performance on such an imbalanced dataset.

8 CONCLUSION

Measurable results were found for all four hypothesis. It was shown that limiting vocab size is an effective way to maintain model performance while drastically speeding up training time. While the initial hypothesis was that POS tags would allow for better classification performance and potentially faster training time, only the latter was true. Models trained with POS features trained faster but did not gain any expressiveness over models trained without POS tags. Both models' training and validation accuracies intersected after about two hours of training time, after which performance was relatively equal. It was found that training custom word embeddings improved performance, but not drastically. Inverse probability loss penalization was used to deal with the class imbalance of the recsys dataset, but this proved to be ineffective due to how severe the imbalance was. A more significant penalization should be attem

ACKNOWLEDGMENTS

I would like to thank Wasi Ahmad for his continuous patience, guidance, and regular assistance with issues during this project. I'd also like to thank Rizwan Parvez for providing high level direction, advice, and some early on assistance. I'm grateful to Professor Kai-Wei Chang for also providing direction and motivation for the project.

REFERENCES

[1] [n. d.]. RecSys2013: Yelp Business Rating Prediction | Kaggle. https://www.kaggle.com/c/yelp-recsys-2013/data
[2] Alexis Coneau, Holger Schwenk, Yann LeCun, and Loic Barrault. [n. d.]. Very Deep Convolutional Networks for Text Classification. https://arxiv.org/pdf/1606.01781.pdf
[3] Alexis Conneau, Douwe Kiela, Holger Schwenk, Loic Barrault, and Antoine Bordes. [n. d.]. Supervised Learning of Universal Sentence Representations from Natural Language Inference Data. https://arxiv.org/pdf/1705.02364.pdf
[4] Xavier Glorot and Yoshua Bengio. [n. d.]. Understanding the Difficulty of Training Deep Feedforward Neural Networks. http://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf
[5] Michael Luca. 2011. Reviews, Reputation, and Revenue: The Case of Yelp.com.
[6] Christopher Olah. 2015. Understanding LSTM Networks. http://colah.github.io/posts/2015-08-Understanding-LSTMs/
[7] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. [n. d.]. GloVe: Global Vectors for Word Representation. https://nlp.stanford.edu/pubs/glove.pdf

APPENDIX

A SOURCE CODE

https://github.com/jdhurwitz/pyfinder/blob/master/README.md