Yelp Review Prediction:

Comparing Traditional Machine Learning and Deep-Learning Techniques in Text Sentiment Analysis

James Keane and Jimmy Melican

Machine Learning (BUSN 41204) Winter 2022

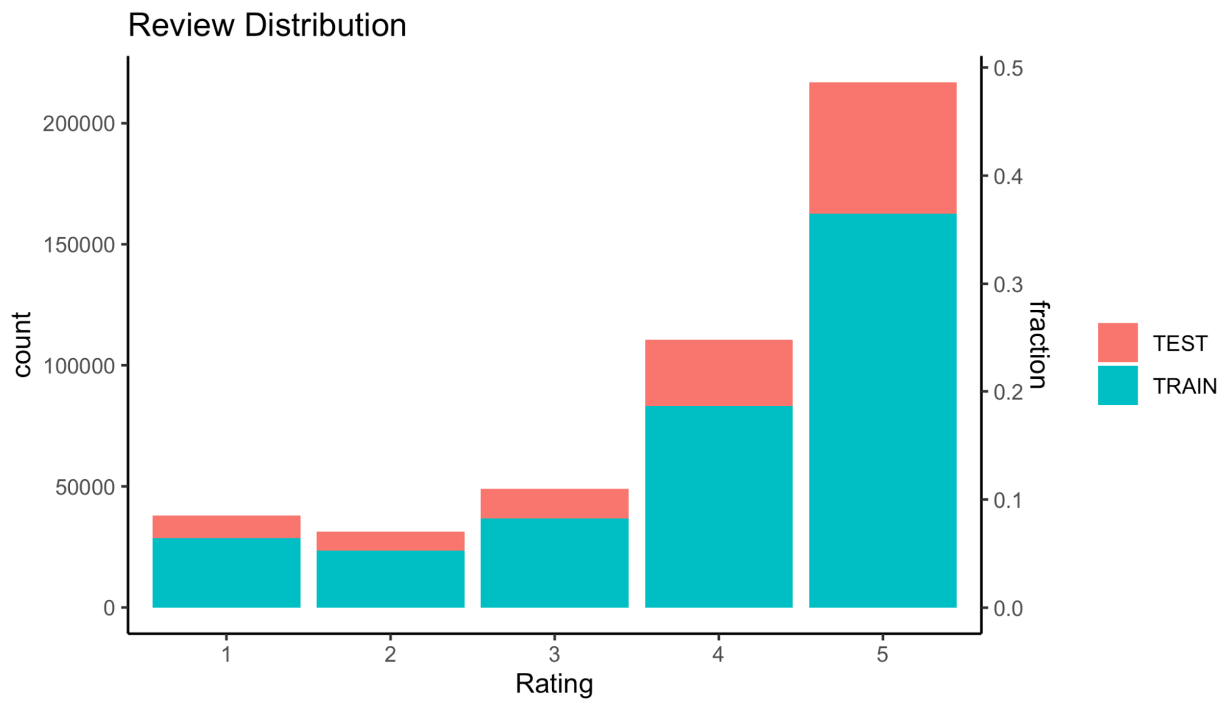
Introduction

Motivation

When you look up a restaurant, whether with Google Search or on Apple Maps, one of the first things you will see is its Yelp rating. The crowd-sourced metric can be an important factor in helping customers determine where to eat. As ratings are often accompanied by written reviews, it may be of interest to businesses, customers, and Yelp itself to be able to predict ratings from the text of the review alone. One potential use case could involve scraping text that mentions a particular restaurant from a website such as Twitter and using a trained model to generate a star-rating. These ratings could then supplement those from Yelp to get a larger sample size.
In this project, we aim to evaluate if a Bidirectional LSTM model is a viable text sentiment classifier compared to traditional Machine Learning techniques, and other state of the art Deep Learning models like Google's BERT.

Data

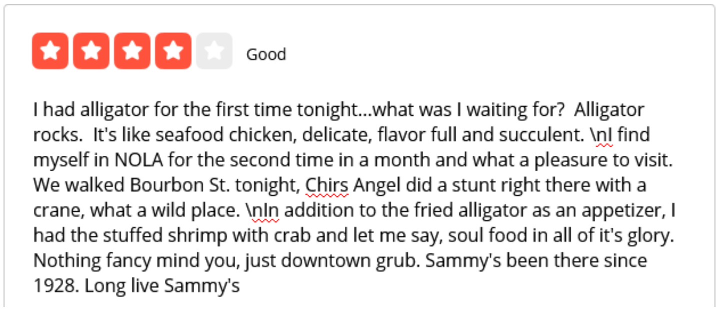
The data was part of the Yelp Open Dataset, a subset of nearly 7 million Yelp reviews covering over 150,000 businesses across 11 metropolitan areas.[1] The data are made available as .json files, and we downloaded the business and review files. We joined these datasets together and then filtered to consider only reviews of restaurants in New Orleans. Since the deep learning models we intended to fit have limitations on input size, we also filtered the data set to exclude reviews over 275 words, which only accounted for about 5% of the reviews within each rating category. This left us with 445,953 reviews, and we then partitioned the data into a 75/25 train/test split,



keeping the distribution of star ratings the same across each class. As can be seen in the graphic above, the data is unbalanced, with nearly half of the reviews being 5 stars.

Text Pre-processing

To coerce our review texts into a form usable for our modeling structures, we first processed the text by cleaning out non-alphanumeric characters, tokenizing, lemmatizing, removing stop words, and converting the end result into a numeric vector form. As an example, we will look at a random review from the training set in its original form.



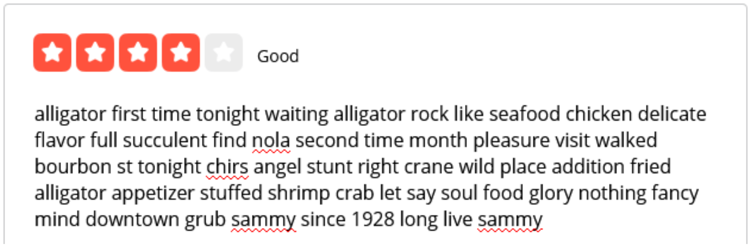
I had alligator for the first time tonight...what was I waiting for? Alligator rocks. It's like seafood chicken, delicate, flavor full and succulent. \nI find myself in NOLA for the second time in a month and what a pleasure to visit. We walked Bourbon St. tonight, Chirs Angel did a stunt right there with a crane, what a wild place. \n\nIn addition to the fried alligator as an appetizer, I had the stuffed shrimp with crab and let me say, soul food in all of it's glory. Nothing fancy mind you, just downtown grub. Sammy's been there since 1928. Long live Sammy's

For the first step, we remove all punctuation, new line characters (\n), and any non-alphanumeric characters. Additionally, we convert all characters to lowercase.

Next we remove the ending inflection on words, returning them to their *lemma* or base dictionary form. This reduces the size of our ultimate vocabularies, and groups common words together (eg *eats* and *eat*).

Not all words are particularly useful for gauging the sentiment of the review. These *stop words* – such as *I, have, up, by* – are necessary for sentence coherence but do not carry any context on their own. As they are incredibly frequent, they can add noise to traditional models dealing with word frequency, and are often removed. For this project, we used a list of English language stop words from the *NLTK* language processing python package.

After cleaning, lemmatizing, and removing stop words, our example review takes the following form. In place of sentence structure, we have a list of tokens.



alligator first time tonight waiting alligator rock like seafood chicken delicate flavor full succulent find nola second time month pleasure visit walked bourbon st tonight chirs angel stunt right crane wild place addition fried alligator appetizer stuffed shrimp crab let say soul food glory nothing fancy mind downtown grub sammy since 1928 long live sammy

Text Vectorization

The final step of text preprocessing involves converting the list of tokens into a numeric representation, capable of being read by a classifier. We do this text vectorization in two ways.

TF - IDF

Our simpler vectorization approach involves finding the frequency of the words in a given review. For our measure of frequency, we measure the product the *Term Frequency (TF)* by the *Inverse document Frequency* for a given term.

$$TF_{i,j} = \frac{Count\ of\ Term_i}{Count\ of\ all\ Terms\ in\ Review_j}$$

$$IDF_i = log\frac{Count\ of\ Reviews}{Count\ of\ all\ Reviews\ with\ Term_i}$$

This product – denoted as *TF-IDF* – better prioritizes terms that are frequent in a given review, but not common across all reviews, as opposed to terms that are simply frequent..

We computed the TF-IDF for all unigrams, bigrams, and trigrams that occur in at least 0.5% of the training reviews. Our end result is a sparse matrix with 334,466 rows and 1,390 columns respectively representing the training reviews and the terms in our vocabulary.

Review Embedding

A higher dimensional representation of vectorizing the review texts is done using *word embedding*. This is a method wherein each word is mapped to a full vector of real numbers, rather than just a single frequency value. For our project, we utilize an existing set of word embedding vectors from Stanford's Global Vectors, or *GloVe* model. In particular, we utilize the pre-trained, 200-dimensional word vectors created from some 2 billion tweets.

To create our final product, we first defined a new vocabulary of the 12,000 most frequent terms across all documents. This vocabulary was notably much larger than the one used for TF-IDF computation, as it was used by deep learning methods with the power to handle much greater feature dimensions. Additionally, we included stop words in this vocabulary, as these methods are also able to assess words in the context of their neighboring words, making stop words potentially useful contextual supplements.

Every token outside of our vocabulary was replaced with the string *<OOV>*, which has a vector representation of all 0s, making it ineffectual on predictions. Every in-vocabulary token was then replaced with its respective 200-dimensional vector from the GloVe twitter model.

Our final review embedding result was a matrix with 334,466 x 200 x 262 dimensions. The first dimension being the count of training reviews, the second being the dimensionality of the embedding vectors, and the third being the maximum token length of the training reviews. To align all review lengths, any review shorter than 262 tokens was pre-padded with <OOV> tokens.

Models

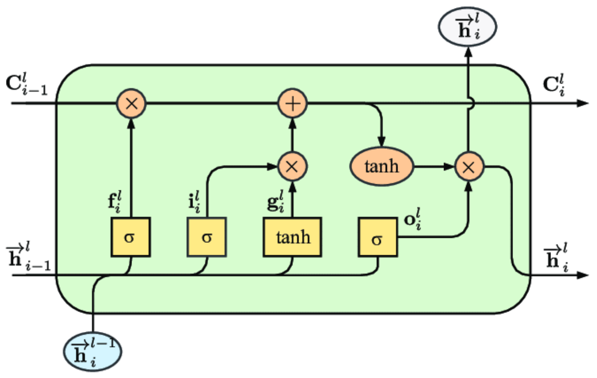
Baseline Models

To compare with our deep-learning models, we fit two traditional machine learning methods – a Logistic Regression and a Random Forest classifier. Both utilize the sparse TF-IDF matrix representations of the review texts as they are unable to input the higher dimensionality of the embedded reviews.

LSTM

Our main model of interest is a Long Short-Term Memory or LSTM neural network. This is an extension on the more standard Recurrent Neural Network (RNN) architecture, which maintains a hidden state that carries forward dependencies across a data sequence (token contexts across review text in this project). The LSTM was originally created to account for the *vanishing gradient* issue of RNNs, where contexts of tokens distant from the current token in the sequence have their weights set to zero, rendering them "forgotten."

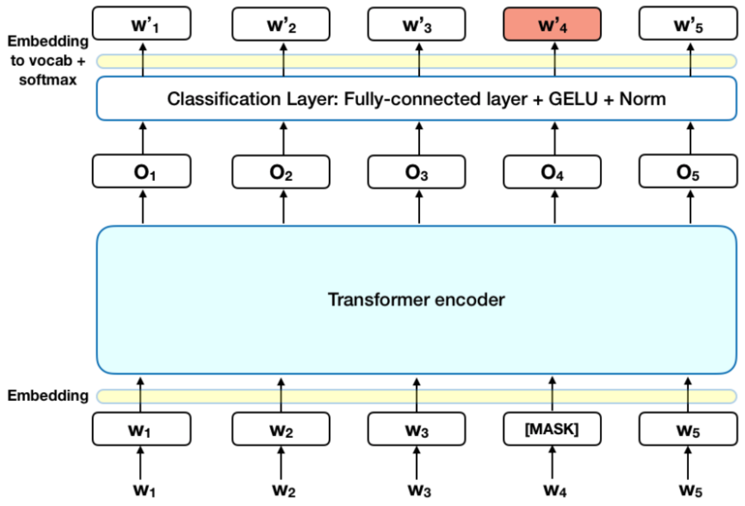
The LSTM is capable of maintaining long-term dependencies as at each step of the network by maintaining an additional cell state to store contexts that are still relevant for future steps, but are not necessarily recent in the sequence. Each step updates this long-term memory by using sigmoid activation to forget and add new contexts based on the new step's inputs and the current hidden state. A further sigmoid activation is then used to similarly selectively update the next step's hidden state. The below cell represents a step in the LSTM sequence.



To better capture contextual dependencies in our reviews, we used a model with bidirectional LSTM layers. This creates an LSTM structure that captures contextual relationships moving both forward and backward in the review text.

BERT

Google's BERT (Bidirectional Encoder Representations from Transformers) model was developed in 2018, and is now being used in their search engine.[2] At the time of its release, it achieved state-of-the-art results on a variety of natural language processing tasks. It is an autoencoding model which was trained on a large corpus of unlabeled text using "masking," which is essentially trying to predict a randomly hidden word from the surrounding text. It is bidirectional because it considers context both to the left and right of a given word. BERT actually encodes its representations at the sub-word level, meaning its vocabulary ranges from entire words to prefixes and suffixes and down to single characters. This eliminates the "out of vocabulary" problem since any new word it comes across can be represented by a combination of subwords, including individual characters if necessary. The pretraining procedure is visualized in a simplified form below. In order to fine-tune the transformer for a classification task such as ours, an additional layer is added to the model to produce the classification output.


w'1
w'2
w'3
w'4
w'5
Embedding to vocab + softmax
Classification Layer: Fully-connected layer + GELU + Norm
O1
O2
O3
O4
O5
Transformer encoder
Embedding
w1
w2
w3
[MASK]
w5
w1
w2
w3
w4
w5

[2] https://arxiv.org/pdf/1810.04805.pdf

Model Fitting

LSTM

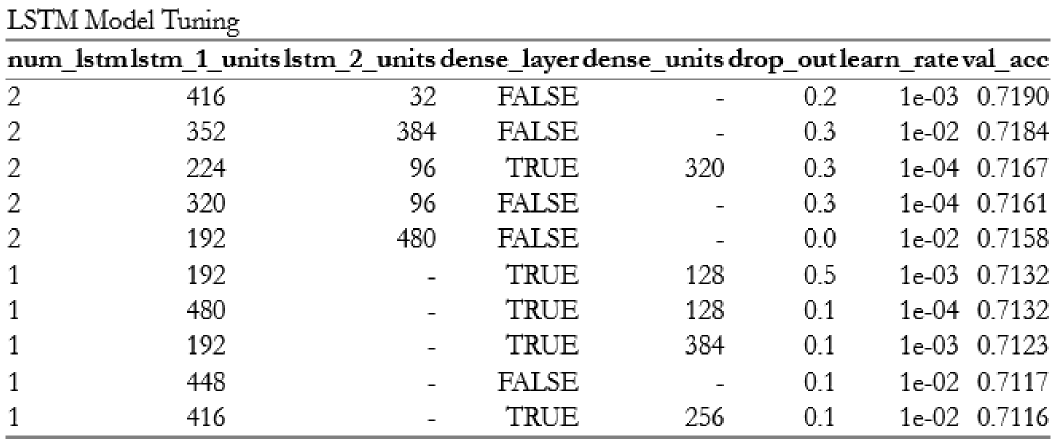
The spine of the architecture we were interested in started with an embedding layer to convert the cleaned review texts to their respective GloVe representations. Additionally, our model ends with a dense layer performing a softmax activation to convert the output up until that point to an array of predicted probabilities for each of the five review star classes.

Outside of that model spine, we considered tuning the following settings:

- ☐ Number of Bidirectional LSTM Layers (1 to 2)
- ☐ Unit count of each Bidirectional LSTM Layer (32 to 512)
- ☐ Inclusion of a non-softmax Dense Layer
- ☐ Unit count of additional Dense Layer
- ☐ Proportion of gradient updates to drop out (0 to 0.5)
- ☐ Learning rate of the model optimizer

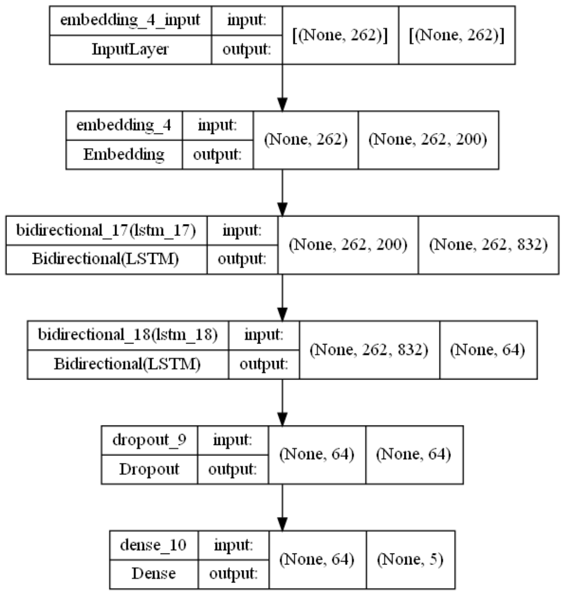
Due to the complexity of the possible model structures, and the scale of the review count and feature set, only a subset of 15 parameter setting permutations were evaluated. Each setting permutation was chosen randomly, and all structures were trained with a cross entropy loss target. A 20% validation set was held out and evaluated against at the end of each training epoch. Each proposed model structure trained for at most 15 epochs, or until the model's classification accuracy on the validation set stagnated or dropped for 3 epochs running. Each run used a batch size of 128 reviews to iterate through the training set.

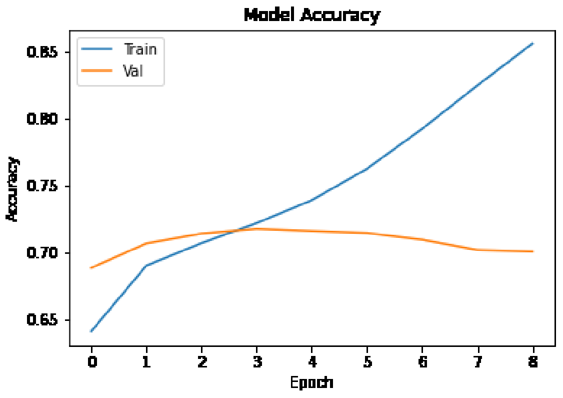
Below we have the top ten parameter settings and their maximum validation accuracy:


LSTM Model Tuning

num_lstm	lstm_1_units	lstm_2_units	dense_layer	dense_units	drop_out	learn_rate	val_acc
2	416	32	FALSE	-	0.2	1e-03	0.7190
2	352	384	FALSE	-	0.3	1e-02	0.7184
2	224	96	TRUE	320	0.3	1e-04	0.7167
2	320	96	FALSE	-	0.3	1e-04	0.7161
2	192	480	FALSE	-	0.0	1e-02	0.7158
1	192	-	TRUE	128	0.5	1e-03	0.7132
1	480	-	TRUE	128	0.1	1e-04	0.7132
1	192	-	TRUE	384	0.1	1e-03	0.7123
1	448	-	FALSE	-	0.1	1e-02	0.7117
1	416	-	TRUE	256	0.1	1e-02	0.7116

Our best-performing model by validation accuracy then comes from a model with two bidirectional LSTM layers and a moderate dropout. In general, it appears that adding a second hidden layer, be it dense or ideally LSTM, helped boost model performance. Our final structure is displayed below:


embedding_4_input | input: | [(None, 262)] | [(None, 262)]
InputLayer | output:
embedding_4 | input: | (None, 262) | (None, 262, 200)
Embedding | output:
bidirectional_17(lstm_17) | input: | (None, 262, 200) | (None, 262, 832)
Bidirectional(LSTM) | output:
bidirectional_18(lstm_18) | input: | (None, 262, 832) | (None, 64)
Bidirectional(LSTM) | output:
dropout_9 | input: | (None, 64) | (None, 64)
Dropout | output:
dense_10 | input: | (None, 64) | (None, 5)
Dense | output:



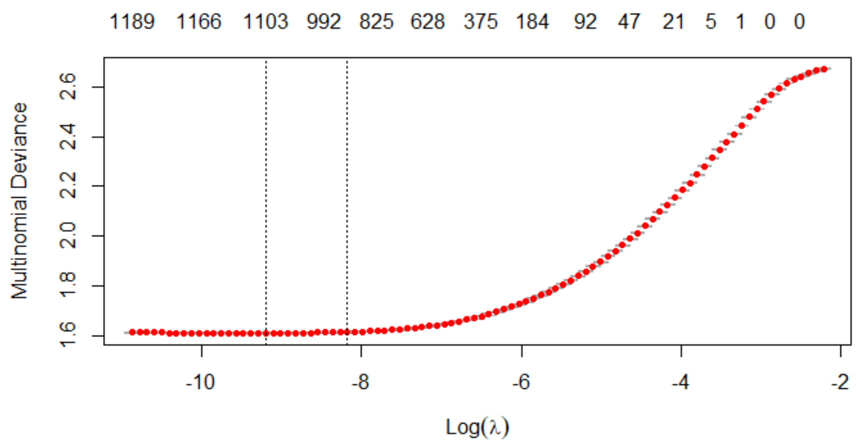
We retrained this model for a maximum of 15 epochs. After 4 epochs, however, we see that the validation accuracy begins to fall off. As a result, our final model is trained up through 4 epochs.

BERT

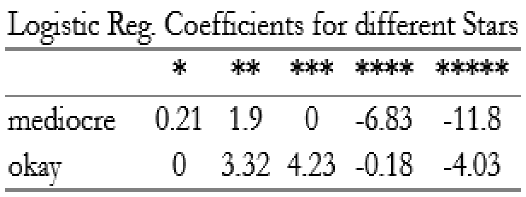
Fitting a text-classification model with BERT is an example of transfer learning, where we take the pretrained weights set on hundreds of millions of parameters used for encoding in the original transformer, add another layer for our classification output, and begin training with our labeled training data. We attempted our own fine-tuning with the “simpletransfomers” Python package. The only text preprocessing required was removing the “\n” code for a new line. Unfortunately, this process ended up being too time consuming as we had already devoted much of our time to fitting and tuning the LSTM. We were able to train a model using about 0.1 percent of the training data but this did not result in accurate predictions. Instead, we decided to use an off-the-shelf model from the Hugging Face “transformers” library which had been fine-tuned for this task using an older version of the Yelp dataset. We then used this tokenizer and model to make predictions on New Orleans restaurant reviews, treating them as a state-of-the-art baseline against which to compare our LSTM.

Multinomial Regression

We tuned a multinomial logistic regression by using 10-fold cross-validation to assess the predictive ability of the prospective model when penalized by various lasso penalty terms. The cross-validation was repeated for 50 different penalty terms, ranging from 0.01 to 0.00002. We opted for the penalty of 0.0002822, as it was within 1 standard deviation of the minimized cross-validation multinomial deviance, but led to a simpler model.



We considered a multinomial as opposed to an ordinal logistic regression as, despite the star labels being innately ordered, the effect of many words in our TF-IDF vocabulary are not constant between all star levels. For instance, words that are middling in sentiment have effects that direct the model towards the middle of the ordinal scale, rather than up or down it..


Logistic Reg. Coefficients for different Stars

	*	**	***	****	*****
mediocre	0.21	1.9	0	-6.83	-11.8
okay	0	3.32	4.23	-0.18	-4.03

Random Forest

To tune our random forest classifier, we considered the following ranges of hyperparameters:

- ☐ Mtry: the count of token features randomly available for each tree (6 to 50)
- ☐ Node Size: the minimum count of reviews required in each terminal node (250 to 1000)
- ☐ Sample Size: the proportion of the training set used in each tree (0.5 to 0.8)
- ☐ Tree Depth: the allowed depth of each tree (1 to 9)

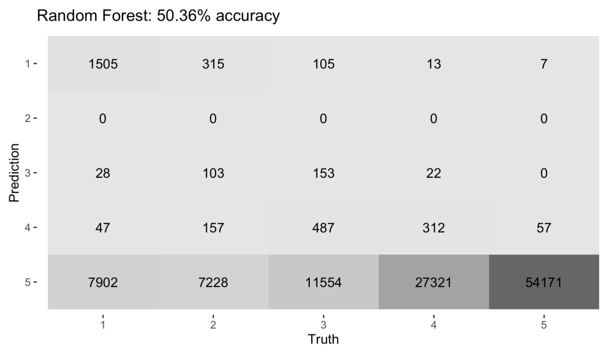
Due to the scale of the data, larger values of Mtry and Tree Depth led to more prohibitive computational times. This was similar for smaller values of node size. Parameter ranges were selected with these costs in mind. We also limited our input tokens to those that survived our ultimate lasso logistic regression model.

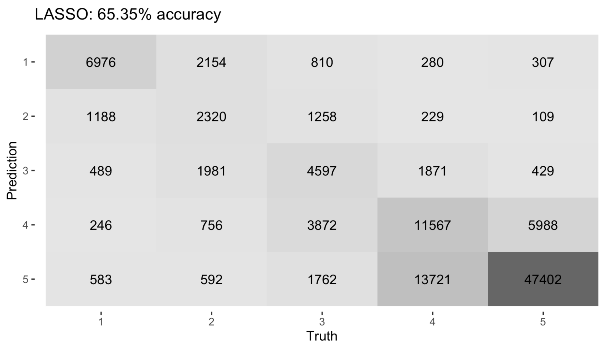
A random selection of 100 parameter permutations were used to build random forest classifiers with 250 trees each. Out-of-bag classification error rate was used to select the optimal model.

Evaluation and Comparison

All final fitted model structures were evaluated on the same 25% of the data held out as a test set. In our baseline Machine Learning models, we see that the Random Forest classifier in particular struggled with the high imbalance of stars in the dataset. It has decent precision when it does predict a review to be 1 star, but ultimately ends up predicting most reviews to be 5 stars, and is quite unable to identify reasons a review ends up as 2-4 stars.

The lasso-penalized multinomial regression does a fair bit better. It provides a more accurate distribution of predicted reviews, but still struggles at identifying non-polarizing reviews with low precision and recall for 2-4 stars.

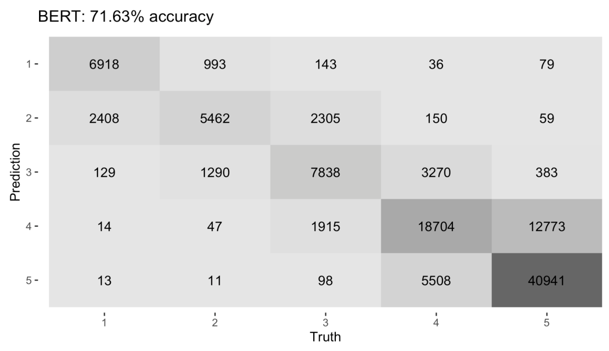


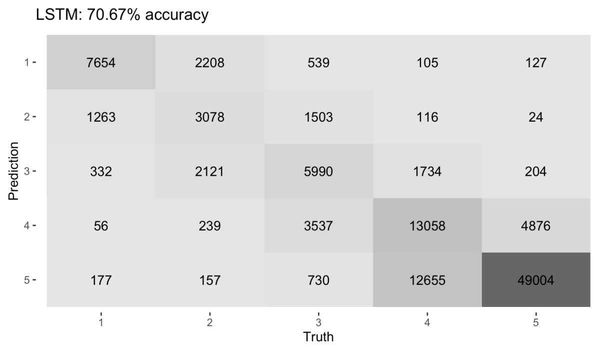


	Precision	Recall	F1	Balanced Accuracy
Class: 1	0.774	0.159	0.263	0.577
Class: 2	NA	0.000	NA	0.500
Class: 3	0.500	0.012	0.024	0.505
Class: 4	0.294	0.011	0.022	0.501
Class: 5	0.501	0.999	0.667	0.528

	Precision	Recall	F1	Balanced Accuracy
Class: 1	0.663	0.736	0.697	0.850
Class: 2	0.455	0.297	0.359	0.635
Class: 3	0.491	0.374	0.424	0.663
Class: 4	0.516	0.418	0.462	0.644
Class: 5	0.740	0.874	0.801	0.792

The BERT model results in the best overall accuracy and the highest amount of nuance in predicting the non-polarized reviews. In comparison, the tuned LSTM has better recall and accuracy for the polarized reviews, but struggles similarly to the multinomial regression in predicting middling reviews (albeit to a lesser degree). This leads to a lesser, but relatively comparable, accuracy than the BERT model, but significantly lower F1 scores for the middle three classes.





	Precision	Recall	F1	Balanced Accuracy
Class: 1	0.847	0.730	0.784	0.859
Class: 2	0.526	0.700	0.601	0.826
Class: 3	0.607	0.637	0.622	0.793
Class: 4	0.559	0.676	0.612	0.750
Class: 5	0.879	0.755	0.812	0.828

	Precision	Recall	F1	Balanced Accuracy
Class: 1	0.720	0.807	0.761	0.889
Class: 2	0.514	0.394	0.447	0.683
Class: 3	0.577	0.487	0.528	0.721
Class: 4	0.600	0.472	0.528	0.684
Class: 5	0.781	0.904	0.838	0.832

Ultimately, our model performance shows that the LSTM is a capable predictor of review sentiment, but it is not as capable at identifying complex, non-extreme sentiment.

Test Set MAEs

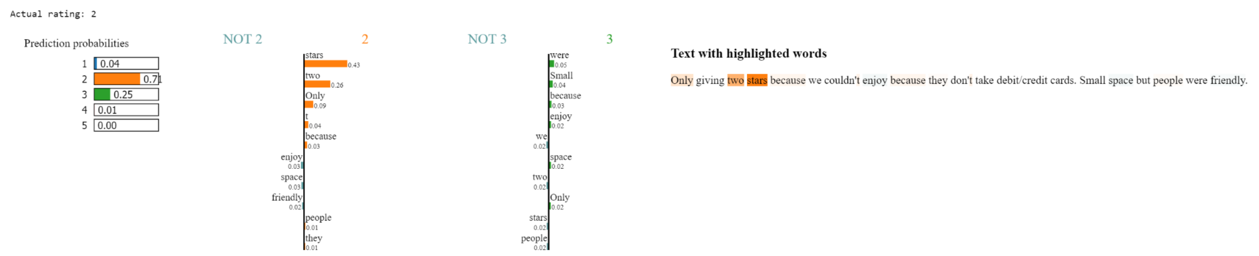
BERT	**LASSO**	**RF**	**LSTM**	**All 4s**
0.2968	0.4326	0.9462	0.327	0.9919

Since the class labels correspond to integers, we can also calculate mean absolute errors from the predicted class to the true class. Numeric errors of the model's prediction labels followed the same pattern, with BERT (0.2968) performing slightly better than the LSTM (0.3270), followed by LASSO (0.4326). The random forest (0.9462) only had a slightly lower MAE than would result from simply predicting every rating to be four stars (0.9919).

Notes on LSTM's Predictions

To get a sense of the capabilities and failings of our proposed LSTM architecture, we used the *lime* package in python to identify specific tokens in a given predicted review that were important in the model's classification.

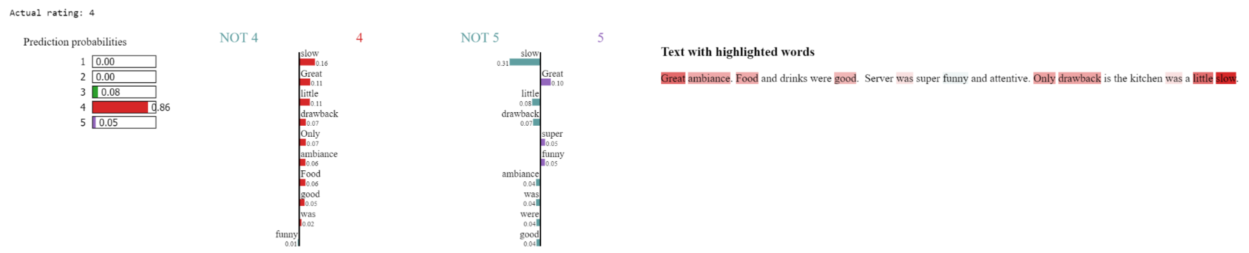
Below we see that the LSTM succeeds in identifying strongly indicative words and phrases, using *“two stars”* to make an accurate 2 star review prediction.



Text with highlighted words

Only giving two stars because we couldn't enjoy because they don't take debit/credit cards. Small space but people were friendly.

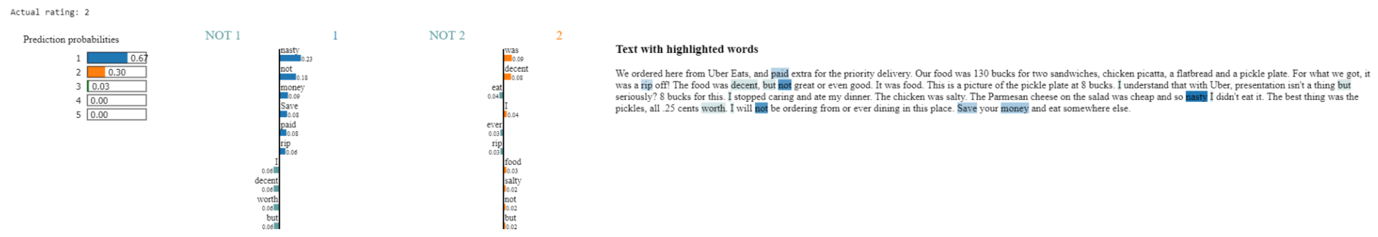
In the following example, we see that the model is also capable of distinguishing between extreme and more moderate sentiment. Here it acknowledges the strongly positive words, but also acknowledges *“slow,” “good”* and *“drawback”* as either negative or not extremely positive, leading to an accurate 4 star.



Text with highlighted words

Great ambiance. Food and drinks were good. Server was super funny and attentive. Only drawback is the kitchen was a little slow.

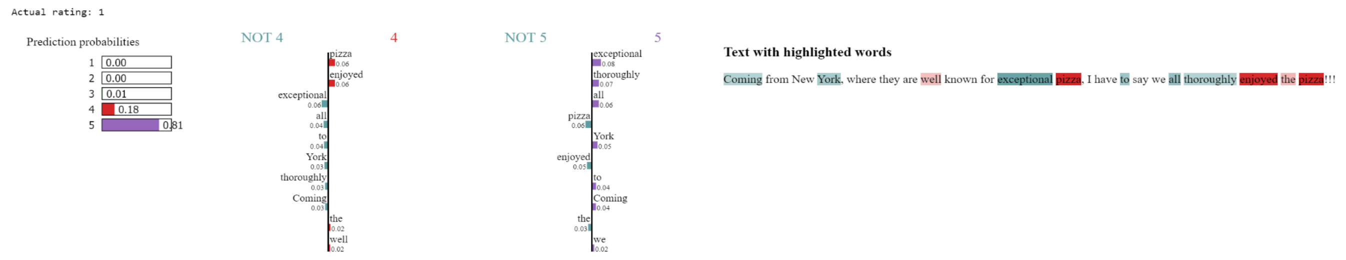
As noted in the results section, one of the model’s shortcomings is in its over-classification of 1 and 5 star ratings. Here, we see the model prefers a 1 star rating due to strong words like *“nasty”* and several bigrams with *“not.”* However, there is some mixed sentiment that leads to an actual 2-star rating.



Text with highlighted words

We ordered here from Uber Eats, and paid extra for the priority delivery. Our food was 130 bucks for two sandwiches, chicken picatta, a flatbread and a pickle plate. For what we got, it was a rip off! The food was decent, but not great or even good. It was food. This is a picture of the pickle plate at 8 bucks. I understand that with Uber, presentation isn't a thing but seriously? 8 bucks for this. I stopped caring and ate my dinner. The chicken was salty. The Parmesan cheese on the salad was cheap and so nasty I didn't eat it. The best thing was the pickles, all .25 cents worth. I will not be ordering from or ever dining in this place. Save your money and eat somewhere else.

Further exploration reveals issues with user error, as the model occasionally predicts 1 star reviews as 5 star. This example below shows that this is likely due to users not understanding the rating scheme at Yelp.



Text with highlighted words

Coming from New York, where they are well known for exceptional pizza, I have to say we all thoroughly enjoyed the pizza!!!

Conclusion

We were pleased to see that our LSTM performed only slightly worse than what could be considered the state-of-the-art BERT model in both classification accuracy and mean absolute error, and it vastly outperformed the traditional machine learning benchmarks of random forest and LASSO regression. Overall, we would judge our LSTM to be a capable model for businesses seeking to generate review ratings from unlabeled text. We think it should be able to classify reviews on Twitter and similar sites as if they were Yelp reviews, and we would be interested in applying our model to such data in the future. Additional future directions in this task could involve training an LSTM for regression on star rating treated as a numeric value rather than considering the ratings to be categorical. We also may be interested in predicting other review metrics such as how “useful” a review was judged to be by other users.