YelpHelp – A Yelp Review Star Classifier

Final Project for CS 182, Spring 2020

Contributors:

- Connor Lafferty (`cclafferty5@berkeley.edu`)
- Ajay Raj (`araj@berkeley.edu`)
- Kartik Kapur (`kapurkartik@berkeley.edu`)
- Suraj Rampure (`suraj.rampure@berkeley.edu`)

Problem Statement and Background

Yelp (http://yelp.com) is an online platform that people use in order to rate businesses from all sectors. Users can rate businesses on an integer scale of 1 to 5 “stars”. Along with a star rating, users can also provide a written review, detailing their experience(s) with said business and adding more context to their star rating.

Our problem is to predict the star rating that will be given for a particular written review. Specifically, our task is that of creating a Natural Language Processing (NLP) model that classifies text into one of five categories (the five possible star ratings).

Our dataset comes from Yelp itself (link (https://www.yelp.com/dataset/)). Our training data (which is a subset of the entire training data; the rest is used for evaluation purposes) consists of 533581 rating and review pairs. The distribution of reviews provided for each star rating is given below.

Star	Count
1	129878
2	35858
3	34263
4	72422
5	261160

The significant class imbalance proved to be a challenge for many of our models, which we describe in detail below.

Another challenge is that the method in which we communicate is quite nuanced; the sentiment of a word doesn’t necessarily correlate with the sentiment of the sentence it is in. In the context of reviews, for instance, one may think that words such as “bad” indicate that a review is negative; however, there are times where such terms are used to describe a business favorably in reference to another business. One such example of this is `“This Thai restaurant was so good, unlike the bad restaurant that I went to earlier.”`.

The two primary metrics we looked at when evaluating the effectiveness of our model were accuracy (or the proportion of exact matches) and mean absolute error, or MAE, the average number of stars a predicted rating was off by.

Our final submitted model was an ensemble of six different GRU-based networks, though we also experimented with attention-based models (BERT). Our final model was able to achieve a testing accuracy of 55.2% and a MAE of 0.847 on challenge sets 3, 5, 6, and 8. On all challenge sets (including the ones held out by course staff), our final model achieves an accuracy of 60.7%, with an MAE of 0.589. On the validation set (which consisted of 20% of the training data, selected randomly), it achieved an accuracy of ~80% and MAE of ~0.35.

Approach

Initial Model

Our first model was an LSTM, with the following characteristics:

- A vocabulary size of 100,000.
- A review length of 300 (i.e. we only looked at the first 300 words of a review).
- Dropout, with a dropping probability of $p = 0.3$.
- Word embeddings of size 256.
- A dense layer, with 256 inputs and 5 outputs (the number of target classes).
- Sparse categorical cross entropy as the loss metric.

For this model, we used the word embeddings provided in Homework 3 for this course (specifically, `wp_vocab10000.vocab`).

GRU Cells

Our second model followed a similar architecture to our first model, though with the following improvements:

- We used a GRU Cell rather than an LSTM Cell. Empirically we found GRU cells to produce more accurate models.
- We made the model bi-directional. This was done in order to put equal emphasis on both the start and end of reviews, as opposed to only being at the end (which is a common issue when dealing with LSTMs).
- We decided to use GLoVe Embeddings (https://nlp.stanford.edu/projects/glove/) of size 100 in order to only use known words to predict the sentiment of the review. This part in particular was not that useful because GLoVe ended up finding only 67,209 of the words in our training set, and didn't have embeddings for the other 32,790 words (i.e. ~1/3 of the words were not found). We completely scrapped the idea of only utilizing pre-trained word embeddings after this experimentation, but kept the word embedding size as 100 for all remaining models.

After some experimentation, we decided to remove the GLoVe embeddings all together. This yielded our third model, and one that would serve as a strong foundation for further experimentation. Other changes that came with this third model included:

- Reducing the vocabulary size from 100,000 to 50,000, as this counters overfitting in some sense.
- Reducing review length from 300 to 150, as the key sentiment of a review typically is at the start.

Henceforth the above model will be referred to as `GRU_BI` .

Changing the Loss Function

Building on top of `GRU_BI` , we first decided to tweak the loss function. So far, we'd been employing sparse categorical cross entropy loss (SCCE).

One idea was to **weight the loss function by the stars they predicted**. The motivation behind doing this was due to the severe class imbalance prevalent in our training set (see the table provided in the Problem Statement and Background). We found that weighting the loss function by multiplying the loss by $\frac{1}{\text{frequency(class)}}$ yielded a good result (i.e putting more weight on lower frequency stars). This did work well; in particular, we found that this weighting scheme helped us predict ratings in the 2 and 3 star range. However, we did have to upper-bound the maximum value of $\frac{1}{\text{frequency(class)}}$ as for the low frequency stars (2 and 3), there would be too much weight. The final weights we settled on (after some tweaking) were [2, 5, 5, 3, 1] (with the weighting being for predictions of 1 star, 2 stars, 3 stars, 4 stars, and 5 stars respectively). We discuss a model we trained using this loss below.

Another modification regarding loss functions we made was to instead use **squared loss**, i.e. $(s_{actual} - s_{predicted})^2$, instead of sparse categorical cross entropy. We trained a model using a weighted squared loss (i.e. a combination of this technique and the one mentioned above), which will be referred to as `GRU_BI_WSL` .

Character Embeddings

A problem that we saw was that words that were unknown (i.e not in our vocabulary) have the same embeddings. In order to account for this, we decided to utilize character embeddings. Character embeddings follow the same idea as word embeddings, except they are done on a character level. These can be preferential to word embeddings because there are many less characters than words which will decreased the amount of vectors needed since we have many less characters than possible words. Additionally, with character embeddings, it becomes possible to handle out of vocabulary words unlike word embeddings. This is useful in our case because our training data may not consist of all the possible words that our model may be tested on. Additionally, character embeddings are able to handle typos better than word embeddings since all non-existant words are equivalent in word embeddings, while with character embeddings it does not matter if the word was seen or not.

Our character embeddings matrix was dimension 72x50, since there are 72 possible symbols at play (upper and lower-case alphabet, plus digits and special characters), and we set the embedding dimension to 50. So, to create an embedding for a word, we take the normal word embedding (either using pretrained ones like GLoVe or training the embeddings in the model) and then append 50 extra features which have to do with the first 5 characters in the word (5 was all that the RAM on Google colab could handle in preprocessing all of the training data), increasing our embedding size to 150 total.

We trained multiple models using character embeddings:

- `GRU_BI_CHAR` , which featured SCCE (Sprase Categorical Cross Entropy) as the loss.
- `GRU_BI_CHAR_WSCC` , which used a star-weighted version of SCCE.

Attention-based Models

So far, our models have been derivatives of recurrent neural networks (RNNs). As seen in CS 182, attention-based models (https://arxiv.org/pdf/1706.03762.pdf) are also quite powerful, so we decided to experiment with them. We decided to utilize a Bidirectional Encoder Representations from Transformers (or BERT) model.

In order to classify the input, add a special <CLS> token to the beginning of the input, and take the outputs of the last layer of attention (which are contextual embeddings of the input words) and create the outputs logits by adding a dense layer on the contextual embedding that corresponds to <CLS>. Our final BERT model will be referred to as `BERT_MODEL` .

The rationale for using this model is that even with the improvements of using LSTM/GRU cells instead of RNN cells, if the LSTM isn't bidirectional, for example, the gradients are less affected by the words at the beginning of the sentence, just by virtue of the fact that the weights are further from the loss in the computation graph. Further, it is the case that the training time is linear in the size of the input for LSTM cells, because the all 300 tokens need to be fed into the LSTM cell sequentially in order to evaluate the input. With attention models, the computation isn't sequential per layer, so it can be parallelized, and matrix multiplication optimization helps with the overall training/test time. Especially in this problem, when the output isn't a sequence, neither the training nor the test time is dependent on the size of the input—this resulted in training times of around 10 minutes per epoch, whereas the other sequential models took around 45 minutes per epoch.

Ensembling

We tried several different models, and they each had their own strengths and weaknesses on the various challenge sets released by course staff. In order to leverage these, we decided to create an ensemble, consisting of a subset of our models. Specifically, we would feed the classification task into each model, and return a weighted combination of the predictions made by each model.

Our final model was an ensemble consisting of the following models with their corresponding weights:

Model	Weight	Description
`GLOVE_GRU_BI`	0.05	Is bidirectional, uses GLoVe embeddings, and GRU cells.
`GLOVE_GRU_BI_CHAR`	0.15	Is bidirectional, uses GLoVe embeddings, GRU cells, and character embeddings.
`GRU_BI`	0.2	Is bidirectional, uses GRU cells, has a vocabulary size of 50,000, and uses SCCE.
`GRU_BI_SQUARED_LOSS`	0.0	Is bidirectional, uses GRU cells, has a vocabulary size of 50,000, and uses squared loss instead of SCCE.
`GRU_BI_WEIGHTED_SQUARED_LOSS`	0.25	Is bidirectional, uses GRU cells, has a vocabulary size of 50,000, and uses weighted squared loss.
`GRU_BI_CHAR_WEIGHTED`	0.15	Is bidirectional, uses GRU cells, uses character embeddings, and weighted SCCE.
`BERT_MODEL`	0.0	The aforementioned BERT model, vocabulary size 50,000, and weighted SCCE.

In order to determine the optimal weights to apply to each model, we performed an exhaustive search, incrementing by steps of 0.05.

Results

We tested our models on both a validation set, which consisted of a random subset of 20% of the training data we were provided, and challenge sets 3, 5, 6, and 8 provided by course staff.

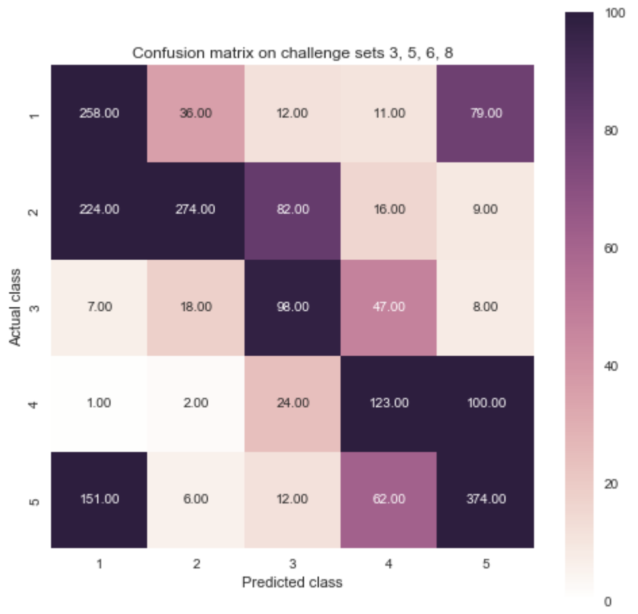
Our initial model received a validation accuracy of 76.9%, with a MAE of 0.30. Our final model had similar validation numbers, but validation accuracy isn't a great indicator in this case because the dataset we were provided consisted of relatively simple and clean reviews to classify. A better indicator is the challenge sets.

Our overall accuracy on all challenge sets (including those held out by course staff, i.e. our score on the leaderboard) is 60.7%, with an MAE of 0.589. However, only challenge sets 3, 5, 6, and 8 were released, and so we used those to perform more detailed analytics. On those 4 sets, our final model achieves an accuracy of 55.2% and a MAE of 0.847.

We looked through some of the reviews in the released challenge sets to get a sense of the kind of review each contained. We noticed the following:

- Challenge Set 3: Mostly an even distribution of ratings.
- Challenge Set 5: All 2 star ratings (which were quite difficult to classify).
- Challenge Set 6: All 5 star ratings (which were relatively easy to classify).
- Challenge Set 8: All short reviews (~70 words or fewer).

Below is a confusion matrix of our final model on all of the released challenge sets (3, 5, 6, 8).



The model tends to confuse 1 and 2 star reviews, and 4 and 5 star reviews. It also predicted many 5 star reviews to be 1 star (151 to be precise).

Here is a per-class accuracy and MAE breakdown, again on the four released challenge sets.

Star Rating	Accuracy	MAE
1	65.2%	1.033
2	45.3%	0.603
3	55.1%	0.543
4	49.2%	0.524
5	61.8%	1.170

Lastly, here is a per-challenge set breakdown of accuracy and MAE:

Challenge number	Accuracy	MAE
3	65.0%	0.284
5	43.4%	0.620
6	45.6%	1.974
8	67.0%	0.412

Tools

All of our code was written in Python, in an iPython Notebook. We used `tensorflow` and `keras` for the majority of our modeling. In particular, these tools allowed us to experiment quite easily with different recurrent cell types (vanilla RNN, LSTM, GRU), loss functions, and solvers, without having to write very much code.

In our baseline model we used Tensorflow 1.0 (since this was largely re-adapted from Homework 3), though after that we transitioned to Tensorflow 2.0. This was primarily because Tensorflow 2 (and Keras) allowed us to build models (and in particular, ensemble models) far quicker.

The BERT model was created using this open source Keras BERT library (https://github.com/CyberZHG/keras-bert).

Our computing environment was Google Colab. Colab provides free GPU hours which proved to be vital, and it also made collaboration far easier.

Lessons Learned

The biggest takeaway we have from this project was that it's better to build multiple models that are fit for specific tasks and weight them accordingly, as opposed to trying to build a single model that does everything well. Our ensemble model allowed us to achieve accuracies on the challenge sets that none of our individual models were able to reach.

We were also surprised that the pretrained GLoVe embeddings (that are cited in many papers for setting up different models) served to limit our models instead of helping them. It makes sense that training the embeddings from scratch contextualizes the embeddings to the data, but perhaps it did not overfit because the complexity of our models was not too high (all GRU cells only had one layer, and the BERT model was only 6 layers, when usually transformer models use upwards of 10 layers of attention).

Team Contributions

Connor (40%): Wrote software to conduct quick, large-scale testing over many models. Mainly developed design/workflow of overall project, including notebooks and source code.

Ajay (25%): Worked on the models that added character embeddings, and the BERT model.

Kartik (17.5%): Worked on the report and various experimentation.

Suraj (17.5%): Worked primarily on the report.

References

1. Yelp Dataset https://www.yelp.com/dataset/ (https://www.yelp.com/dataset/)

2. Illustrated Guide to LSTM's and GRU's https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21 (https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21)
3. Homework 3, CS 182/282A at UC Berkeley, Spring 2020
4. Multi-hot Sparse Categorical Cross-entropy https://cwiki.apache.org/confluence/display/MXNET/Multi-hot+Sparse+Categorical+Cross-entropy (https://cwiki.apache.org/confluence/display/MXNET/Multi-hot+Sparse+Categorical+Cross-entropy)
5. Attention Is All You Need https://arxiv.org/pdf/1706.03762.pdf (https://arxiv.org/pdf/1706.03762.pdf)
6. BERT Explained: State of the art language model for NLP https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270 (https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270)