CS182 Final Project: Predicting Yelp Reviews

Lawrence Wong, Tram Tran, Varun Srinivasan

May 2020

1 Problem Statement and Approach

In this project, we were given access to a dataset between the raw text of a Yelp review and the star rating that Yelp review was associated with. The dataset was pretty simply a list of raw Yelp reviews (misspellings and all) and their associated star ratings. Our goal was to design and implement a neural network that would predict the assigned star rating given the text of a Yelp review. Besides just being an interesting problem to solve, this task has a variety of potential benefits, such as allowing Yelp to "correct" the overall rating for a business to be more in line with the actual reviews than with the ratings which may be "out-of-sync" with the text of the reviews.

In solving this problem we assess models based on two metrics which we refer to in the future, the raw accuracy or percentage of correctly guessed stars (% ACC), and the average distance of a wrong answer from the correct numbers of stars (AD). We chose to approach this problem from many angles, with each member of our group exploring a different approach for this problem and attempting to optimize that approach. This gave us the maximum opportunity to find the best kind of model, and also gave us a lot of room to explore and present interesting findings about the nature of this problem. Because of this, we've chosen to present our findings approach-by-approach.

Overall, we found RNN-based models to work very well, and specically non-sequential models worked very well. Surprisingly for RNN, most perturbations we could come up with did not improve the results greatly, although for RNNs as with all of our approaches, a balanced data-set yielded improvements. Additionally, with our BERT models we found that using ordinal labels (something we hadn't heard of until this project) was beneficial in improving our model. Additionally, some "outside" the box approaches we tried like a Word2Vec model and a CNN+LSTM model, ended up not performing very well, but it was interesting to see the results nonetheless. Overall, our BERT model with ordinal labels turned out to perform the best.

2 BERT Models

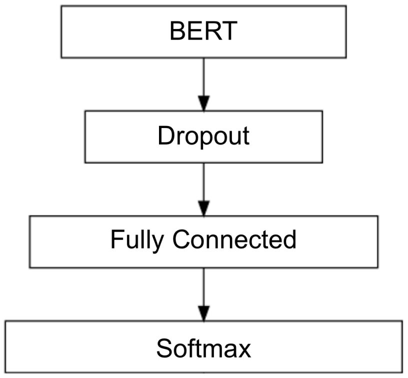
Many advances in the field of NLP have been led by pre-trained models. Therefore, we decided to further explore pre-trained BERT, a model proposed by Google. We fine-tuned (using Tensorflow) the pre-trained BERT model provided on TF-Hub and created multiple BERT-based models that are similar to a Google's BERT Colab example [2], but each with some modifications in the hidden layer's size, activation and the loss function.

2.1 Preparation

We shuffled the given data set and giving 90% for training and 10% for validation. Due to limitation in computation resources, we further split the training dataset into two equal size training sets. Each model is trained with a batch size of 16, dropout rate 0.1, learning rate 2e-5, for 2 epochs for one of the training set, and another 2 epochs for the other training set. We used BERT base uncased with max sequence length of 128 and the default optimizer from the BERT module. All text reviews fed are lowercased, tokenized, broken into WordPieces based on the pretrained BERT's vocabulary, and added special tokens "CLS" and "SEP".

2.2 Initial Model

This BERT model is almost identical to Google's example model on the right. The only modification we made was changing the output dimension of the fully connected layer to 5 as we have 5 categories. This model uses cross entropy loss and predicts the label that has the highest probability among the 5 categories.



2.3 Modified Loss

This model has the same architecture as the above BERT model and a modification in the loss function. The new loss function for each sample is

$$-\frac{1}{2}\mathbf{log}(\hat{y}_y) - \frac{1}{2} \times \prod_{i=-1}^{1} \mathbf{log}(\hat{y}_{y+i})$$

where $\hat{y}$ is the output of softmax and y is the index of the true label. If $y + i < 0$ or $y + i > 4$ then $\mathbf{log}(\hat{y}_{y+i}) = 1$. The idea behind this loss was that we want to increase both the probability of the correct label and the sum of the probabilities of the correct label, its left neighbor, and its right neighbor.

2.4 Ordinal Labels

So far, our models do not take into account that the output labels are ordinal. After doing a little research on ordinal regression and deep learning, we decided to implement a new BERT model inspired by a paper that we read [3]. First, we modified the fully connected layer output dimension to 4, used sigmoid activation, and removed softmax. Secondly, we changed the loss function of each sample to

$$loss = -\sum_{i=0}^{i=3} e_i \mathbf{log}(\hat{y}_i) + (1 - e_i)\mathbf{log}(1 - \hat{y}_i)$$

where e_i is the i+1th element of the true label's embbedding $\{\mathbf{label} \rightarrow \mathbf{embedding} : 1 \rightarrow [0,0,0,0], 2 \rightarrow [1,0,0,0], 3 \rightarrow [1,1,0,0], 4 \rightarrow [1,1,1,0], 5 \rightarrow [1,1,1,1]\}$. The predicted label is (1 + the index of the first element of the sigmoid output that has value less than 0.5) or (5 if all values $\geq$ 0.5). With this implementation, a review with a rating of k is classified automatically as having ratings 1,...,k-1 too. Therefore, we impose an ordering on the output labels of the model.

2.5 Ensemble of Binary Classsifiers

Our final idea for BERT was an ensemble of binary classifiers. This model was created in an attempt to reduce the effect of imbalanced ratings distribution in the dataset (1 and 5 have a lots more samples than other labels). We created five binary classifiers, one per rating category. They were almost identical to the initial BERT model, the only difference was the fully connected layer's output dimension was 2 and not 5. Each classifier was assigned a different rating category and was trained to predict the probability of a review is in its assigned rating category. Moreover, all classifiers were trained on a balanced dataset, 50% samples in the classifier's assigned category and 50% are not in the category. At inference time, the ensemble model predicts the category assigned to the classifier that outputs the highest probability.

2.6 Results

As mentioned above, the given dataset is imbalanced because there are a lot more ratings with 1 and 5 stars than other numbers. Therefore, we decided to test all BERT models on 2 datasets, one with imbalanced ratings distribution (similar to the given dataset) and another one with balanced distribution (equal number of samples per category)

2.6.1 Imbalanced Set

Label 1: 12988
Label 2: 3609
Label 3: 3429
Label 4: 7228
Label 5: 26105
Total: 53359 samples

Model	% ACC	AD
Initial Model	79.60	0.246
Ordinal Labels	79.47	0.244
Modified Loss	79.52	0.244
Binary Classifiers	76.74	0.288
RNN	73.27	0.386

Table 2: Model Performance Data

For this dataset, the initial model has highest accuracy and a little higher AD. We noticed that other models have much less "extreme" misclassified samples (1 predicted as 5 and vice versa) and more off-by-1 samples. Moreover, the initial model has lower precisions for labels 2,3,4 than other models. The ensemble of binary classifiers (EBL) has the worst accuracy and AD. We suspect this is because we trained EBL on uniform distributions. Surprisingly, EBL does a lot better on labels 3,4 but worse on 2.

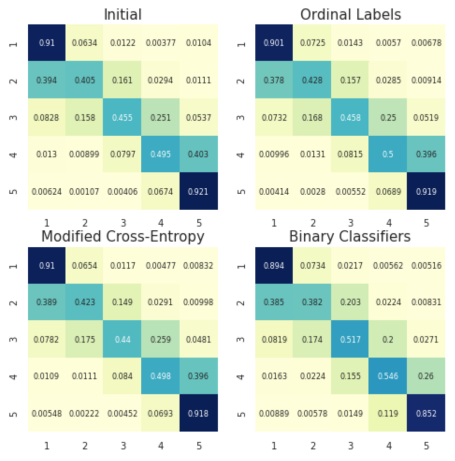


Figure 4: Normalized Confusion Matrices

2.6.2 Balanced Set

Each Label: 3429
Total: 17145 samples

Model	% Accuracy	AD
Initial Model	63.68	0.424
Ordinal Labels	64.12	0.416
Modified Loss	63.62	0.423
Binary Classifiers	63.73	0.424
RNN	52.06	0.643

Table 3: BERT Model Performance Data

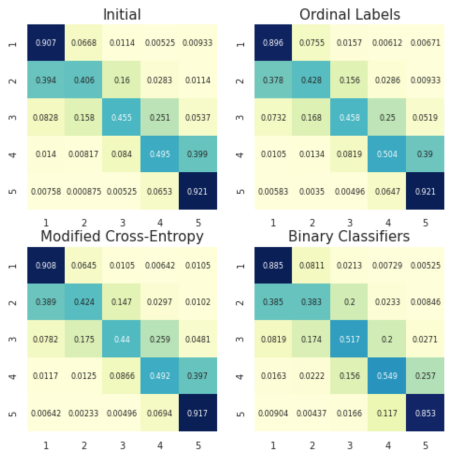


Figure 5: Normalized Confusion Matrices

The model with ordinal labels (MOL) works the best for this dataset. Also as expected, EBL is on par with other models if the data's distribution is uniform. After looking at the results, we concluded that MOL is the most robust among all BERT models. It has lowest ADs and good accuracies for both test sets.