Yelp Review Rating Prediction

with RoBERTa and Data Augmentation

Jacy Zeng **Sasha Manghise**

1. Introduction

Text classification is a fundamental task in NLP which assigns a set of predefined categories or labels to a body of open-ended text. From a technical standpoint, classifying user-generated text poses an extremely difficult task. The human language is highly nuanced and diverse, and the ability to analyze something as unstructured as text data is applicable in many fields of research. Such prediction models can be used for sentiment analysis, language detection, or speech recognition - our report will explore another application known as *rating prediction*.

Given Yelp's latest release of over 6 million restaurant reviews, we designed and trained a text classifier that predicts a user's star rating from their text review. A review consists of a user's written description of their experience at a restaurant, accompanied by a star rating from 1 to 5 which denotes the user's overall opinion of the business. A 1-star review generally indicates a 'terrible' experience, while a 5-star review indicates a 'great' experience. With over 184 million reviews worldwide, Yelp is one of the most popular crowd-sourced review forums among both businesses and customers. Our model can be used by businesses who rely heavily on customer feedback to make data-driven decisions and improve current product offerings. More accurate and robust variants of a text classifier could also be used to compare written reviews from various sources (e.g. Amazon vs. eBay) where user reviews with similar sentiments may be classified according to different criteria or rating systems.

2. Background and Related Work

2.1. Preprocessing

Text data is typically pre-processed to remove uninformative words and only preserve those that are relevant to the classification task. We primarily used the NLTK package in Python to preprocess the text reviews into a format suitable for our model. Normalization of the data was done through lemmatization - unlike stemming, this process considers the context of a word within a sentence and has been shown to offer better precision. We removed (and replaced) contractions, stop words, punctuation, and special characters before tokenizing the data.

2.2. Transformers

Several approaches to text classification models have been explored in the literature - most recently, Google's BERT and transformer-based methods were shown to outperform many state-of-the-art models trained for such tasks. BERT is a bi-directional transformer that uses a masked language model to learn language representations of large amounts of unlabeled text data. We initially considered this model for rating prediction, but a more recent iteration on the model - RoBERTa – proved to yield better results. In comparison, RoBERTa (Liu et al., 2019), which stands for 'Robustly Optimized BERT', trains with less compute time yet 1000% more data and compute power by utilizing larger batch sizes and dynamic masking during training.

2.3. Data Augmentation

As shown in Figure 1, the training data is heavily skewed towards 5-star and 1-star reviews. In order to balance class labels and make our model more robust to a variety of inputs, we used data augmentation techniques to create more data from underrepresented categories. Data augmentation is a common practice in text classification tasks - these techniques aim to boost model performance by generating additional data from slightly perturbed versions of the existing training data.

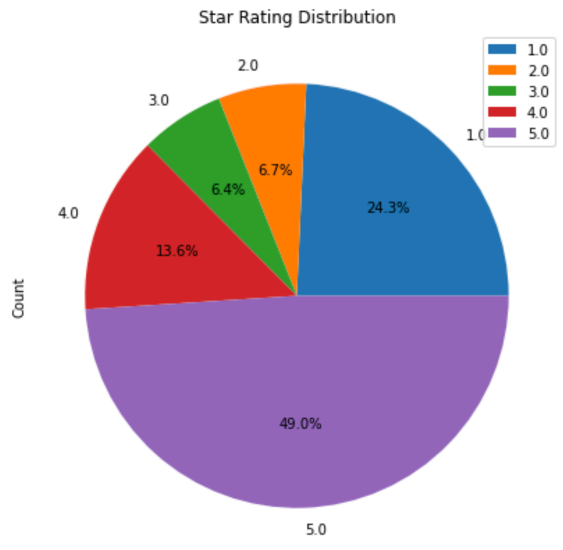

Figure 1. Distribution of Yelp review star ratings from the original training data of 533, 581 reviews.

Our model uses an approach similar to one explored in a literature review (Wei and Zou, 2019) of data augmentation techniques for training of NLP models for text classification. The paper presents a technique called EDA (easy data augmentation) which combines random deletion, random swap, random insertion, and synonym replacement. We build upon this approach by using a pre-trained RoBERTa model for random insertion and synonym replacement. Words are replaced and inserted based upon contextual word embeddings, rather than pure randomness. All four augmentation techniques were sequentially combined into a single augmenter object and used on the original training data.

Using these EDA techniques, we generated an additional 1550 more 2-star reviews, and 5158 more 3-star reviews. The majority of data augmentation was focused on 3-star reviews since reviews of a 'neutral' experience are hardest to classify.

3. Methods and Approach

Our final model is an example of transfer learning based on the previously discussed RoBERTa model architecture. We created a neural network which first runs the textual inputs through the pretrained-RoBERTa model, and then takes the results from the pre-trained model through a custom neural network architecture.

While researching transfer learning of neural networks, we discovered that it could be useful to freeze the pre-trained layer (RoBERTa) – this allows us to train other parts of the neural net to convergence with just the data, and then use a lower learning rate to unfreeze the layer in the final training phase and fine-tune the model. Freezing a layer causes the weights in the layer to remain unchanged during back propagation. To evaluate the efficacy of this technique, we trained two versions using different numbers of epochs and tested them on the validation set. The training took between 10-24 hours, and often resulted in memory overflow issues, which we remedied by decreasing epoch and batch sizes. During the first pass of the frozen RoBERTa layer, the model was able to handle larger batch sizes but with the 2nd pass of the unfrozen RoBERTa layer, we had to drastically decrease the batch size each time to avoid running out of ram. We used a 90/10 training/validation split to validate the model, and ran the validation data through the model and batches. The resulting validation accuracies were saved during testing, and the mean validation accuracy was used as a measure of model performance.

4. Results

As detailed in Table 1 of the Appendix, three different model architectures were successfully run to completion. Previous iterations which incorporated both freezing and unfreezing of layers did not show significant improvements in model performance, so our final model only uses freezing in the final RoBERTa layer.

The 2-step neural network's had a training accuracy of approximately 46%, and a validation accuracy of 49%. A second model uses a frozen RoBERTa layer with a batch size of 128, 4 epochs, and no activation function. The final model similarly uses a frozen RoBERTa layer, but with 3 epochs and a logsoftmax activation function. Both models yielded a 65% validation accuracy, and we chose to incorporate an activation function into our final model for improved accuracy.

4.1. Challenges

Tokenization of the text data proved insufficient for training purposes, leading us to encode the text using *tokenize.encode*, change the words into *input_ids,* and then mask them. After encoding the data, we also faced issues with ram space on Kaggle and Google Colab – this was resolved by increasing the number of training epochs, decreasing batch sizes, and creating 30000+ text arrays per input rather than just one array. While evaluating various combinations of neural network architectures and optimizers, training was frequently interrupted because the network did not receive the full amount of training data. Saving the model progress between iterations allowed us to easily reload the model between runs of frozen and unfrozen network layers.

5. Conclusions

5.1. Lessons Learned

This process highlighted the importance of both conducting extensive research before training, as well as ensuring that the correct data was being inputted into the neural networks. Although results may appear successful, it is nevertheless important to validate the format of both model inputs and outputs to ensure they are reasonable. By researching different transfer-learning methods, we also learned the benefits of freezing/unfreezing network layers, and how to incorporate pre-existing models into neural networks.

Preprocessing, augmenting, and cleaning text inputs also proved to be just as important as the model architectures themselves. As discussed in Section 4.1, incorrectly formatted data presents a multitude of challenges for model performance and accuracy.

For future exploration, we would test out the neural network using larger batch sizes for the second pass with the unfrozen BERT layer to analyze the impact of batch size on model accuracy. We would also like to test out more neural networks architectures and layers since we have figured out how to correctly parse the data into a format that can be directly inputted into the neural network. Finally, we would look more into the impact of freezing/unfreezing layers on model performance in text classification tasks.

6. Team Contributions

Jacy Zeng*: Developed and ran neural networks; encoded data; documented model process and accompanying challenges.

Sasha Manghise*: Performed exploratory data analysis; preprocessed, cleaned, and augmented data; formatted and edited technical report.

**Equal contributions.*

7. References

[1] Fchollet., "Keras Documentation: Transfer Learning &amp; Fine-tuning.", 15 Apr. 2020, doi: 1 May 2021

[2] Y. Liu et al., "RoBERTA: A Robustly Optimized BERT Pretraining Approach", 2019, doi: 1907.11692v1

[3] Mujtaba, Hussain. "Introduction to Transfer Learning: Transfer Learning to Detect COVID-19." 11 June 2020. Web. doi: 2 May 2021

[4] J. Wei and K. Zou, "EDA: Easy data augmentation techniques for boosting performance on text classification tasks," 2020, doi: 10.18653/v1/d19–1670.

Appendix

Table 1. Model parameters, training accuracy, and validation accuracy for various model architectures.

Model Architecture	Frozen (require_grad = false)	Parameters	Training Time	Training Accuracy	Validation Accuracy	File Location
self.roberta = RobertaModel.from_pretrained("roberta-base") self.dropout2 = nn.Dropout(0.2) self.relu = nn.ReLU() self.fc1 = nn.Linear(768,1000) self.bn1 = nn.LayerNorm(1000) self.dropout3 = nn.Dropout(0.3) self.fc2 = nn.Linear(1000,528) self.bn2 = nn.LayerNorm(528) self.dropout15 = nn.Dropout(0.15) self.fc3 = nn.Linear(528,680) self.dropout4 = nn.Dropout(0.4) self.fc4 = nn.Linear(680,5)	True	epochs = 4 batch_size = 128	8 hours	~0.68	0.6519	'/content/drive/MyDrive/182_nlp/model_freezesmall1.pt'
self.roberta = RobertaModel.from_pretrained("roberta-base") self.dropout2 = nn.Dropout(0.2) self.relu = nn.ReLU() self.fc1 = nn.Linear(768,1000) self.bn1 = nn.LayerNorm(1000) self.dropout3 = nn.Dropout(0.3) self.fc2 = nn.Linear(1000,528) self.bn2 = nn.LayerNorm(528) self.dropout15 = nn.Dropout(0.15) self.fc3 = nn.Linear(528,680) self.dropout4 = nn.Dropout(0.4) self.fc4 = nn.Linear(680,5)	False	epochs = 1 batch_size = 15 count = 0 optimizer_class = optim.Adam lr = 1e-3	6 hours	0.46466669231653	0.4888	'/content/drive/MyDrive/182_nlp/model_unfreeze1e3learningrate.pt'
FINAL MODEL: self.roberta = RobertaModel.from_pretrained('roberta-base') self.dropout2 = nn.Dropout(0.2) self.relu = nn.ReLU() self.fc1 = nn.Linear(768,1000) self.bn1 = nn.LayerNorm(1000) self.dropout3 = nn.Dropout(0.3) self.fc2 = nn.Linear(1000,528) self.dropout15 = nn.Dropout(0.15) self.fc3 = nn.Linear(528,5) self.softmax = nn.LogSoftmax(dim=1)	True	epochs = 3 batch_size = 128 optimizer_class = optim.Adam lr = 1e-3	6 hours	0.5793685929479264	0.6421	'/content/drive/MyDrive/182_nlp/model_freeze3epochswithsoftmax.pt'
self.roberta = RobertaModel.from_pretrained('roberta-base') self.rnn = nn.RNN(768,1000, n_layers, batch_first=True) self.dropout2 = nn.Dropout(0.2) self.relu = nn.ReLU() self.fc1 = nn.Linear(768,1000) self.bn1 = nn.LayerNorm(1000) self.dropout3 = nn.Dropout(0.3) self.fc2 = nn.Linear(1000,5) self.softmax = nn.LogSoftmax(dim=1)	False	epochs = 2 batch_size = 13 count = 0 optimizer_class = optim.Adam lr = 1e-3	12 hours	0.49615386813879014	Stopped 80% of way due to Kaggle timeout	