Sentiment Labelled Sentences (March 2020)

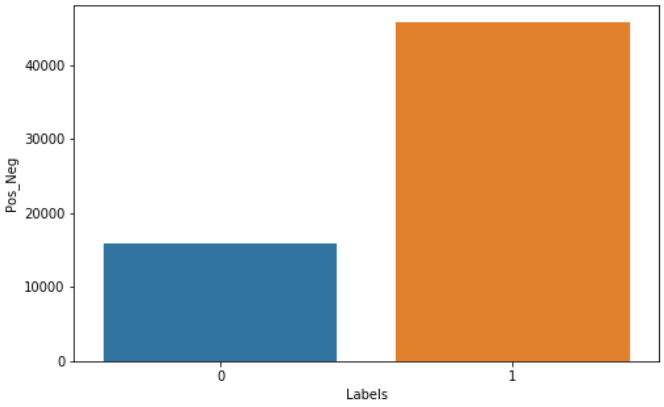
[Name Removed], Philip Rundall, and [Name Removed]

I. Our Data

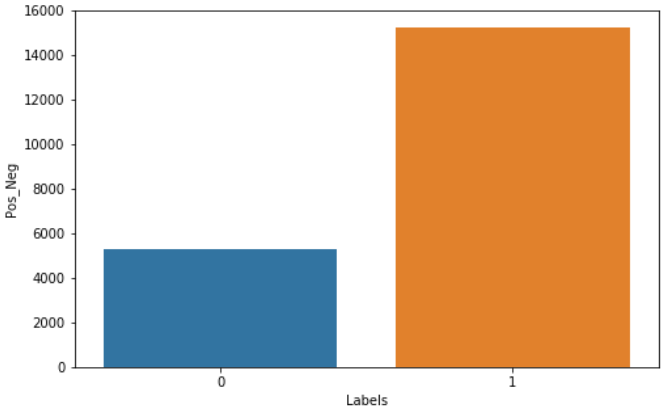
Our dataset consists of online reviews from IMDB, Yelp, and Amazon. We sourced part of our data from UC Irvine's Machine Learning Repository[1], and the rest of it from Kaggle[2]. Our total dataset is 100,000 observations.

All of these reviews are labeled as either positive, 1, or negative, 0. We used Scikit learn to split the data to be 60% training, 20% validation, and 20% test data. Overall, our dataset contains more positive than negative reviews, but all of our datasets have the same ratio of positive to negative reviews.

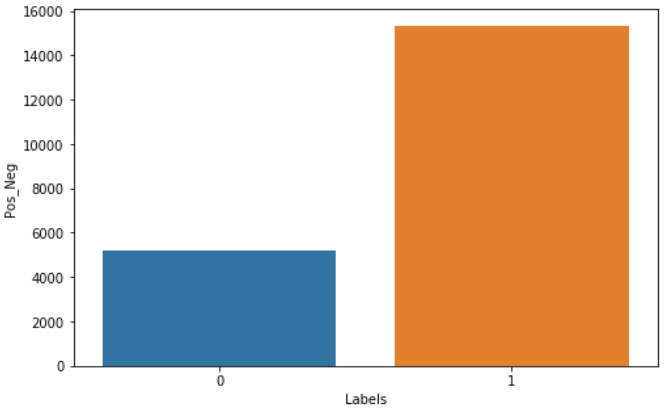
Training Data



Validation Data



Testing Data



Having more positive than negative reviews will make it difficult for all of our models to predict if a review is truly negative. This was the most data that was prelabeled for binary classification that we could find in the allowed time for this project though. In the future, this project could be improved by either increasing our dataset to make the ratio even, or resample from the negative reviews using a synthetic minority oversampling technique to make our dataset more equal.

II. Deep Learning Models

Two models which we will be using are deep learning models. Specifically, they will be variants of Recursive Neural Networks (RNN). We will also use the Multinomial Naive Bayes, Logistic Regression and Decision Trees.

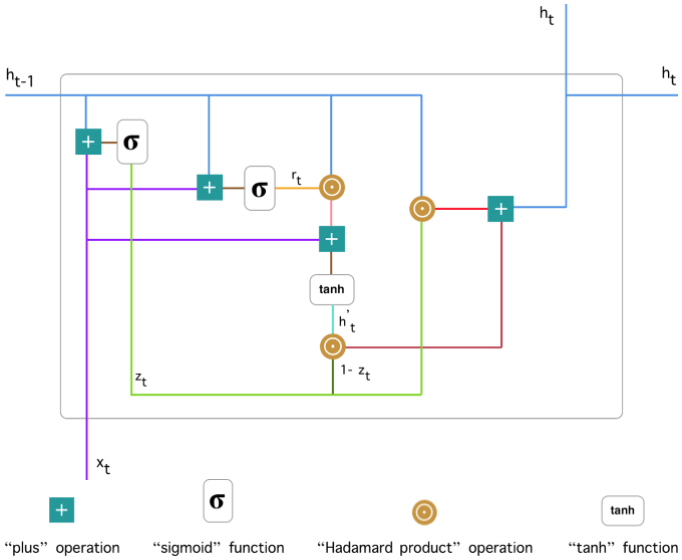
A. Gated Recursive Unit Model

The first deep learning model we will be training and testing is a Gated Recurrent Unit (GRU). It is a subset of Recursive Neural Networks (RNN). This model takes into account the order of words in a string, similar to Long Short Term Memory (LSTM) neural networks. This is important because the order of words in a string can contribute to what the true sentiment of the string is. We chose this model over an LSTM model because GRU models tend to outperform LSTM models on smaller datasets and are also more computationally efficient. Seeing as our entire dataset is about 100,000 observations and we are running our models on laptops, this seemed to be a good choice.

The GRU model works by stringing together multiple GRU units. Within each GRU unit there are two sigmoid functions, both are fed with a new data point (in this case a word from the string) and the output from the previous unit. The first sigmoid function determines how much "memory" to keep

from the previous unit and the second determines how much "memory" to forget from the previous unit. The output from both of these sigmoid functions are then fed into a hyperbolic tangent function. The output of this is then sent to two places, the next GRU unit, and the next layer, whether it is a hidden layer or the output layer.

Below is a figure[3] of a GRU unit, along with what the equations within each unit are.



$$z_t = \sigma(W^{(z)}x_t + U^{(z)}h_{t-1})$$

$$r_t = \sigma(W^{(r)}x_t + U^{(r)}h_{t-1})$$

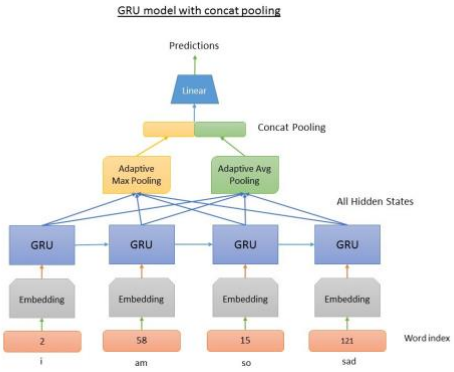
$$h_t' = \tanh(Wx_t + r_t \odot Uh_{t-1})$$

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot h_t'$$

B. *Concat Pooling Gated Recursive Unit Model*

The second deep learning model we will be training and testing is a Concat Pooling GRU model. This model is similar to the GRU model we previously introduced but it has 2 added layers before the output layer. There is a max and mean pool layer. This layer consists of a max and mean function whose inputs are all of the GRU unit outputs. Then the output of the max and mean units are fed to the Concat Pooling layer, where the mean and max are concatenated. This is then sent to the output layer. This model is more complex than our previous GRU model so it takes more computation power but will hopefully be more accurate.

Below is a figure[4] for a Concat Pooling GRU model. Our first GRU model will be similar, except without the Pooling and Concat layers.



C. *Building and Tuning Deep Learning Models*

We will use PyTorch's Torchtext library to build our model. Torchtext is a package from PyTorch specifically made for natural language processing. The original source code that our model was based off of is from a GitHub repository[5], but it has been modified to fit our data.

To clean the data, all letters are changed to lowercase and all numbers and special characters are removed. All letters are changed to lowercase because this does not affect the sentiment of a review by that much but will make the model difficult to train if left in. The same theory applies to removing special characters.

The features that we will be using in our neural networks is called an embedded word vector. For every word in a string, it will be assigned a vector of values. We will be using a pre-trained vector from Stanford's GloVe project[6]. We used the smallest pre-trained vector, glove.6B.50d.txt, because our dataset is small, and we are running our models on our laptops. Once the data is cleaned, every word in a review will be assigned to a vector from our pretrained embedded vector.

Our input layer will be 100 units, one for each word in a string. If a string is shorter than 100 words, the extra units will be padded with 0's and if it is larger than 100 words, it will only input the first 100 words. For every review, the corresponding embedding vector will be sent into a GRU unit, in order that they appear in the string.

The loss function our model uses is cross entropy loss. Because this is a classification problem, we want our loss to be larger as a sentence is more misclassified. For example, a negative sentiment sentence labeled as 0.9 will be a larger loss than a negative sentiment labeled as 0.4.

The first hyperparameter we tuned was the output layer activation function. We first used a ReLu function, however this made our model perform very poorly, under 10% accuracy on validation data. We believe this is because it made our model too complex, because all of the GRU units already contain multiple activation functions. Thus, we will leave it as a linear activation function.

The second hyperparameter we tuned was the number of hidden layers. Theoretically, 1 hidden layer is enough for any neural network, but we want to adjust this to see if we can see an increased performance. For this we will try 2, 3, and 5 hidden layers.

The final hyperparameter that we tuned was the number of epochs. Epochs is the number of times a dataset is fed forwards and backwards through a neural network during training. Generally, the larger the number of epochs, the more accurate your neural network will be. For this we will try 2, 3, and 4 epochs.

Because a neural network is a complex and computationally complicated model to train, we are keeping the number of hidden layers and epochs low. If we had access to a GPU or more RAM, we would increase the size of both of these to see if it could improve the performance.

D. Simple GRU Model Results

Our first GRU model performed best when it had 5 hidden layers and 4 epochs, as expected. Here is the accuracy after training each model and using validation data for each number of hidden layers and epochs.

		Number of Epochs		
		2	3	4
Number of Hidden Layers	2	0.7403	0.7412	0.7428
	3	0.7389	0.742	0.7418
	5	0.7418	0.7419	0.7419

As our model gets more complex, computational time is also an important factor to monitor. Here is the computational time in minutes and seconds for our first GRU model.

		Number of Epochs		
		2	3	4
Number of Hidden Layers	2	4:25	6:03	8:37
	3	6:04	9:15	12:14
	5	10:27	16:49	20:46

As we can see, the computational time does get much larger as our model gets more complex. It also gets more accurate but not by much. Even though it takes more computation time, we will treat the 5 hidden layers and 4 epochs as the best tuned hyperparameters of the first GRU model.

Below is the confusion matrix for our best performing Simple GRU Model.

		Actual	
		0	1
Predicted	0	4	3
	1	5301	15242

Our model was much better at predicting positive values than negative values. This is because there were not many negative reviews in our validation set, as stated earlier.

E. Concat Pooling GRU Results

Our Concat Pooling GRU model did not consistently perform better as we increased the number of hidden layers and epochs. Below is the accuracy of our Concat Pooling GRU model after being trained and using validation data.

		Number of Epochs		
		2	3	4
Number of Hidden Layers	2	0.7282	0.3788	0.2587
	3	0.6607	0.3271	0.2594
	5	0.7418	0.3691	0.7418

As we can see, adding epochs or hidden layers can either improve or worsen our model's performance. Again, we will look at computational time for each set of hyperparameters.

		Number of Epochs		
		2	3	4
Number of Hidden Layers	2	11:26	17:18	21:24
	3	6:57	9:53	14:12
	5	11:59	18:13	24:13

Our computational time was also inconsistent as model complexity increased. For 3 hidden layers, our training time actually decreased. We will treat the 5 hidden layers and 2 epochs as our best performance of this model. As it had the same accuracy as 4 epochs but a much lower computation time. Below is the confusion matrix for this model:

	Actual	
	0	1

Predicted	0	0	3
	1	5302	15245

From this confusion matrix, we see that this model struggled to predict negative sentiment reviews too. This is again because our dataset had more positive reviews than negatives.

F. Deep Learning Model Conclusion

Our first GRU model worked better than the more complex Concat Pooling GRU model. This is probably because our dataset was too small for such a complex model. If the dataset was larger and we had access to more computation power, it could have performed better.

The first GRU model performed best at its most complex state but this also increased its computational time. There was not a large marginal increase in accuracy for the extra computation time though. This is again probably due to the fact that our dataset is not large enough for a deep learning model to be efficient.

The Concat Pooling GRU model that we trained was very inconsistent as we tuned hyper parameters. Computational time and accuracy could both increase and decrease as we add model complexity. This model is too complex for the dataset we are using.

In conclusion, while deep learning models have been a breakthrough for Natural Language Processing, it requires a very large dataset and large computational power, both of which we are lacking in this scenario. This means that other simpler models may perform as well or better than the deep learning models which we tested.

	$y = 1$ (+)	**634**	**14,611**

Initially, we already predicted that our models would find it difficult to predict if a review is truly negative, which means that we will expect errors in the $\widehat{y} = 1$ (+) column and the $y = 0$ (-) row because of significantly more positive than negative reviews in our dataset. The confusion matrix confirms this suspicion, because this model has difficulty predicting that sentences actually portray negative sentiments because of the limited negative words sample size. This is the model with the best accuracy so far – let's see if some other classifier can beat it!

REFERENCES

[1] Dimitrios Kotzias et. al, "From Group to Individual Labels using Deep Features" in *KDD*, 2015. https://archive.ics.uci.edu/ml/datasets/Sentiment+Labelled+Sentences

[2] M. Utagh, "IMDB Review Dataset" in *Kaggle*, 2018. https://www.kaggle.com/utathya/imdb-review-dataset

[3] S. Kostadinov, "Understanding GRU Networks" *Towards Data Science*, 2017. https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be

[4] S. Himanshu, "Sentiment Analysis - TorchText" *Medium*, 2018. https://medium.com/@sonicboom8/sentiment-analysis-torchtext-55fb57b1fab8

[5] J. hpanwar08, "Sentiment Analysis in torchtext" *Github*, 2018. https://github.com/hpanwar08/sentiment-analysis-torchtext/blob/master/Sentiment_Analysis_torchtext.ipynb https://machinelearningmastery.com/naive-bayes-classifier-scratch-python/