Classification of Yelp Reviews

Main Objective

- In this report, we will analyze the Yelp reviews dataset.
- The objective of this report is to find out the best model that takes in as input a review written by a Yelp user and finds whether the review is positive or negative.
- We will use a variety of hyperparameter modifications of Recurrent Neural Networks. We will also use LSTM and GRU neural networks and find out the best model for this dataset.

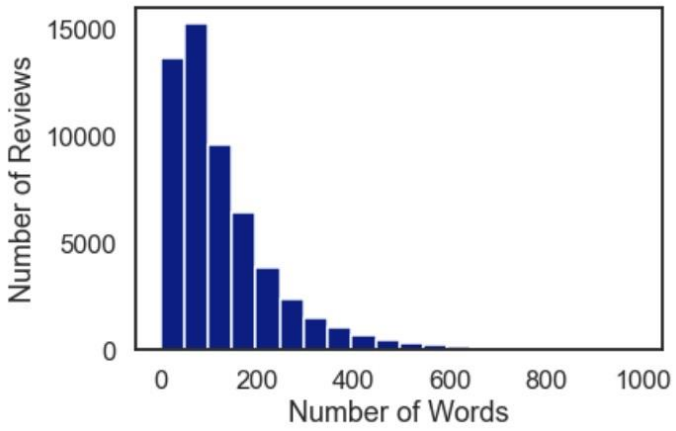
Brief Description of the Dataset

- This dataset has one feature and one target column.
- It has 560,000 training samples and 38,000 testing samples.
- The feature corresponding to each sample contains a review written by a Yelp user regarding a business, describing their experience with it in English. Its data type is tensorflow.string.
- The user also enters 1 to 4 stars describing their overall experience. This dataset converts 1 or 2 stars to 0 (negative experience) and 3 or 4 stars to 1 (positive experience).
- The target corresponding to each sample contains either a 1 or a 0 depending on whether the customer chose to describe the overall experience as positive or negative. Its data type is int64.

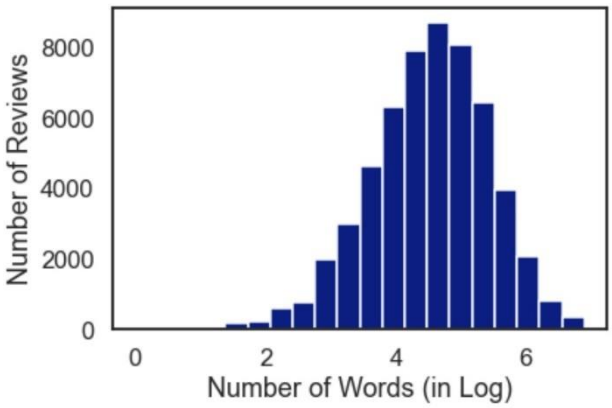
Data Cleaning and Feature Engineering

- The dataset is too large and has 560,000 training samples. We shall only use 10% of the training set, i.e., 56,000 samples to train our model. We extract these samples out of the larger training dataset after shuffling the dataset to ensure there are no systematic problems arising because of the training set not being uniform.
- We split all the training and testing reviews into two lists.

- Similarly, the review scores are split into two NumPy arrays.
- We subsequently convert the reviews into a dataset whose elements are slices of a tensor so we can encode the text into vectors.
- Now we use the Text Vectorization encoder with the maximum features set to 10,000 to encode our text. The encoder is adapted to the training reviews and then applied to both the training and the testing set. This encoder converts all the words and other special characters to numbers and creates an array of numbers out of a string of text.
- The encoder already pads our data by adding zeros to ensure all the vectors have the same length equal to that of the longest review, which in this case is 991 words.
- We plot the number of words in the reviews as a histogram to better understand how many words we should keep in our reviews while training the models.



- We find that the average review has 132.2 words with a standard deviation of 122 and has a skew value of 2.3.
- We expect this to closely resemble a lognormal distribution, so we use the np.log1p() method to log transform it and plot it again to find that it now does resemble a normal distribution.



- Based on these statistics, we choose to cap the number of words per review at 300 to ensure our models work fast while ensuring that we can train them accurately.

Deep Learning Models

In this section, we will use various deep learning models to predict whether a given review is net positive or negative:

1. **Simple RNN (Base Case)**: We start with the most straightforward recurrent neural network. This network has three layers. The details of this model are given below.
   i) The first layer takes each integer (input dimension 10,000) and embeds it in a 64-dimensional vector (embedding dimension).
   ii) The second layer is the simple RNN which has a hidden dimension of 5. The kernel initializers are set to normal random with a very small standard deviation of 0.001. The recurrent initializers are set to Identity and the activation is set to ReLU.
   iii) The third layer is dense and has an output of dimension 1 with the activation set to sigmoid. This layer will classify the review as 0 or 1.
   iv) The model has 640,000 embedding parameters, 350 parameters for the simple RNN, and 6 parameters for the dense layer. The total number of parameters is 640,356.
   v) We set the optimizer to RMSprop with a learning rate of 0.0001.
   vi) We use binary cross-entropy as the loss function and use accuracy as our metric to judge the performance of the model.
   vii) We set the batch size to 100 and train the model for 7 epochs.

   The performance of the model is shown below:

```
Epoch 1/7
560/560 [==============================] - 29s 51ms/step - loss: 0.4788 - accuracy: 0.7503 - val_loss: 0.2909 - val_accuracy:
0.8789
Epoch 2/7
560/560 [==============================] - 32s 56ms/step - loss: 0.2802 - accuracy: 0.8841 - val_loss: 0.2649 - val_accuracy:
0.8919
Epoch 3/7
560/560 [==============================] - 34s 60ms/step - loss: 0.2341 - accuracy: 0.9058 - val_loss: 0.2301 - val_accuracy:
0.9102
Epoch 4/7
560/560 [==============================] - 34s 60ms/step - loss: 0.2109 - accuracy: 0.9179 - val_loss: 0.2782 - val_accuracy:
0.8878
Epoch 5/7
560/560 [==============================] - 35s 62ms/step - loss: 0.1969 - accuracy: 0.9226 - val_loss: 0.2183 - val_accuracy:
0.9131
Epoch 6/7
560/560 [==============================] - 33s 58ms/step - loss: 0.1864 - accuracy: 0.9278 - val_loss: 0.2410 - val_accuracy:
0.9040
Epoch 7/7
560/560 [==============================] - 34s 60ms/step - loss: 0.1801 - accuracy: 0.9310 - val_loss: 0.2084 - val_accuracy:
0.9178
```

2. **Simple RNN (Increased Dimensions)**: We now make small modifications to our previous model by increasing the hidden dimension of the RNN to 10 from 5 and increasing the embedding dimension to 128 from 64. This increases the total number of parameters to 1,281,401. We keep everything else the same as in the previous model to see if increasing these dimensions will help us get a better classification model. We get the following results in this case:

```
Epoch 1/7
560/560 [==============================] - 42s 75ms/step - loss: 0.4570 - accuracy: 0.7755 - val_loss: 0.2623 - val_accuracy:
0.8938
Epoch 2/7
560/560 [==============================] - 41s 73ms/step - loss: 0.2604 - accuracy: 0.8947 - val_loss: 0.2598 - val_accuracy:
0.8947
Epoch 3/7
560/560 [==============================] - 42s 75ms/step - loss: 0.2196 - accuracy: 0.9129 - val_loss: 0.2239 - val_accuracy:
0.9162
Epoch 4/7
560/560 [==============================] - 48s 86ms/step - loss: 0.1999 - accuracy: 0.9233 - val_loss: 0.2101 - val_accuracy:
0.9188
Epoch 5/7
560/560 [==============================] - 43s 77ms/step - loss: 0.1860 - accuracy: 0.9289 - val_loss: 0.2026 - val_accuracy:
0.9201
Epoch 6/7
560/560 [==============================] - 43s 77ms/step - loss: 0.1763 - accuracy: 0.9322 - val_loss: 0.2204 - val_accuracy:
0.9155
Epoch 7/7
560/560 [==============================] - 43s 77ms/step - loss: 0.1696 - accuracy: 0.9347 - val_loss: 0.2170 - val_accuracy:
0.9179
```

We find that the training time increases by about 25%. The training accuracy after 7 epochs shows an increase of less than 0.5% and the validation accuracy is the same for three places of decimal. Thus, it appears that the additional hidden and embedding dimensions, while significantly increasing the number of parameters in our model, do manage to make the training model fit slightly better, the effect on the validation set is negligible and the two models perform roughly the same.

3. **Gated Recurrent Unit (GRU):** Instead of using a simple RNN for our second layer, we now use a GRU. We start with the number of hidden dimensions equal to 5 and the number of embedding dimensions set to 64. All other details are the same as in the previous two models. The performance of this model is significantly worse than the previous two models and is shown below:

```
Epoch 1/7
560/560 [==============================] - 37s 64ms/step - loss: 0.6927 - accuracy: 0.5264 - val_loss: 0.6922 - val_accuracy:
0.5280
Epoch 2/7
560/560 [==============================] - 39s 70ms/step - loss: 0.6917 - accuracy: 0.5291 - val_loss: 0.6910 - val_accuracy:
0.5280
Epoch 3/7
560/560 [==============================] - 41s 73ms/step - loss: 0.6901 - accuracy: 0.5292 - val_loss: 0.6893 - val_accuracy:
0.5284
Epoch 4/7
560/560 [==============================] - 42s 75ms/step - loss: 0.6885 - accuracy: 0.5293 - val_loss: 0.6881 - val_accuracy:
0.5286
```

```
Epoch 5/7
560/560 [==============================] - 43s 77ms/step - loss: 0.6876 - accuracy: 0.5294 - val_loss: 0.6874 - val_accuracy:
0.5288
Epoch 6/7
560/560 [==============================] - 45s 80ms/step - loss: 0.6870 - accuracy: 0.5294 - val_loss: 0.6871 - val_accuracy:
0.5289
Epoch 7/7
560/560 [==============================] - 46s 82ms/step - loss: 0.6865 - accuracy: 0.5294 - val_loss: 0.6868 - val_accuracy:
0.5290
```

The model has a very poor accuracy of only around 0.53 while also not offering any time benefits. Just to confirm that GRUs are not optimum for our particular case, we also try a GRU with the number of hidden dimensions set to 10 and the number of embedding dimensions set to 128. We find that there is no improvement in the accuracy of our model while the training time goes up significantly. The results for this case are shown below.

```
Epoch 1/7
560/560 [==============================] - 53s 94ms/step - loss: 0.6926 - accuracy: 0.5144 - val_loss: 0.6919 - val_accuracy:
0.5294
Epoch 2/7
560/560 [==============================] - 56s 100ms/step - loss: 0.6908 - accuracy: 0.5298 - val_loss: 0.6896 - val_accuracy:
0.5292
Epoch 3/7
560/560 [==============================] - 61s 110ms/step - loss: 0.6886 - accuracy: 0.5301 - val_loss: 0.6879 - val_accuracy:
0.5293
Epoch 4/7
560/560 [==============================] - 65s 116ms/step - loss: 0.6875 - accuracy: 0.5303 - val_loss: 0.6873 - val_accuracy:
0.5294
Epoch 5/7
560/560 [==============================] - 58s 104ms/step - loss: 0.6867 - accuracy: 0.5301 - val_loss: 0.6868 - val_accuracy:
0.5294
Epoch 6/7
560/560 [==============================] - 60s 106ms/step - loss: 0.6860 - accuracy: 0.5303 - val_loss: 0.6863 - val_accuracy:
0.5295
Epoch 7/7
560/560 [==============================] - 60s 107ms/step - loss: 0.6852 - accuracy: 0.5302 - val_loss: 0.6859 - val_accuracy:
0.5296
```

Based on these two models, we decide that GRUs are not fit for our purpose.

4. **LSTM:** Finally, we try an LSTM (Long-Short Term Memory) network. Once again we set the number of hidden dimensions to 5 and the number of embedding dimensions to 64. The performance metrics in this case are:

```
Epoch 1/7
560/560 [==============================] - 42s 74ms/step - loss: 317936192.0000 - accuracy: 0.5313 - val_loss: 46329012224.0000
- val_accuracy: 0.5328
Epoch 2/7
560/560 [==============================] - 42s 75ms/step - loss: 15802331136.0000 - accuracy: 0.5329 - val_loss: 16284268.0000
- val_accuracy: 0.5331
Epoch 3/7
560/560 [==============================] - 43s 76ms/step - loss: 154843296.0000 - accuracy: 0.5342 - val_loss: 3079774.7500 - v
al_accuracy: 0.5331
```

```
Epoch 4/7
560/560 [==============================] - 42s 75ms/step - loss: 16837.3145 - accuracy: 0.5354 - val_loss: 2845847.7500 - val_a
ccuracy: 0.5348
Epoch 5/7
560/560 [==============================] - 45s 81ms/step - loss: 160534592.0000 - accuracy: 0.5393 - val_loss: 3365095424.0000
- val_accuracy: 0.5403
Epoch 6/7
560/560 [==============================] - 47s 83ms/step - loss: 191507152.0000 - accuracy: 0.5394 - val_loss: 987396416.0000 -
val_accuracy: 0.5424
Epoch 7/7
560/560 [==============================] - 46s 82ms/step - loss: 4958822400.0000 - accuracy: 0.5425 - val_loss: 347620480.0000
- val_accuracy: 0.5430
```

We see that the accuracy is once again only marginally better than 0.5 while the training time is significantly greater than that for simple RNNs. Thus, both GRUs and RNNs are not well suited for our task.

Best Model

Clearly, the simple RNN outperforms the GRUs and the LSTMs, both in terms of accuracy as well time taken for training. Furthermore, the simple RNN with 10 hidden dimensions and 128 embedding dimensions takes a significantly greater time than the one with 5 hidden dimensions and 64 embedding dimensions to train while not outperforming in terms of accuracy. Thus, the model best suited for our needs is the simple RNN model with 5 hidden and 128 embedding dimensions. To ensure that we are using the best possible version of this model, we change some of the hyperparameters and retrain our model and see if there is any noticeable difference in performance. To do this, we try another simple RNN model by changing two hyperparameters. Instead of setting the maximum number features to 10,000, we set it to 20,000 and instead of taking the first 300 words for training and testing our models, we now take the first 400 words. The performance metrics in this case are shown below.

```
Epoch 1/7
560/560 [==============================] - 44s 78ms/step - loss: 0.5326 - accuracy: 0.7225 - val_loss: 0.3668 - val_accuracy:
0.8411
Epoch 2/7
560/560 [==============================] - 47s 84ms/step - loss: 0.3078 - accuracy: 0.8720 - val_loss: 0.2385 - val_accuracy:
0.9030
Epoch 3/7
560/560 [==============================] - 50s 89ms/step - loss: 0.2339 - accuracy: 0.9048 - val_loss: 0.2312 - val_accuracy:
0.9018
Epoch 4/7
560/560 [==============================] - 55s 99ms/step - loss: 0.2063 - accuracy: 0.9172 - val_loss: 0.2174 - val_accuracy:
0.9116
Epoch 5/7
560/560 [==============================] - 51s 92ms/step - loss: 0.1903 - accuracy: 0.9244 - val_loss: 0.2285 - val_accuracy:
0.9093
Epoch 6/7
560/560 [==============================] - 48s 85ms/step - loss: 0.1792 - accuracy: 0.9303 - val_loss: 0.2104 - val_accuracy:
0.9186
Epoch 7/7
560/560 [==============================] - 48s 86ms/step - loss: 0.1701 - accuracy: 0.9342 - val_loss: 0.2501 - val_accuracy:
0.9078
```

We find that there is no improvement over our simplest RNN model and the validation accuracy oscillates close to 0.91. This is the same result we got for our baseline simple RNN which also took considerably less time to train.

Finally, we set the maximum number of features to 5,000 and consider the first 200 words for our model. The performance metrics for this case are shown below.

```
Epoch 1/7
560/560 [==============================] - 22s 39ms/step - loss: 0.4746 - accuracy: 0.7607 - val_loss: 0.3187 - val_accuracy:
0.8631
Epoch 2/7
560/560 [==============================] - 21s 38ms/step - loss: 0.3035 - accuracy: 0.8699 - val_loss: 0.2643 - val_accuracy:
0.8873
Epoch 3/7
560/560 [==============================] - 22s 40ms/step - loss: 0.2570 - accuracy: 0.8945 - val_loss: 0.2746 - val_accuracy:
0.8854
Epoch 4/7
560/560 [==============================] - 26s 46ms/step - loss: 0.2369 - accuracy: 0.9050 - val_loss: 0.2435 - val_accuracy:
0.8994
Epoch 5/7
560/560 [==============================] - 24s 43ms/step - loss: 0.2251 - accuracy: 0.9101 - val_loss: 0.2357 - val_accuracy:
0.9042
Epoch 6/7
560/560 [==============================] - 25s 45ms/step - loss: 0.2169 - accuracy: 0.9139 - val_loss: 0.2288 - val_accuracy:
0.9091
Epoch 7/7
560/560 [==============================] - 24s 42ms/step - loss: 0.2099 - accuracy: 0.9179 - val_loss: 0.2269 - val_accuracy:
0.9074
```

The validation accuracy is down by about 1%, which is significant since this is a difference of 1% out of a maximum possible 10%. Moreover, the time to train hasn't changed considerably. Thus, we conclude that our first model, the base simple RNN model is the best one for our needs.

Key Findings

Based on the 7 models we tried (using different neural network structures and different hyperparameters), we found the best model and gained useful knowledge about other models as well.

- We found that the two more complicated neural networks, i.e., GRU and LSTM, have a worse performance than the simple RNN.
- We found the maximum number of hidden and embedding dimensions that we should consider to both a fast, accurate model are 5 and 64 respectively. Increasing the dimensions and thus the complexity of our neural network only overfits our data while having no effect on the validation accuracy.

- We found the maximum number of features that we should consider in our encoding to ensure our model has the best predictability. We found that 10,000 maximum features give a better performance than both 20,000 as well as 5,000. Thus, we believe the optimum answer should be close to 10,000. Moreover, since the change in the accuracy of our model for these three hyperparameters was not huge, it's good enough that we use 10,000 features for our model instead of focusing on optimizing this number.
- We found that the number of words that we consider in our training and testing models influences the accuracy of our model. While 200 words produce a slightly less accurate model than 300 words, 400 words don't change the accuracy of the model. Thus, the optimum number of words to consider for our RNN model is close to 300 and small changes around this number will not have a huge effect on the outcome of our model.
- Finally, we found that we were able to produce simple RNN models with high accuracy of more than 90% in predicting the outcome score for the reviews. Thus, we have successfully found a good, easy to implement model four our dataset.

Next Steps

We started with a huge training dataset, with 560,000 samples. Since the number was too large and would have taken a very long time for our models to train on this dataset, we chose to only work with 10% of this dataset. With faster computing resources, a complete analysis of this dataset can be performed, and this helps produce better models and will also allow us to be more confident in our models. Similarly, we could not test our model on the entire validation set and we used only a subset. Faster computing would also allow us to test our model on the entire testing dataset. This would ensure that we can create better models which allow us to correctly predict the sentiment of a vast array of reviews instead of possibly specializing on a sub-class.

There were a large number of hyperparameters that went into our models and we couldn't possibly check the performance as we change all of them. A longer version of this project could vary many of these hyperparameters such as the learning rate for gradient descent, changing the activation functions, changing the kernel initializers etc., and observe the corresponding effect on our models. We could also change the loss function and the optimizer for the gradient

descent to Adam or Adagrad and record any difference in our model. Overall, there are a large number of possibilities of further steps that can be undertaken for a more in depth study.

We attach the python code used to prepare this report as an appendix. The code is not necessary to follow the report but can be used to verify any statements made in the report.