Abstract

In our experiments we wanted to analyze the structure and tradeoffs for building NLP neural nets. We want to use this data to build chat bots in the future. We analyzed Reuters data set, which contains 127600 sample articles, all with a label category of Sports, World, Business, and Sci/Tech. Our best model performed with an accuracy of 88.22% on the test data set and used two bidirectional LSTM layers, a dense layer, dropout, and an output layer. This model used a vocab size of 2000 words instead of the default 1000 words for the other models.

Introduction

This week we researched natural language processing (nlp) models. We were given the task of researching how to develop a conversational agent or chatbot to assist customer support representatives. To work on training an NLP model we used Reuters new data with labels of each text as World, Sports, Business, and Sci/Tech. We ran several experiments to test different architectures to use for an nlp model. Some of the experiments ran were changing number of layers, methods like LSTM and Simple RNN, regularization, and text processing.

Literature Review

An interesting approach was used by a team of Scientists, Zhixuan Zhou, Huankang Guan, Meghana Moorthy Bhat, Justin Hsu, to build a fake news detection model. Fake news is a big problem in the world today as fake news can quickly spread using the internet and social media. The team was able to show that fact tampering can be effective on fake news classifier models as well as under-written real news. A solution to this problem is using fact checking in conjunction with the linguistic model. They propose crowdsourcing a knowledge graph as a straw man solution to collecting timely facts about news events. This approach adds another layer for the model to make sure it can accurately classify false news. More research on this will be done in the coming years as companies, people, and governments look to curb the spread of false news. The link to the paper is in the appendix.

Review research design and modeling methods

This AG news collection is constructed by choosing the 4 largest categories from World, Sports, Business, and Sci/Tech. Each of these classes contains 30000 training samples and 1900 testing samples for a total number of 120000 training samples and 7600 testing samples. In the table below is an example of the first five data point containing the text in the description column and the news classification category in the label column.

Table 1: News Data Examples

	description	label
0	AMD #39;s new dual-core Opteron chip is designed mainly for corporate computing applications, including databases, Web services, and financial transactions.	3 (Sci/Tech)
1	Reuters - Major League Baseball\Monday announced a decision on the appeal filed by Chicago Cubs\pitcher Kerry Wood regarding a suspension stemming from an\incident earlier this season.	1 (Sports)
2	President Bush #39;s quot;revenue-neutral quot; tax reform needs losers to balance its winners, and people claiming the federal deduction for state and local taxes may be in administration planners #39; sights, news reports say.	2 (Business)
3	Britain will run out of leading scientists unless science education is improved, says Professor Colin Pillinger.	3 (Sci/Tech)
4	London, England (Sports Network) - England midfielder Steven Gerrard injured his groin late in Thursday #39;s training session, but is hopeful he will be ready for Saturday #39;s World Cup qualifier against Austria.	1 (Sports)
5	TOKYO - Sony Corp. is banking on the \$3 billion deal to acquire Hollywood studio Metro-Goldwyn-Mayer Inc...	0 (World)

For analyzing the data, we used natural language processing techniques and models to build a multi-classification model for classifying the text into one of four categories: 1. Sci/Tech 2. Sports 3. Business 4. World. In the figure below is an example model of nlp. The text input is first transformed in In the figure below is an example model of nlp. The text input is first transformed into a text vectorization, then the vector is converted into an embedding to capture word associations. The bidirectional layer in the figure is used so that the model can learn the context of a word based on the words left and right of the word. It is then passed to a dense layer then softmax transformation is performed for the classification model.

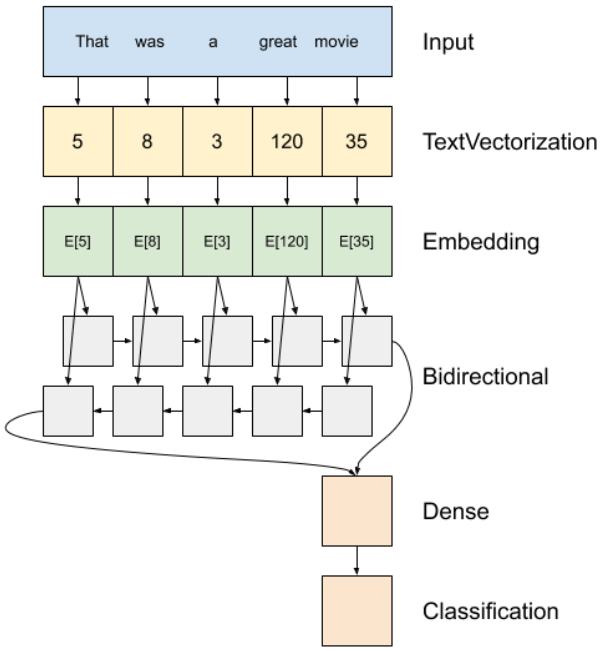

Figure 1: NLP Architecture Model

Implementation and Programming

These models were implemented using the python programming language and several libraries. The main libraries used were pandas for data cleaning and managing, keras tensorflow was used to build the

neural networks, sklearn for scaling and confusion matrix, numpy to handle arrays, and matplotlib to build the plots. The sequential class was used to build the neural networks with different activation functions, number of layers, and number of nodes. LSTM, Bidirectional, SimpleRNN, and Conv1d were all methods we used to build hidden layers for nlp model. For regularization techniques we used tensorflow's EarlyStopping and dropout methods to prevent overfitting. The compile method was used with a loss function of SparseCategorialCrossentropy because we did not one hot encode the data. The evaluate method was used to test the accuracy of the model after being fitted on the test data. The development environment was done in jupyter noteboks and the code can be found attached to this report in pdf format.

We approached the problem by conducting4 experiments. The first experiment we performed different exploratory data analysis procedures to see how it would affect the models. The different eda procedures were trying different vocab sizes, removing stop words, and changing output sequence length. The second experiment we used a deep neural network with 2 layers, a bidirectional simple rnn layer and a dense layer, with regularization of dropout. For the third experiment we built a model with 2 layers, using a LSTM layer and dense layer, with regularization. For the fourth experiment we built a CNN with 1 convolution and max pooling layers using Conv1D and regularization.

Data Preparation

Once downloading the data we explored and prepared the data before building the neural net model. Using the TextVectorization method for keras we vecotorized and encoded the data. From analyzing the data we learned that there were 95976 different words in the corpus. Also, there were a total of 3909695 words in all 127600 articles, with each article between 3 and 173 tokens in it. In the figure below we can see the tokens per document graphed out and see that most articles contain around 30 tokens per document.

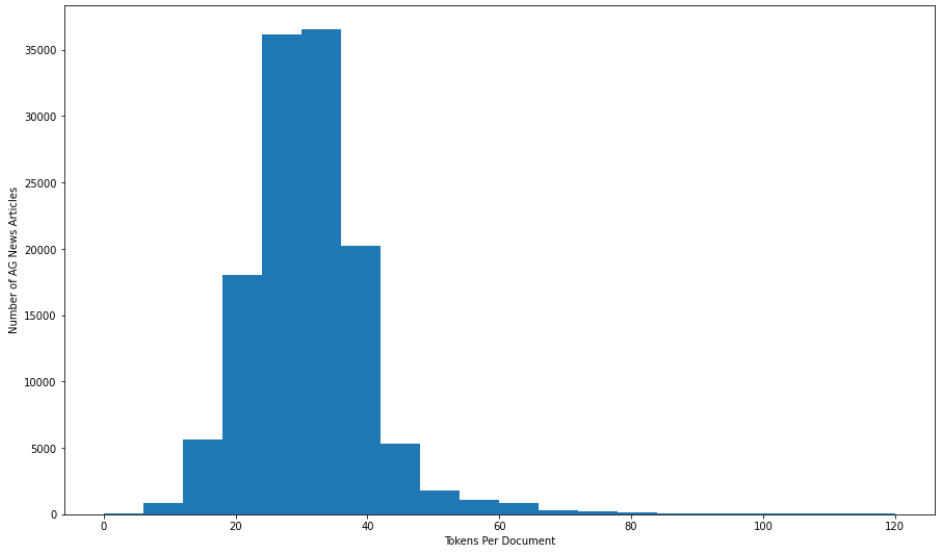

Figure 2: Tokens Per Document

After analyzing the data, we built encoder for the top 1000 most common words in the corpus, to speed up processing. The top common words were as listed: 'the', 'a', 'to', 'of', 'in', 'and', 'on', 'for', 'that', '39s', 'with', 'its', 'as', 'at', 'is', 'said', 'by', 'it'. From this we see the top words in the vocabulary list don't have much predictive power for our categories. We later experimented with removing the stop words from

the vocab list to make more space for words with more predictive power. From the figure below we can see that the top 1000 most common words usually only leave about 30-40% of the words unknown in an article. We also used 0 padding to pad by default up to the longest article of 128 words and 1 was encoded in the vector for words not within the vocab list. When encoding the TextVectorization method also converts all spelling to lowercase for all of the words and strips punctuation. This is so the words "The" and "the" don't get counted as two different words. Our dataset was split into a split of 114000 samples for training, 6000 validation samples, and 7600 samples for testing.

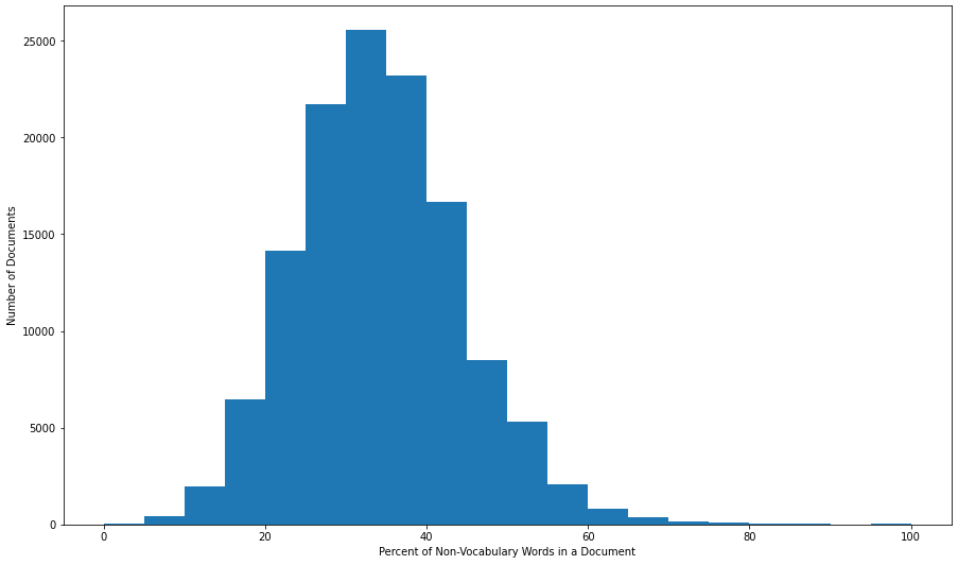

Figure 3: Percent of Non-Vocabulary Words in Document

Results

In this section we will go over the results of the neural net models. We reviewed RNNs, then LSTMs, and 1D CNNs, then both with regularization like dropout and early stopping. Some of the constants between models is an optimizer of adam, loss function of SparseCategoricalCrossentropy, 10 epochs, and batch size of 64.

Experiment 1

For our first experiment we performed different EDA techniques to see the affect the data had on the models. The different techniques we applied were trying three different vocab sizes, removing stop words, and changing the output sequence length.

For trying out different vocab lists we experimented using the default size of 1000 words, a size of 500 words, and a size of 2000 words. All models used the same architecture as seen below with two biderictional LSTM layers of 64 and 32 nodes, a dense layer of 63 with an activation layer of relu, then a dropout layer of 50%, followed by an output layer with a softmax activation function for classification.

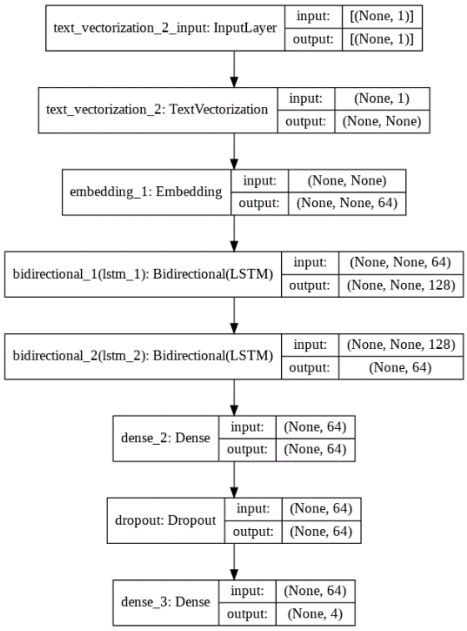

Figure 4: Architecture for model to experiment out different vocab sizes

From the table below we can analyze the results from this technique. We can see that accuracies generally improved as the number of vocab words increased. Also, surprisingly the processing time of the model did not take much longer for longer vocab lists suggesting that the increase in vocab size might not have as big of impact on processing time as originally thought by the team.

Table 2: Training, Validation, and Testing Accuracy for Different Vocab Sizes

Vocab Size	Training Accuracy %	Validation Accuracy %	Test Accuracy %	Time Seconds
500 Words	83.09	82.92	83.05	2570
1000 Words	89.23	86.61	85.59	2640
2000 Words	89.02	87.34	88.22	2800

Next we removed the stop words from the text. We removed the stop words using the nltk package in python and was able to remove common words like “the” and “a” from the text in the data. We used same architecture as the previous experiment and compared the result of the two. We compared the two both using a vocab size of 1000 words. After removing stop words the top words were as listed: '39', 'said’, ‘new', 'us', 'reuters', 'quot', 'ap', 'two', 'gt', 'lt', 'first', 'monday', 'wednesday', 'company',tuesday', 'world', 'thursday'. From the results we were able to see the model with the stop words removed performed better on test set but not as well on the validation set as the model with the stop words in the vocab list. Overall it seems like a good step to remove stop words since they hold little predictive power for our categories and free up some space for a few more words in the vocab list.

Stop Words Removed	Training Accuracy %	Validation Accuracy %	Test Accuracy %	Time Seconds
No	89.23	86.61	85.59	2640
Yes	88.64	85.36	86.52	2250

For the third EDA technique we changed the sequence length of the encoder. By default we had it set to the length of the longest document but for this experiment we set it the length to 60, which from Figure 2 can be seen to fully cover the vast majority of documents. If a document is longer than 60 tokens then the rest of it will be cutoff from the input and if a document is shorter than the length of 60 then 0's will be added to the end. From the results we see that the accuracies for shortening the output length were slightly less but it was also slightly faster to reduce output length.

Output Length	Training Accuracy %	Validation Accuracy %	Test Accuracy %	Time Seconds
128	89.23	86.61	85.59	2640
60	86.01	85.89	85.57	2240

Experiment 2

For our second Experiment we build a model using a Bidirectional SimpleRNN layer with 64 nodes for one direction, and a dense layer of 64 nodes with relu, then an output layer using sofmax. A summary of the model can be seen the figure below.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
text_vectorization_17 (TextV (None, 60)                0
_________________________________________________________________
embedding_7 (Embedding)      (None, 60, 64)            64000
_________________________________________________________________
bidirectional_12 (Bidirectio (None, 128)               16512
_________________________________________________________________
dense_11 (Dense)             (None, 64)                8256
_________________________________________________________________
dense_12 (Dense)             (None, 4)                 260
=================================================================
Total params: 89,028
Trainable params: 89,028
Non-trainable params: 0
```

Figure 5: Summary of Simple RNN model

In the table below can be seen the accuracies and time of the Simple RNN model. As can be seen the processing times for SimpleRNN are much faster than using LSTM, however the accuracy on the

validation and test set also went down by about 5% when compared to using a LSTM model which incorporates long term memory unlike a SimpleRNN model.

Training Accuracy %	Validation Accuracy %	Test Accuracy %	Time Seconds
88.44	81.67	82.79	670

Experiment 3

For the next experiment we analyzed using an LSTM model. In previous experiments we used bidirectional LSTM layers so for this experiment we tried using a single unilateral LSTM layer of 64 nodes, a dense layer of 64 nodes with relu, followed by an output layer with softmax.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
text_vectorization_17 (TextV (None, 60)                0
_________________________________________________________________
embedding_11 (Embedding)     (None, 60, 64)            64000
_________________________________________________________________
lstm_17 (LSTM)               (None, 64)                33024
_________________________________________________________________
dense_19 (Dense)             (None, 64)                4160
_________________________________________________________________
dropout_9 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_20 (Dense)             (None, 4)                 260
=================================================================
Total params: 101,444
Trainable params: 101,444
Non-trainable params: 0
```

In the table below we can see that the unidirectional LSTM layer model performed about as equally as well with only a slight hit to the validation accuracy, but a reduced computation time of nearly 40%. A unilateral approach might be a good idea to use instead of bidirectional if little is gained in accuracy from bidirectional, as the unidirectional approach is much faster.

Directional layer	Training Accuracy %	Validation Accuracy %	Test Accuracy %	Time Seconds
Unidirectional layer	87.03	85.68	85.54	980
Bidirectional layer	86.76	86.47	85.56	1600

Experiment 4

For the last experiment we tested a model using a 1DConv layer. This model used a 1D convolutional layer of 64 features, and kernels size of 3, then a max pooling 1d layer, followed by a flattening layer, then a dense layer of 64 nodes, then regularization of dropout with 50% dropout.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
text_vectorization_17 (TextV (None, 60)                0
_________________________________________________________________
embedding_41 (Embedding)     (None, 60, 64)            64000
_________________________________________________________________
conv1d_27 (Conv1D)           (None, 58, 64)            12352
_________________________________________________________________
max_pooling1d_23 (MaxPooling (None, 29, 64)            0
_________________________________________________________________
flatten_5 (Flatten)          (None, 1856)              0
_________________________________________________________________
dense_75 (Dense)             (None, 64)                118848
_________________________________________________________________
dropout_37 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_76 (Dense)             (None, 4)                 260
=================================================================
Total params: 195,460
Trainable params: 195,460
Non-trainable params: 0
```

From the experiment we see that a 1D convolutional layer performed relatively well on the datasets and processing time was exceptionally quick. This could be an avenue to take if we want to significantly cut on training time while not sacrificing too much accuracy.

Training Accuracy %	Validation Accuracy %	Test Accuracy %	Time Seconds
87.06	85.73	85.62	150

Conclusion

In the table below we analyze the results from all experiments. In conclusion we can see that the model containing a vocab size of 2000 words performed the best. However, this model also took the longest to process. Vocab size and processing times will have to be trade offs that the team analyzes more when building a chatbot.

Model	Training Accuracy %	Validation Accuracy %	Test Accuracy %	Time Seconds
1DConv	87.06	85.73	85.62	150
Unidirectional layer	87.03	85.68	85.54	980

Bidirectional layer	86.76	86.47	85.56	1600
SimpleRNN	88.44	81.67	82.79	670
128 Output length	89.23	86.61	85.59	2640
60 Output length	86.01	85.89	85.57	2240
No Stop Words	89.23	86.61	85.59	2640
Contains Stop Words	88.64	85.36	86.52	2250
500 Words	83.09	82.92	83.05	2570
1000 Words	89.23	86.61	85.59	2640
2000 Words	89.02	87.34	88.22	2800

In conclusion the team was able to gain some valuable insights in building neural net models for natural language processing. We learned from the experiments that vocab size is very impactful to the model accuracy but increases processing time as the size grows. For the output length we saw accuracy dip only a little from decreasing output length but only a slight decrease in processing time. Also removing the stop words contributed slightly to improving the accuracy of the model. These were valuable lessons in how to transform the data before inputting it into the model. We also compared LSTM, SimpleRNN, and 1ConvD, with LSTM performing the best. To save processing time we could use a unidirectional LSTM layers or if decide we can sacrifice some accuracy for a huge gain in computational processing time then we could use a 1ConvD model. Our best model performed with an accuracy of 88.22% on the test data set and used two bidirectional LSTM layers, a dense layer, dropout, and an output layer. This model used a vocab size of 2000 words instead of the default 1000 words for the other models.

From this experiment we will begin to develop a chatbot to assist customer support representatives. This will be a substantial development as we will have to gather much data and define the problem, we want the chatbot to solve, but I am confident that with the knowledge gained from these experiments that we can build a quality chatbot, given we can get quality to train the model on. We will host the model in the cloud, allowing for a serverless approach if we use one of their services and enable predictions to be sent to the model using streaming the needed data to the model for real time

engagement between the chatbot and representatives. This project will help the business by empowering customer representatives to engage with customers better.

Appendix

Fake News Detection via NLP is Vulnerable to Adversarial Attacks:
https://arxiv.org/ftp/arxiv/papers/1901/1901.09657.pdf