Machine Learning Project Report: Sentiment Analysis

CECS 456 Spring 2021

Meng Moua, Devin Suy

https://github.com/mmoua89/ML-ChatApp

Goal

The objective of this project is to train two machine learning models to approach the task of sentiment classification of English text. To solve this binary classification problem, two supervised learning methods we employed were to train variations of Naive Bayesian and Recurrent Neural Network classifier models. These models are trained on sentiment labeled movie review text with the use of Python libraries nltk, scikit-learn, tensorflow and keras.

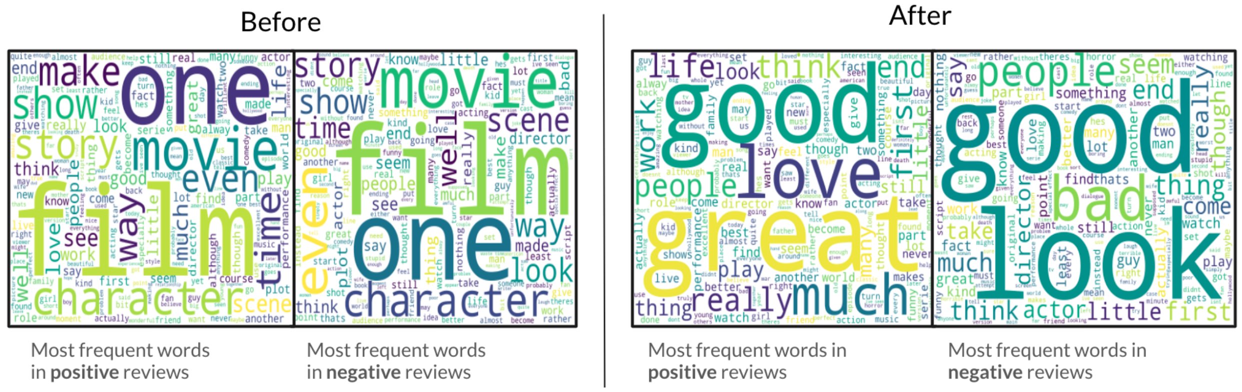
After training and evaluation, these models serve the backend predictions for our integrated data application. Given any user inputted English text of variable length, the trained models will be able to process the message and generate a corresponding sentiment classification label for the text.

Dataset

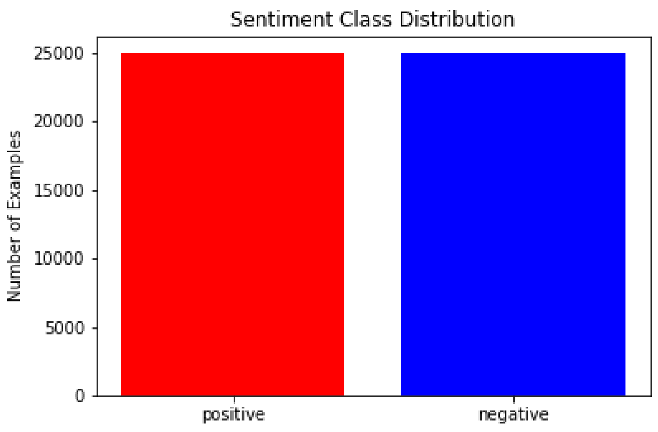
The dataset containing the sentiment labeled movie review text that we used to train our models on is sourced from the Stanford Large Movie Review Dataset (https://ai.stanford.edu/~amaas/data/sentiment/). The dataset itself contains 50,000 movie reviews scraped from the Internet Movie Database (IMDb) each with a corresponding sentiment label of either positive or negative.

Data Exploratory Analysis

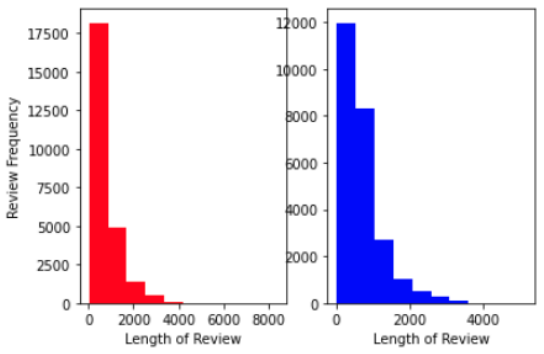
We divided the movie reviews into two sets based on their sentiment label and generated word clouds for each. The size of the word corresponds to the frequency of the word appearing among the reviews, with the largest words being the most common.


Before
After
Most frequent words in **positive** reviews
Most frequent words in **negative** reviews
Most frequent words in **positive** reviews
Most frequent words in **negative** reviews

This process was repeated before and after the removal of domain specific words to better visualize the words among these reviews that the models may learn to associate.



```
Positive Reviews:
  Longest review length: 8356 characters
  Shortest review length: 31 characters
  Avg review length: 752 characters

Negative Reviews:
  Longest review length: 5127 characters
  Shortest review length: 17 characters
  Avg review length: 726 characters
```

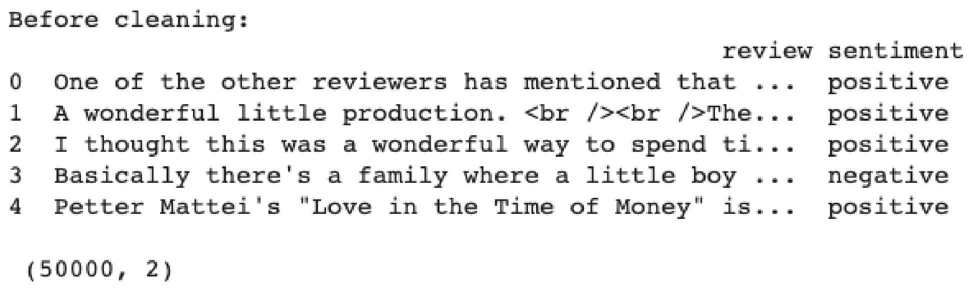


The dataset itself was shown to contain an exactly equal distribution of reviews among the two sentiment classes. A histogram plot also reveals that the vast majority of reviews in the dataset lie between 0 and 2000 characters for both classes.

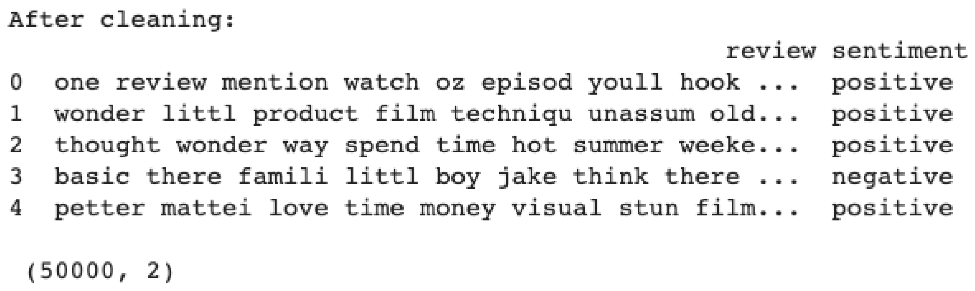
Data Preprocessing

The 50,000 dataset reviews contained various characters used in HTML tags such as special characters, numbers, symbols, and the most common words used in English. We don’t want to process these data because we want to avoid errors and duplicate data that might lead to poor results.

We use preset library packages in python like ToktokTokenizer from NLTK, Beautiful Soup, and Corpus to trim those unwanted data out. And here is an example of before and after the result.



```
Before cleaning:
                                                review sentiment
0  One of the other reviewers has mentioned that ...  positive
1  A wonderful little production. <br /><br />The...  positive
2  I thought this was a wonderful way to spend ti...  positive
3  Basically there's a family where a little boy ...  negative
4  Petter Mattei's "Love in the Time of Money" is...  positive

 (50000, 2)
```



```
After cleaning:
                                                review sentiment
0  one review mention watch oz episod youll hook ...  positive
1  wonder littl product film techniqu unassum old...  positive
2  thought wonder way spend time hot summer weeke...  positive
3  basic there famili littl boy jake think there ...  negative
4  petter mattei love time money visual stun film...  positive

 (50000, 2)
```

By looking at the above examples, we can clearly see that those tags “<br /><br />” are removed from line 1, and you may notice that the common words have been trimmed out and all the similar words will convert to its original root word.

For example, “family” or “families” -> “famili”

After the cleaning process, we will have the important keywords left contained in the bag. The size of the dataset remained the same at (50000, 2), which we then use sklearn split to randomly partition our data into 80% training and 20% testing.

Experimentation: Naive Bayesian

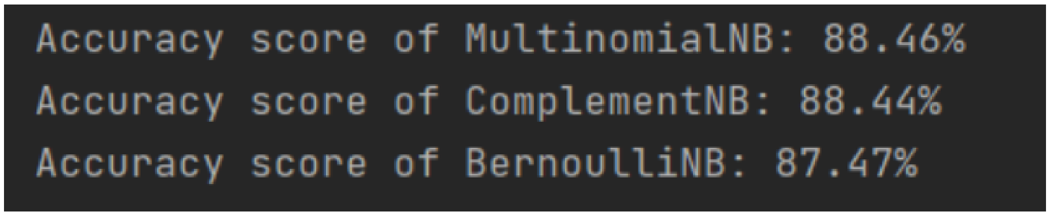
This is a learning model that we will use to classify and calculate the probability of how often a word occurs throughout the training process. In Naive Bayes, there are 5 different types including GaussianNB, CategoricalNB, BernoulliNB, MultinomialNB, and ComplementNB. I choose MultinomialNB, BernoulliNB, and Complements because they have accuracy scores higher than the others. Here is the basic algorithm that each type is used to train the dataset.

$$\underset{\text{Posterior Probability}}{\underset{\downarrow}{P(c \mid x)}} = \frac{\overset{\text{Likelihood}}{\overset{\nwarrow}{P(x \mid c)}}\,\overset{\text{Class Prior Probability}}{\overset{\nearrow}{P(c)}}}{\underset{\text{Predictor Prior Probability}}{\underset{\searrow}{P(x)}}}$$

$$P(c \mid \mathrm{X}) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

Posterior = likelihood * proposition/evidence

In MultinomialNB, this is the algorithm that we use to train our data because we have many reviews and each review is considered as a document. And this type is exactly what we need to use, and the accuracy score result came out higher than ComplementNB and BernoulliNB. In ComplementNB, this type is good to use when there is an imbalance of datasets that occurs throughout the process and this will help to match the data and bind them together to get the result. The last one is the BernoulliNB, this algorithm is the best use in counting the word by using only 0 and 1 (binary) to represent each word and then measure how often it occurred. For example, 0 means not occurred, and 1 means occurred. As a result of these 3 types, here are the scores that we got after the training process completed.



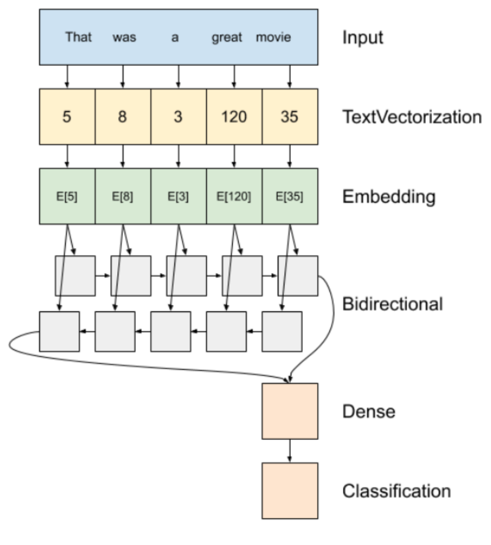
```
Accuracy score of MultinomialNB: 88.46%
Accuracy score of ComplementNB: 88.44%
Accuracy score of BernoulliNB: 87.47%
```

In our testing the MultinomialNB variant consistently outperformed the others for our task and dataset. After tuning the data cleaning to allow negation words such as “not”, “nor”, “never”, and “no”, below are the final results achieved for MultinomialNB.

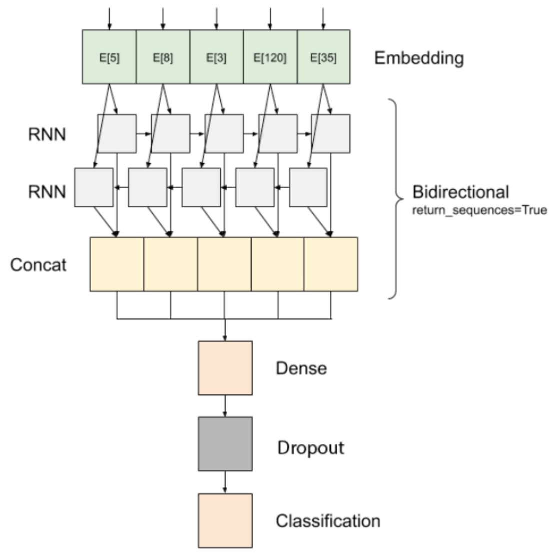
```
                    precision    recall  f1-score   support

Negative Sentiment       0.88      0.89      0.89      4900
Positive Sentiment       0.90      0.88      0.89      5100

          accuracy                           0.89     10000
         macro avg       0.89      0.89      0.89     10000
      weighted avg       0.89      0.89      0.89     10000
```

Experimentation: Recurrent Network

The other approach we used for creating a classification model is to use a recurrent neural network. Our intuition behind this decision is that in the context of natural language processing each movie review actually contains sequential data. Take for example a movie review, in which the first few words in the review may describe a subject, where the subsequent words then express something about that subject. Together, this sequential information can provide more context and be used to determine the overall sentiment of review text. Below is an overview of the recurrent network architectures we built and tested using keras's sequential model with TextVectorization, Embedding and Long short-term memory (LSTM) RNN layers.



"Single" LSTM architecture



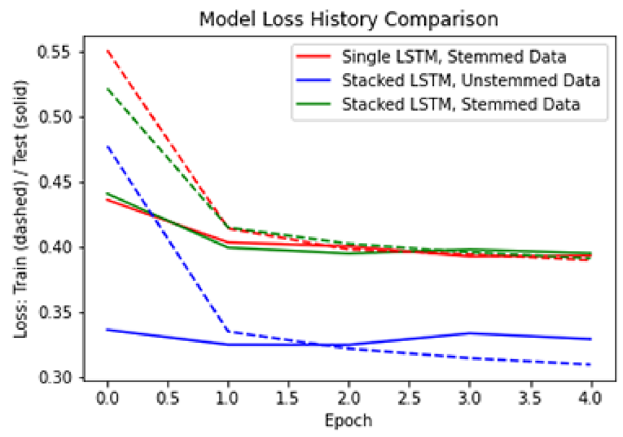
"Stacked" LSTM architecture

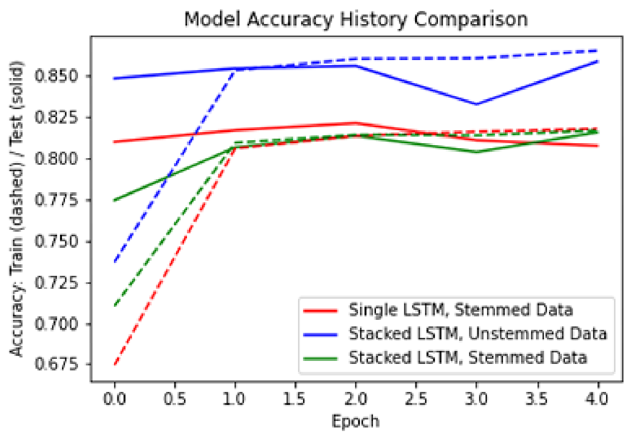
Rather than vectorize the input text first, our reasoning was that we can create an end to end model that is easier to deploy into our data application by including the text preprocessing as part of the model’s input. As shown above, the raw input text first passes through a TextVectorization layer, which is simply constructed with a vocabulary size parameter and serves as an encoder. We then pass all of our text data to this encoder in order to fit it to our dataset. Next these encoded word vocabulary indices move to an embedding layer, which simply maps each distinct word to a trainable vector, allowing our model to learn associations for each word. After this preprocessing, the raw input text has been transformed into a sequence of vectors, one for each word.

This sequence of vectors is what the LSTM layer receives as input. We opted for a bidirectional LSTM to amplify signal from the start of the sentence to still be able to affect the output. This is also possible since our application will not be streaming predictions, the entire input text is available at each time, allowing us to be able to propagate the input forwards and backwards through this LSTM layer to this advantage.

The stacked architecture (right) is very similar to the single architecture (left) except we introduce a second LSTM layer stacked above the other. Setting the *return_sequences=True* flag in keras translates to the top layer returning the full sequence for all of its states, instead of just the final one, and this is in turn fed directly as input to the second LSTM layer below. We also add a dropout layer after the fully connected layer before classification to introduce regularization. In this project, we tested a few different variations of these architectures, including the effect of tuning the size of the learned vocabulary in the encoding and embedding layers.

Below is the first of these, in which we trained and compared the single and stacked model architecture on the data processed as described above. Next is the variant in which we repeat the same data cleaning procedure, but instead avoid stemming each word to it's root "Stacked LSTM, Unstemmed Data". This model is identical to "Stacked LSTM Stemmed Data" where only this last step of data cleaning is different. This change was observed to consistently improve results, achieving minimum test loss at 0.329, and maximum test accuracy at 85.87% outperforming the others.

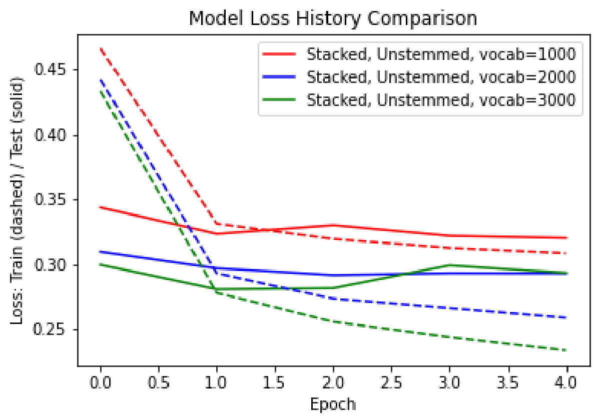


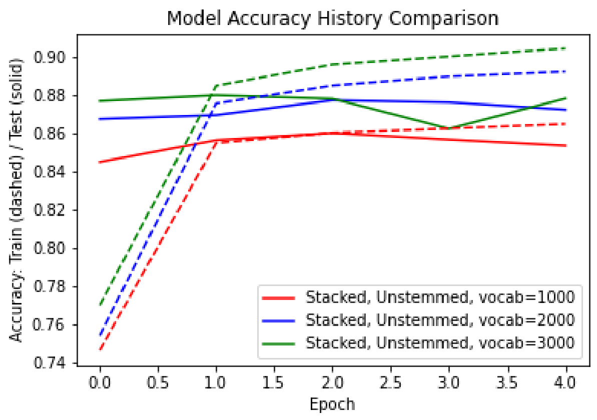


Additional training details is the use of keras ModelCheckpoint and EarlyStopping callbacks, in which the training set for 10 epochs was terminated at 4 epochs.

Loss Function	Binary cross entropy
Activation	RELU on final dense before classification, LSTM: sigmoid, tanh
Optimizer	Adam
Learning Rate	0.0001

Further tuning the "Stacked LSTM, Unstemmed Data" model's conditions, we then repeated the training process on variants in which we increased the vocabulary size parameter of the input encoder and embedding as shown below.





We are able to observe an increase in model performance under these conditions with increasing vocabulary size, as the model is allowed more words to store representations and therefore associations for. Though the loss is not consistent for the 3000 variant, it achieves the highest test accuracy at 87.81%.

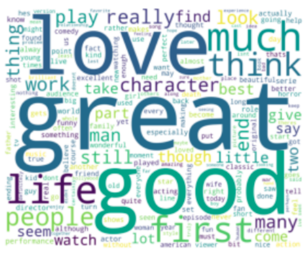
```
                    precision    recall  f1-score   support

Negative Sentiment       0.86      0.90      0.88      4900
Positive Sentiment       0.90      0.86      0.88      5100

          accuracy                           0.88     10000
         macro avg       0.88      0.88      0.88     10000
      weighted avg       0.88      0.88      0.88     10000
```

Analysis

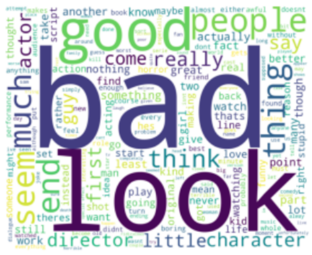
Similar to before, we generated word clouds based on frequency the word appears in the reviews, this time we divide them into the set of reviews that were correctly classified and those that were misclassified.

Correctly Classified

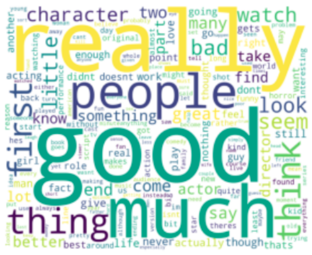
Incorrectly Classified



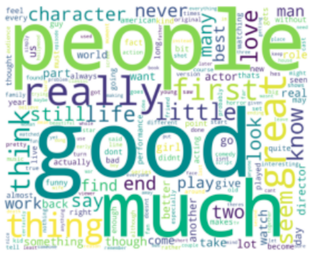
Most frequent words for **correctly** classified **positive** reviews



Most frequent words for **correctly** classified **negative** reviews

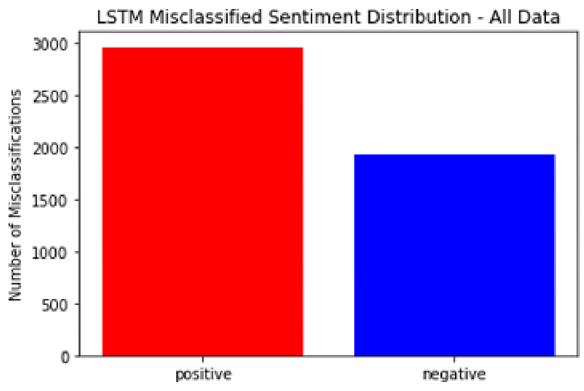


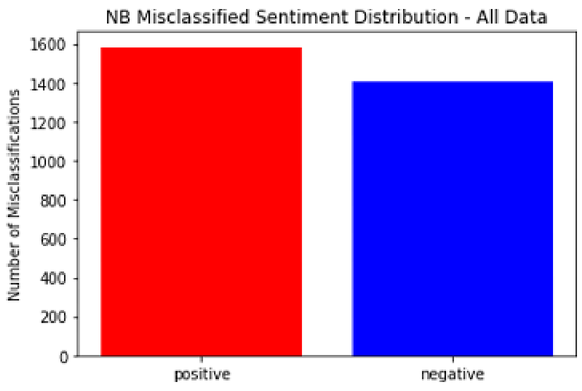
Most frequent words for **incorrectly** classified **positive** reviews



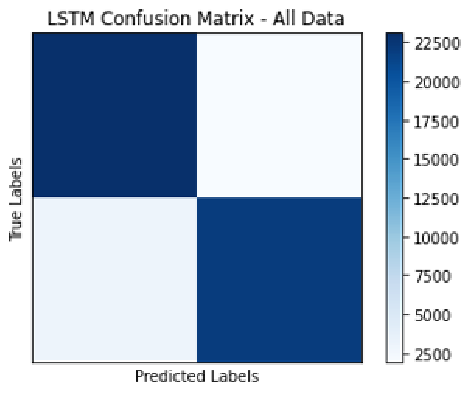
Most frequent words for **incorrectly** classified **negative** reviews

Among the correct classified reviews we find words that we expect to see such as love, great, good for positive reviews and bad for negative reviews. It appears the model has been able to learn these associations to some degree. It is also worth pointing out that the word "good" is fairly common and appears as a frequent word in all four subsets

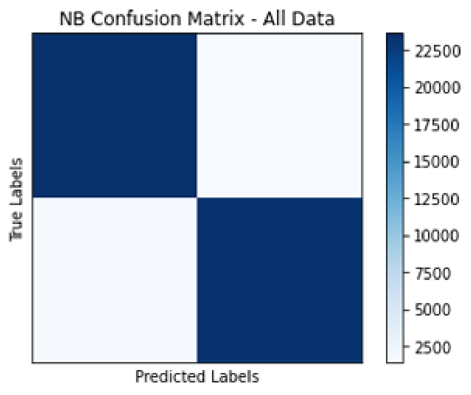




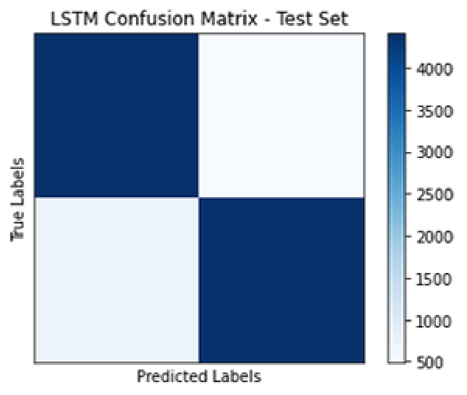
This can also possibly help to explain why both models have more difficulty in correctly classifying positive reviews, since in our dataset the word good commonly appears in both negative and positive reviews. This is because it is also common for many reviews in our data to contain both positive and negative comments about the movie, sending mixed signals that our models have more difficulty placing a polar classification label on.



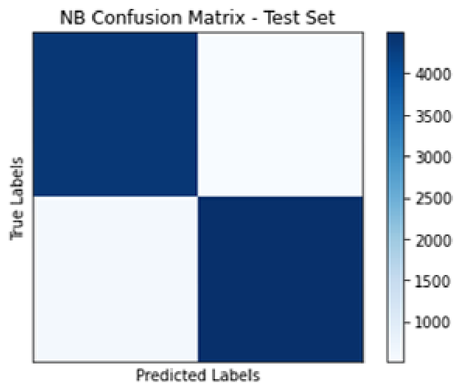
True Positives: 23078 (46.16%)
False Negatives: 1922 (3.84%)
False Positives: 2959 (5.92%)
True Negatives: 22041 (44.08%)



True Positives: 23592 (47.18%)
False Negatives: 1408 (2.82%)
False Positives: 1584 (3.17%)
True Negatives: 23416 (46.83%)



True Positives: 4406 (8.81%)
False Negatives: 494 (0.99%)
False Positives: 708 (1.42%)
True Negatives: 4392 (8.78%)

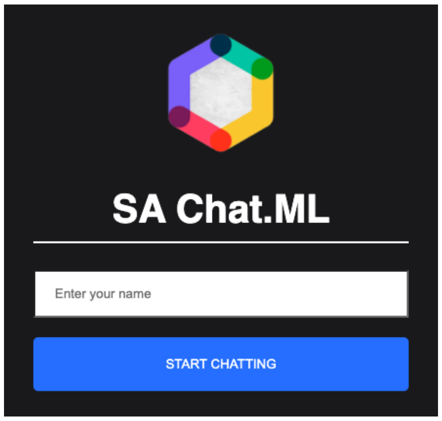


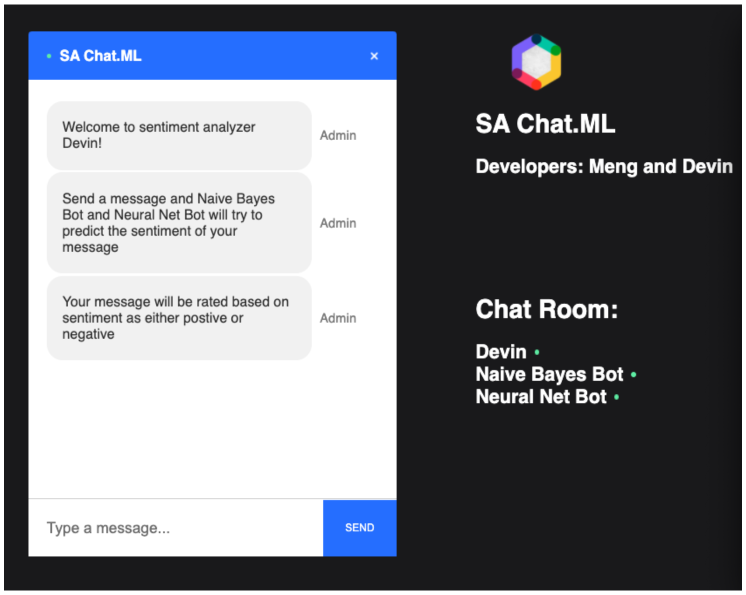
True Positives: 4378 (8.76%)
False Negatives: 522 (1.04%)
False Positives: 604 (1.21%)
True Negatives: 4496 (8.99%)

Both models appear to suffer from this issue with less polar reviews but still achieve relatively high accuracy on this task of sentiment classification. The Bayesian model outperforms the Recurrent model by just one percentage point for this dataset. As our overall goal involves general sentiment classification not confined to this domain specific data, we would like to experiment with existing pre-trained word embeddings as we believe this will allow better generalization to this task outside of this dataset.

Data Application Deployment

As part of our goal for this project, we have built and deployed a data application that integrates predictions from the two trained models. (https://sachatml.appspot.com/)



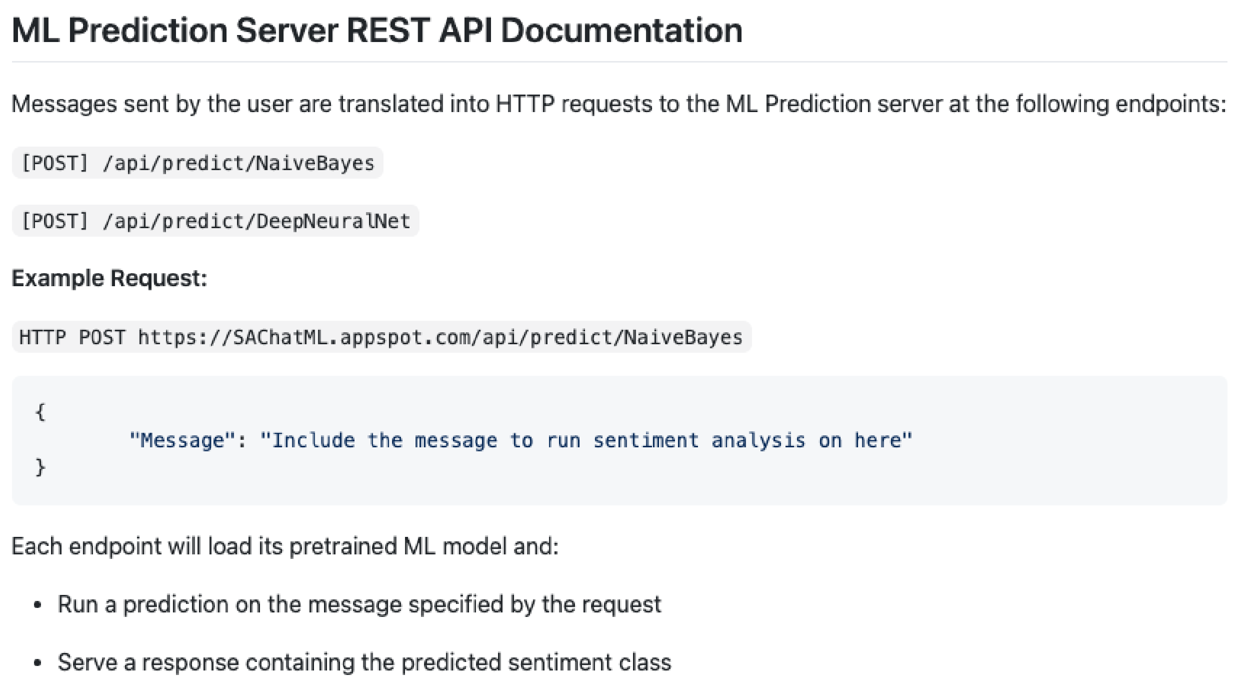


Welcome to sentiment analyzer Devin!

Send a message and Naive Bayes Bot and Neural Net Bot will try to predict the sentiment of your message

Your message will be rated based on sentiment as either postive or negative

The front end UI was constructed using the ReactJS framework and sends user input messages to a RESTful prediction api run using the Python Flask library.



ML Prediction Server REST API Documentation

Messages sent by the user are translated into HTTP requests to the ML Prediction server at the following endpoints:

`[POST] /api/predict/NaiveBayes`

`[POST] /api/predict/DeepNeuralNet`

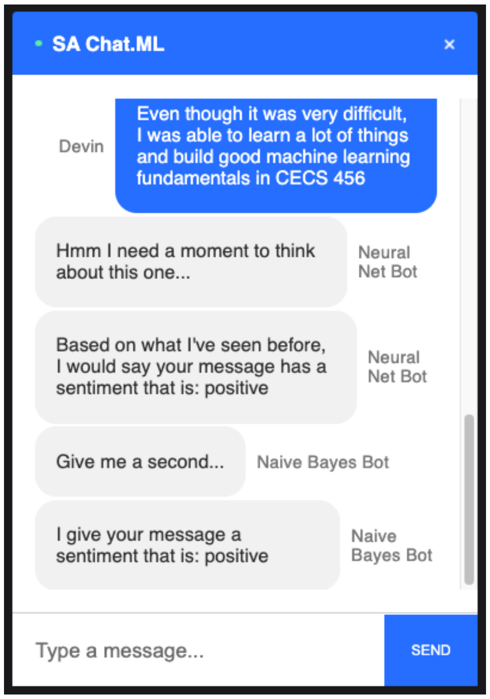
Example Request:

`HTTP POST https://SAChatML.appspot.com/api/predict/NaiveBayes`

```
{
        "Message": "Include the message to run sentiment analysis on here"
}
```

Each endpoint will load its pretrained ML model and:

- Run a prediction on the message specified by the request
- Serve a response containing the predicted sentiment class



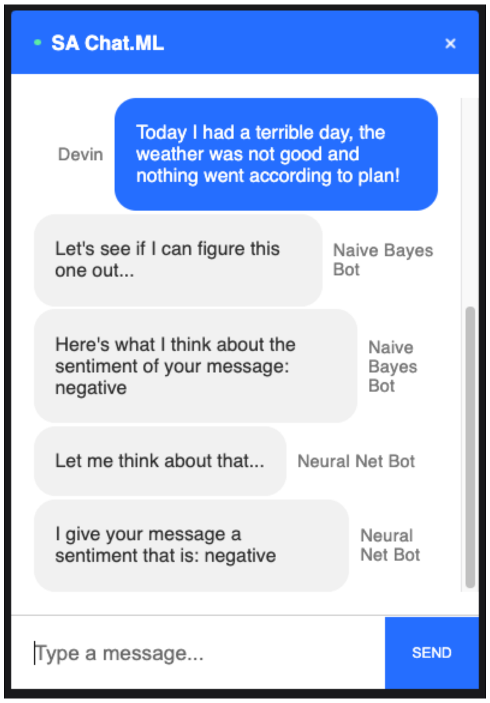
Devin: Even though it was very difficult, I was able to learn a lot of things and build good machine learning fundamentals in CECS 456

Neural Net Bot: Hmm I need a moment to think about this one...

Neural Net Bot: Based on what I've seen before, I would say your message has a sentiment that is: positive

Naive Bayes Bot: Give me a second...

Naive Bayes Bot: I give your message a sentiment that is: positive



Devin: Today I had a terrible day, the weather was not good and nothing went according to plan!

Naive Bayes Bot: Let's see if I can figure this one out...

Naive Bayes Bot: Here's what I think about the sentiment of your message: negative

Neural Net Bot: Let me think about that...

Neural Net Bot: I give your message a sentiment that is: negative

The trained models themselves are hosted on Google Cloud Platform's (GCP) AI Platform in which Flask api calls communicate with the cloud hosted models for prediction via HTTP requests. The entire machine learning integrated data application is deployed on GCP App Engine using docker containerized image files.

Distribution of Responsibilities

Meng	• Data preprocessing • Naive Bayesian model • ReactJS components
Devin	• Data exploratory analysis, figures, analysis • Recurrent Network LSTM model • RESTful api integration and GCP deployment

References

Brownlee, Jason. “How to Use Word Embedding Layers for Deep Learning with Keras.” *Machine Learning Mastery*, 1 Feb. 2021.

CLEMENT, Guillaume J. “Why & How to Use the Naive Bayes Algorithms in a Regulated Industry with Sklearn: Python + Code.” *Medium*, Towards Data Science, 30 Nov. 2020.

Frassetto, Stefano. “You Should Try the New TensorFlow's TextVectorization Layer.” *Medium*, Towards Data Science, 3 Aug. 2020,

Maas, Andrew L. “Large Movie Review Dataset.” https://ai.stanford.edu/~amaas/data/sentiment/.

Naive Bayes Text Classification, nlp.stanford.edu/IR-book/html/htmledition/naive-bayes-text-classification-1.html.

Perera, Shanika. “Positive or Negative? Spam or Not-Spam? A Simple Text Classification Problem Using Python.” Medium, Towards Data Science, 28 June 2019.

Sinha, Nimesh. “Understanding LSTM and Its Quick Implementation in Keras for Sentiment Analysis.” *Medium*, Towards Data Science, 3 Mar. 2018,

Tensorflow “Text classification with an RNN” https://www.tensorflow.org/tutorials/text/text_classification_rnn