CS-4375 Final Project

Sentiment Analysis

Travis Guillett
UT Dallas Compartment of Computer Science
Travis.Guillett@utdallas.edu

Tryston Minsquero
UT Dallas Compartment of Computer Science
tam190005@utdallas.edu

Abstract

(Travis Guillett: 50%, Tryston Minsquero: 50%)

This report goes over our implementation of sentiment analysis using streamlit, keras, and nltk as the predominant libraries and frameworks. The goal is to predict given a body of text whether the sentiment is "positive" or "negative". Models explored are the Recurrent Neural Network model and the Long Short Term Memory model. Ultimately the Long Short Term Memory model is used for the implementation. Tweaks to these models to account for the statistical nature of reviews are also considered and explored.

1. Introduction

(Travis Guillett: 50%, Tryston Minsquero: 50%)

1.1 Overview

Our project deals with one specific application of NLP: sentiment analysis. Sentiment analysis is the use of NLP to analyze the emotions and opinions conveyed in a given set of data. Naturally, this is an important tool for customer service and measuring performance metrics. To put the motivation bluntly: some companies can get large volumes of reviews, and it doesn't make sense to have a human sift through them manually to determine if they are a good rewiew or bad review. Sentiment analysis saves time by automating much of the process. Sentiment analysis includes detecting statements like "I like that" are positive, and statements such as "that is ugly" are negative, and our project deals specifically with that type of classification. Various other neural network models are unfit for this purpose due to a lack of longer-term memory. There are some models that are fit for this purpose, however. A popular one is LSTM, which is the one we will be using.

1.2 Natural Language Processing

Natural Language Processing (NLP) is a field in Artificial Intelligence dealing with understanding human languages, particularly the written and oral forms of such [1]. There are various other applications of NLP:

- Chatbots (for instance, automated support to reduce workload of customer service representatives)
- Language translation (for instance, Google Translate)
- Speech recognition (for instance, Amazon Alexa, Google Assistant, speech-to-text translation)
- Auto-complete
- Automated proof-reading (such as in Grammarly)

Natural language processing comes in these 5 steps [1]:

- Lexical analysis
- Syntactic analysis
- Semantic analysis
- Discourse integration
- Pragmatic analysis

The first step, lexical analysis, is the separation of a body of text into tokens for analysis, which involves removing formatting quirks.

The second step, syntactic analysis, regards the ordering of words in a body of text, as well as the grammatical correctness of the body of text. Part of syntactic analysis is finding the infinitive (root) version of a word, which is called stemming.

The third step, semantic analysis (which may benefit from stemming) regards finding the meaning of words and determining if they are meaningful in a body of text (usually done through a dictionary).

The fourth step, discourse integration, is the analysis of context in a body of text; that is, analyzing how given words in a body of text impact other words in that text.

The last step, pragmatic analysis, regards getting insight and context from the body of text.

Our project is mainly concerned about the first 3 steps of NLP. We mainly care about converting the text of a review into something a machine learning algorithm can work with. Usually these models work with a numbers, specifically floating-point numbers. Hence, we need to vectorize our text. To vectorize the text means to convert it into a sequence of numbers which represent the words in it.

1.3 Recurrent Neural Networks and Long Short Term Memory

The model that our project uses which will use this vectorized text, as previously mentioned, is Long Short Term Memory (LSTM). This is based off the Recurrent Neural Network (RNN) model. RNN is a supervised deep learning algorithm [1] that allows neurons to use their previously stored information as input for themselves in the future. This is something that traditional neural networks do not do. Instead, they essentially start “thinking” the moment they are asked to predict, instead of using past “thoughts” to inform decisions [4]. RNN and LTSM try to do that through the use of a loop.

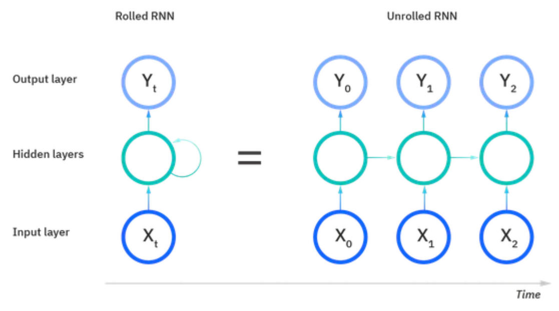


Fig. 1. IBM’s visualization of RNN [2]

This is essentially how the model “remembers” information. Information at one point in time is reused as information at the next point. The reason this is useful in our case is that when reading a paragraph, it is important to have context of what it just read.This way the model can make better predictions as it has context over everything it needs. However, this is fairly typical when doing text sentiment analysis.

The issue with RNN is one of diminishing returns. Unlike other neural networks, weights are not updated the same way. Instead, we must propagate through time in a model like RNN [1]. The gradient used for determining how to tune the model is multiplied by weights from older iterations. This results in a gradient that starts to shrink, resulting in increasingly smaller changes to the system.

Essentially, as you back-propagate through time, you get a pattern that looks like this:

$$w_{new} = w_{old} - L\nabla$$

Where w_{new} is the new weight, w_{old} is the old weight, L is the learning rate, and ∇ is the gradient. As we iterate this formula over time for a longer sequence, the $L\nabla$ term will diminish.

So how do we get around the diminishing weights over time? We can try and connect more past information to inform how the model tunes the weights, as a look at just the most recent information won’t cut it. The answer is LSTM, a model that handles these long-term dependencies [1], [4].

LSTM does this by factoring in not only the previous state as input, but also input from the current time instance as well as an additional portion of memory which contains information from states further in the past.

One of the inputs for the neurons in this model simply transfers data to the next neuron relatively unchanged [4]. This is the cell’s ”state”.

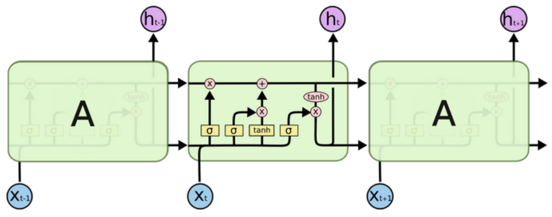


Fig. 2. Olah’s visualization of LSTM [4]

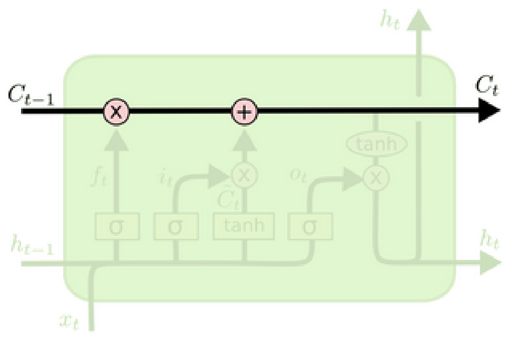


Fig. 3. Olah’s visualization of the cell state input/output [4]

Information can be altered through the use of gates, meaning we can add or remove information from the cell state [1], [4]. Part of this is achieved through a sigmoid layer and simple multiplication.

2. Related Work

(Travis Guillett: 50%, Tryston Minsquero: 50%)

We make use of information pertaining to the LSTM model from various sources. Many of them cover the same sort of process of how to use it, but explain various parts in different levels of depth.

S. Li [3] tries the recurrent neural network approach on a different dataset than the one we used. The approach required padding or truncation to input text since the model required inputs to be the same length. Then, during the lexical analysis portion, words are converted into IDs (that is, they are tokenized). As outlined earlier, RNN has a problem of diminishing returns, so while [3] touches on an important precursor to LSTM, we can do better.

We opt to use the approach from the Analytics Vidhya article by Santhosh [1]. The data set used by them as well as our project comes from a set of reviews of Amazon Alexa, with their body of text paired with a numerical rating (reference [7]). Additionally, we make use of another data set: reviews of smartphones purchased on Amazon (reference [8]). The idea behind using this portion of the data in particular is that lower-ratings tend to come along with text with a negative sentiment. Hence, we use the rating and text body pairs to train our model to predict that statements like “I love it” are positive, and that statements such as “Not much features” are negative. Picking up on more subtle hints of positivity or negativity is also desired.

Reference [1] already contains code to pre-process and train the model on the Amazon Alexa reviews, which our code will be based on. The unaltered code can already classify reviews with 90.9% accuracy, which is not much better than the RNN model from Towards Data Science. Hence, it will need to be tuned to achieve better performance.

Worth acknowledging as well are the articles that Santhosh refers to, particularly the one by Olah (reference [4]). This article goes a bit further in-depth about the nature of RNN and LSTM and how it works, as well as the math surrounding it. It also details variations of the LSTM model, as well as what may come in the future as an improvement of LSTM (though this will not be implemented in our project).

S. Virahonda [5] and W. Koehrsen [6] also wrote helpful articles that illustrated some of the motivation behind the neural network design from K. T. Santhosh [1], particularly the use of embedding and dropout layers.

This project utilizes what was made available from these sources to make an accurate model for our sentiment analysis task. Ultimately much of the ideas are derived from references [1] and [5].

3. Proposed Approaches

(Travis Guillett: 50%, Tryston Minsquero: 50%)

In this section, we first go over how we do exploratory data analysis on the reviews. That is, how we prepare the training and evaluation data. Then, we explore how we implement the LSTM model as well as how we tweak it.

3.1 Problem Definition

Given a review with a string of text, the task of the sentiment analysis is to detect whether the review is positive or negative without looking at the numerical rating. In a sense, this is problem is very close to that of guessing the rating, just that we only care about a positive and negative region.

3.2 The Basic Model

Reference [1] gave us a guideline for how to approach our problem of sentiment classification.

1) Load the data set and convert it into usable data in our program
2) Classify the review sentiments as positive or negative based on their numeric rating
3) Remove null values from the data set
4) Clean up unnecessary white-space, punctuation, capitalization, URLs, and other symbols and formatting quirks
5) Remove “stop words”, that is, words that contribute nothing to the sentiment on the text. This can be as simple as removing short or long words, but in this instance we use a dictionary of words to remove.
6) Convert the review into a numerical vector, that is, a vector of numeric identifiers for the words. This involves culling words that appear infrequently as well.
7) Build the LSTM model using a library such as Keras. This is where the parameters and hyperparameters would be tweaked.
8) Split the data into training and testing portions
9) Evaluate the model’s accuracy

This is a fairly solid approach, given the source created a model with 90.9% accuracy using the same data set.

There are other ways this can be tweaked. This includes:

- Tweaking the threshold for a positive rating and negative rating
- Tweaking the stop words
- Using a different variant of LSTM
- Using a different activation function
- Adding, removing, or altering layers in the neural network

4. System Design and Implementation

(Travis Guillett: 50%, Tryston Minsquero: 50%)

This section will cover how we designed and implemented the ideas discussed before. We will cover topics on how we used keras for the model, Streamlit for the interactive web app, and NLTK for processsing text. We will not go too in-depth with each framework, but the steps we took to achieve our goals with them.

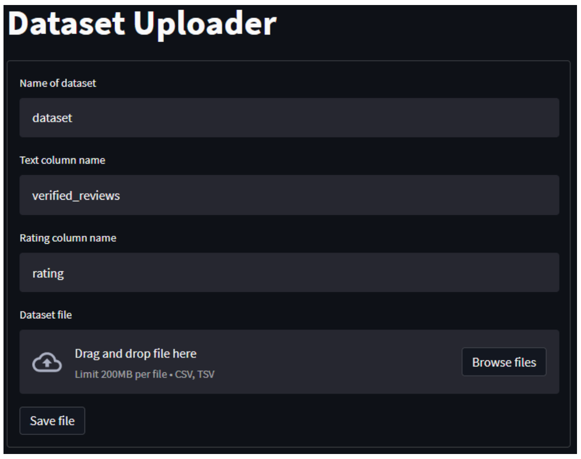
4.1 Overview

Currently the system is designed to be used through a framework called Streamlit, which lets one interact with a Python program using a web-based UI. This allows us to easily output data to the browser and keep everything organized in different files. The purpose of this was to easily create and modify different models and add and analyze the data in a meaningful way that can be seen and used by others. More importantly with each other.

Our web app has five different pages as of now: Dataset Uploader, Data Visualizer, Model Design, Training, and Predicting. Each page has its own purpose for the end result, making it easier to make the best sentiment analysis model. However, the pages are subject to change, in terms of the number of pages and their content. The main page is displays the current page, implemented with `main.py` file.

4.2 Dataset Uploader

The Dataset Uploader page allows the user to import data from a .csv or a .tsv file, which saves the data to use on other pages of the app. This is implemented with the `data_uploader.py` file. The user also must specify



Dataset Uploader

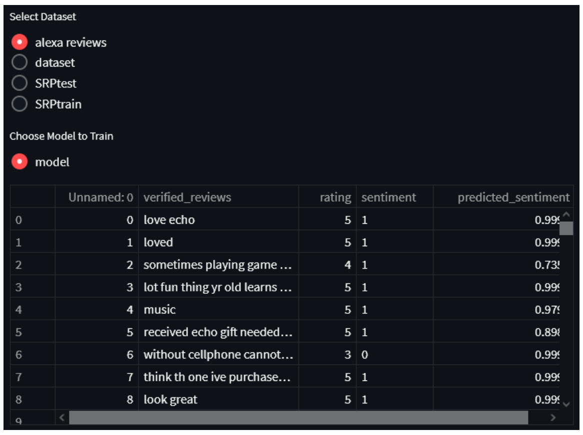
Name of dataset: dataset

Text column name: verified_reviews

Rating column name: rating

Dataset file

Fig. 4. The UI we created with Steamlit that let us add more training/evaluation data



Select Dataset

[✓ alexa reviews]
dataset
SRPtest
SRPtrain

Choose Model to Train

[✓ model]

	Unnamed: 0	verified_reviews	rating	sentiment	predicted_sentiment
0	0	love echo	5	1	0.99
1	1	loved	5	1	0.99
2	2	sometimes playing game …	4	1	0.73
3	3	lot fun thing yr old learns …	5	1	0.99
4	4	music	5	1	0.97
5	5	received echo gift needed…	5	1	0.89
6	6	without cellphone cannot…	3	0	0.99
7	7	think th one ive purchase…	5	1	0.99
8	8	look great	5	1	0.99
9					

Fig. 5. The UI we created with Steamlit that let us plot data. Plots are not pictured here since they are shown in various other figures in this report.

the column names for the numerical rating and text that will be analyzed. The file will be saved onto the machine hosting the web app by streamlit using pandas, so it can be used later and converted into a pandas Dataframe. The pre-processed data set still needs to be parsed into something our program understands. The data set we use comes in the format of tab-separated values or comma-separated values, so it is a matter of reading which columns were which, splitting each row up and casting/parsing the values as their proper types, then creating the proper entry objects that the Python code can use.

Regardless of how we do our model, this is a necessary step.

The main reason for implementing this rather than manually adding each dataset, is that it allows us to easily add datasets for the model to train and test on. The first dataset we used was the amazon alexa reviews, which happened to be extremely biased towards positive reviews. This ultimately led to early iterations of our model being extremely biased to giving a positive sentiment on almost any text. Having the ability to add a variety of datasets lets us avoid this much easier. We later added smartphone reviews on amazon which was much larger and contained more negative reviews to train on.

4.3 Data Analysis

The next page we have is Data Visualization which is implemented through the `data_visual.py` file. This allows the user to pick a dataset they uploaded and analyze properties of the data. This allows the user to make better decisions for training. Making it easy to notice if there is any issues with the data, with a large bias. Additionally, showing the effect that cleaning has the dataset which will be input into the model.

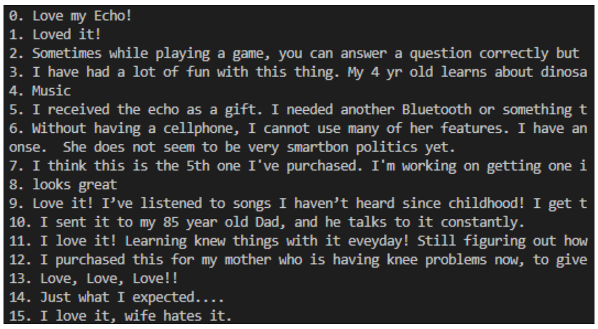
4.3.1 Cleaning the text

Before we actually can use the data, it is best to modify to clean up the text as stated before. Most of the implementation for this is in the `data_cleaner.py` file, which includes functions to modify the dataset dataframe. The process of cleaning the data goes through the process:

1) Converting the text into one case (we chose lowercase)
2) Removing all special characters
3) Removing all digits and numbers
4) Removing all stopwords (used NLTK's english stopwords)
5) Convert all words into their lemma (Lemmatization)

One issue we had was using NLTK the NLTK downloader with Streamlit, specifically stopwords. We ultimatily had to download it to the repository and call it, but was not a big hiccup.

Modifying the data like this should not actually effect the sentiment given. It is more of a way to help the model only use key aspects of a text in order to predict a sentiment.

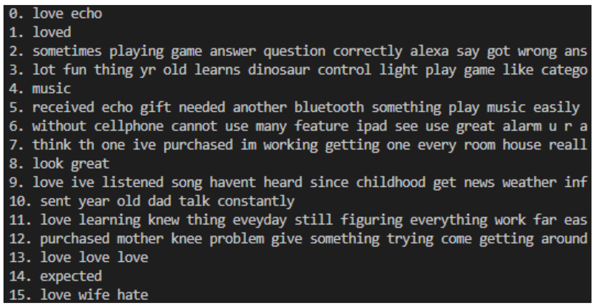


```
0. Love my Echo!
1. Loved it!
2. Sometimes while playing a game, you can answer a question correctly but
3. I have had a lot of fun with this thing. My 4 yr old learns about dinosa
4. Music
5. I received the echo as a gift. I needed another Bluetooth or something t
6. Without having a cellphone, I cannot use many of her features. I have an
onse.  She does not seem to be very smartbon politics yet.
7. I think this is the 5th one I've purchased. I'm working on getting one i
8. looks great
9. Love it! I've listened to songs I haven't heard since childhood! I get t
10. I sent it to my 85 year old Dad, and he talks to it constantly.
11. I love it! Learning knew things with it eveyday! Still figuring out how
12. I purchased this for my mother who is having knee problems now, to give
13. Love, Love, Love!!
14. Just what I expected....
15. I love it, wife hates it.
```

Fig. 6. Data from [7] before cleaning

4.3.2 Applying Sentiments

One other minor step is determining what data from the training set is considered "positive" and what is considered "negative". The approach we use is probably the simplest: set a threshold on the numerical rating. In our case, a rating above a three indicates a positive review, and anything else indicates a negative review. Thus we are using a



```
0. love echo
1. loved
2. sometimes playing game answer question correctly alexa say got wrong ans
3. lot fun thing yr old learns dinosaur control light play game like catego
4. music
5. received echo gift needed another bluetooth something play music easily
6. without cellphone cannot use many feature ipad see use great alarm u r a
7. think th one ive purchased im working getting one every room house reall
8. look great
9. love ive listened song havent heard since childhood get news weather inf
10. sent year old dad talk constantly
11. love learning knew thing eveyday still figuring everything work far eas
12. purchased mother knee problem give something trying come getting around
13. love love love
14. expected
15. love wife hate
```

Fig. 7. Data from [7] after cleaning

binary classification as our output would be two possible outcomes.

There is clearly a few problems that come with this categorization as it does not encompass all sentiments and the cut off is arbitrarily picked. We have the model design to include more than two categories, but it is not completely implemented as it takes away from the scope of this project. However, if we were to look into this further, we could consider including a neutral category or having the model predict the exact numerical rating of the review.

4.3.3 Vectorization

After cleaning the dataset, we now move onto the lexical analysis. Reviews are vectorized, that is, the words are converted into numerical ids, and reviews become vectors of such ids which will be the input for the model. Because we have already removed useless data from cleaning, we have optimized the input to be as similar as possible. Not having an abundance of different ids. That being said, we still will still a max number of words that will be tokenized. This is useful so we have an upper limit on what the input dimension will be. This is a value which the model should know. We arbitrarily picked 500 as the default, but this should be tweaked for further optimization, and used during model creation.

4.4 Model Design

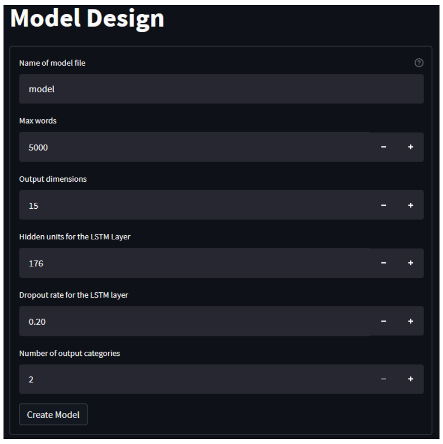
The next page we have, pictured in figure 8, is Model Design which is implemented through the `model_design.py` file. This purpose of this page allows the user the create a new model with a template that was designed for text sentiment analysis, easily allowing the tweaking of hyper parameters. The model can be saved and used in other pages, similar to how datasets are implemented.

To be clear: this page allows for one to work with several different models in parallel. This allows us to compare models with different parameters and hyper parameters.

4.4.1 Final Design

The model consists of the following layers with the hyper parameters you can modify via the web app UI:

1) An Embedding layer
   a) Max words (Input dimension)



Model Design

Name of model file: model

Max words: 5000

Output dimensions: 15

Hidden units for the LSTM Layer: 176

Dropout rate for the LSTM layer: 0.20

Number of output categories: 2

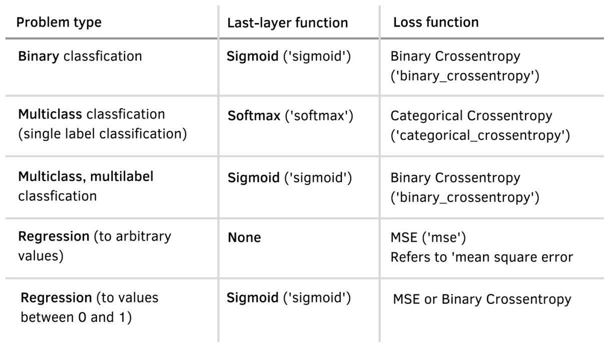
Fig. 8. The UI we created with Steamlit that let us tweak the model parameters

   b) Output dimension for embedding layer
2) An LSTM layer
   a) Hidden units for the LTSM layer
   b) Dropout rate for the LTSM layer
3) A Dense layer
   a) Number of output categories

The embedding layer is the first layer our model has. This is extremely vital and will be necessary no matter what type of neural network we design. The layer must take two arguments: number of possible tokens and dimensions of the embeddings [5]. Thus, the first argument must greater than or equal to the max number provided in vectorization. Otherwise we will have an error as the input will be large than the input dimension of the model. We picked 500 as the default we chose this during vectorization. The second value will be the output dimension for the next layer (which in the case of the template, it is the LTSM layer).

The LTSM layer is next. The LTSM layer can take a variety of arguments but we only have two for the template: hidden units and dropout rate. The hidden units is the dimensions of the output. The dropout rate is the rate the model is randomly turning off neurons instead of relying on all of them instead during training. This is "one of the most effective and commonly used in regularization techniques" [5]. This forces it to find more meaningful patterns to optimize metrics, which in this case is accuracy.

Our last layer is the Dense layer. The Dense layer gives us the final outputs. Only one argument is used via Streamlit as the rest are inferred from that one argument. The argument is the number of output categories is the number of classes the model classifys. Because we are using 'Negative' and 'Positive', it is a binary classification and be 2. Because it is a Binary classification, we use a sigmoid activation function and a binary cross-entropy for the loss function. We decided on that from the table seen in figure 9.



Problem type	Last-layer function	Loss function
Binary classfication	**Sigmoid** ('sigmoid')	Binary Crossentropy ('binary_crossentropy')
Multiclass classfication (single label classification)	**Softmax** ('softmax')	Categorical Crossentropy ('categorical_crossentropy')
Multiclass, multilabel classfication	**Sigmoid** ('sigmoid')	Binary Crossentropy ('binary_crossentropy')
Regression (to arbitrary values)	**None**	MSE ('mse') Refers to 'mean square error
Regression (to values between 0 and 1)	**Sigmoid** ('sigmoid')	MSE or Binary Crossentropy

Fig. 9. Table for activation and loss functions [5]

We do allow for the increase in the number of categories, which would be considered to be multi-class classification. In which case we use softmax as the activation function and categorical cross-entropy as the loss function. So for example, number of categories is 3, the outputs would be 'Postive', 'Negative', and 'Neutral'. However, this is not fully implemented yet as it goes outside of the scope of this project. If given enough time though, we would like to look into this further.

We can see the code of how we design the model in Listing 1.

```python
model = Sequential()
model.add(Embedding(num_words, output_dim))
model.add(LSTM(hidden_units, dropout=dropout))
model.add(Dense(category_num, activation='sigmoid' if category_num == 2 else '
        softmax'))
model.compile(loss = 'binary_crossentropy' if category_num == 2 else '
        categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Listing 1. The model defined in the Python code

The overall template design of our model was influenced from different sources, but the final iteration is more derived from [5]. This keeps the model simple in order to easily modify the hyper parameters. Many of the default values are also set to decent values already.

4.4.2 Training & Evaluation

During the course of training the model, we initially used our data set for mostly training, and did a conservative amount of evaluation. This proved to be a flawed approach which actually exacerbated the positivity bias we end up covering in the empirical analysis. Initially, sentiments were very heavily skewed.

Note how in figure 10 the model overwhelmingly and confidently classifies 4 and 5 star reviews as positive (the circles shown are outliers). Then note how for lower rated reviews, the model appears to be unsure of where the sentiment lies.

4.5 Training Environment

The training environment to train a model with a given dataset can be found on the Training page. This page is implemented through the 'training.py' file and uses the datasets and models that should have been produced before hand.

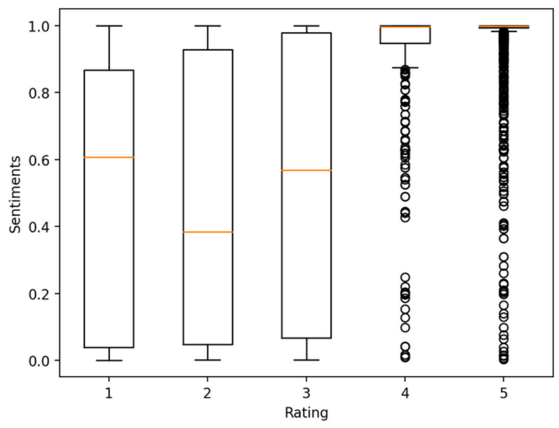


Fig. 10. Initial skewed sentiments plotted against ratings, but with 4-star cut off.

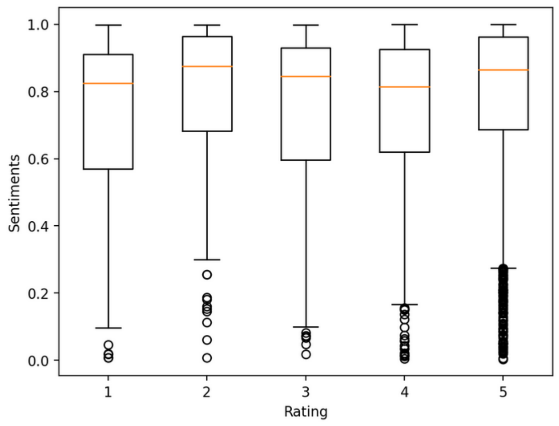
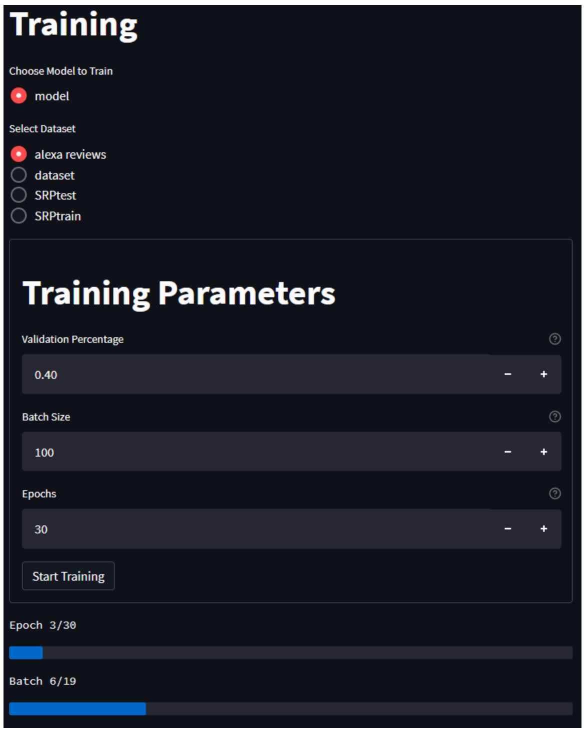


Fig. 11. Initial skewed sentiments plotted against ratings, but with 3-star cut off, where more positivity bias is present.

After the user has selected a model and dataset to train with, the user also must selected a validation percentage, batch size, and number of epochs. The validation percentage is what is used with sklearn's `train_test_split` function. So for example, let x be the validation percentage where $0 < x < 1$. The model will train with a random set from the dataset selected that is $(1-x)\%$ the size of the dataset. Then once it has completed training, the model will evaluate itself with the remaining data in the dataset, which would be $x\%$ of the dataset.

This a common practice, and lets use quickly analyze our model on a given dataset. This also always training to have multiple functions. So if we want to only train, we would set the validation percentage to .01. Conversely, if we only wanted to validate, we would set the validation percentage to .99.

Batch size and epochs is simply the values that get passed to "fit" function that is passed. We show a progress bar that shows less information on the web app during training than a console would. During training you would be able to see the loss and accuracy for each batch in order to monitor training. However, when running the project


Training

Choose Model to Train

[✓ model]

Select Dataset

[✓ alexa reviews]
dataset
SRPtest
SRPtrain

Training Parameters

Validation Percentage: 0.40

Batch Size: 100

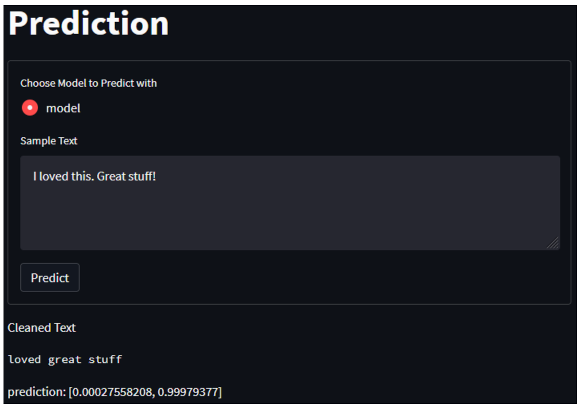
Epochs: 30

```
Epoch 3/30
Batch 6/19
```

Fig. 12. The UI we created with Steamlit that let us train/evaluate the model

locally this is still visible in the console, and the final result of accuracy and loss is presented when training is over. The training is automatically saved to the model and can be looked upon further in predictions and data visualizations.

4.6 Predicting


Prediction

Choose Model to Predict with

[✓ model]

Sample Text

I loved this. Great stuff!

Cleaned Text

```
loved great stuff
```

prediction: [0.00027558208, 0.99979377]

Fig. 13. The UI we created with Steamlit that let us see the model's output

The last page is the predict page. This page is fairly simple, it simply outputs a prediction based off model and a sample text. This was useful if we want to test a model on specific text or words. The page is implemented with 'predict.py'.

5. Empirical Analysis

(Travis Guillett: 50%, Tryston Minsquero: 50%)

In this section we go over the results of training our model.

5.1 Predicted Sentiment Compared to Actual Ratings

Since one of the ideas in training the model with this data set was associating positivity and negativity with the numeric rating of a review's text, we might expect the model to predict that text from a 5-star review is positive, text from a 1-star review is negative, and text from reviews with other ratings sit somewhere between. We may also expect that given we use a threshold and classify positive & negative in a binary way during training, we may get a sharp transition from negative to positive at the threshold, rather than a gradual change.

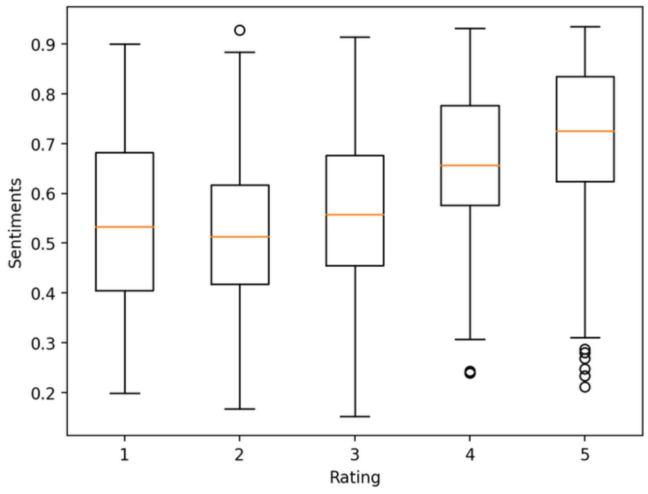


Fig. 14. Predicted sentiments for one of the data sets

As it turns out, we do get a very gradual change, but the sentiment prediction seems to be very positive-leaning, even for lower rated reviews.

5.2 Issues with Numerical Ratings

We found that most positive and negative reviews actually sit on extremes of the 5-star system used in the data sets we sourced. On one hand, this can allow us to simplify the system by gearing our classification around these extremes.

However, on the other hand, that leaves out the portion of reviews that do not fall on extremes. Plus, it doesn't account for another major hurdle in training the model using the numeric ratings: reviews tend to overwhelmingly be positive, particularly 5 stars. This resulted in a lot of data that, for what it's worth, may have had negative sentiments that were paired with positive reviews, making it more difficult to rely on ratings as a way of differentiating positive and negative reviews.

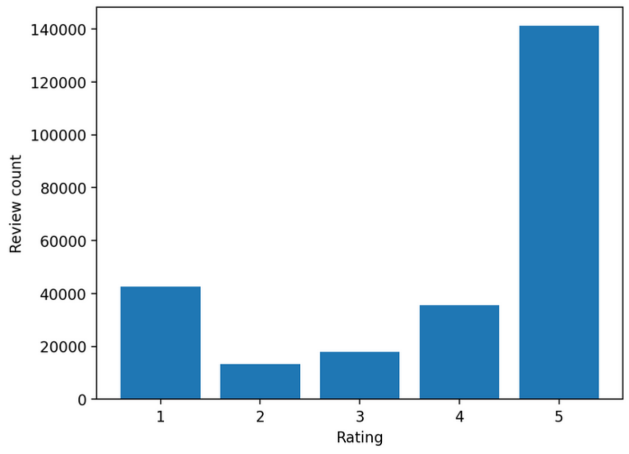


Fig. 15. Distribution of ratings from data set [8]

One hyperparameter we experiment with is how the rating is used to classify a positive or negative review. Do we consider 3 stars neutral, less than that negative, and more than that positive? Do we consider only 5 star reviews positive? The answer really depends on the nature of numerical ratings as a whole.

Our initial approach was to set the threshold at 3 stars, but this resulted in the overwhelming majority of reviews being classified as positive. Looking through the data sets ourselves, we found that reviews with a rating below 5 stars tend to contain some level of negativity. So we decided to instead go with the approach of classifying only 5 star reviews as positive, which noticeably improved the model's ability to detect negativity. However, the model seemed to get less accurate at classification. Picking a middle ground threshold of 4 stars didn't remedy the issue.

5.3 Issues with word frequency

Another issue with using data sets with overwhelmingly positive reviews is that the model will be trained to recognize words as coming from a positive context.

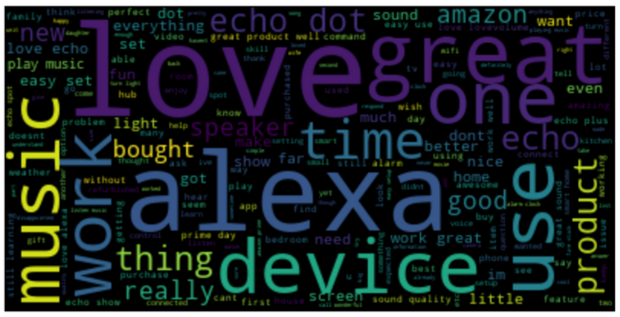


Fig. 16. Word cloud from data set [8]

Many of the words pictured in figure 16 are found overwhelmingly in positive reviews, even if otherwise neutral. Many words featured in negative reviews show up less frequently, leading to the model being uncertain as to how to classify them. For instance, during some iterations of the model, a few reviews we asked it to predict had the word “terrible” in them, and the model only had 30% confidence the review was negative.

6. Conclusion

(Travis Guillett: 50%, Tryston Minsquero: 50%)

We were not able to get the model to confidently predict a review is negative for the most part. However, we did notice reviews with actual negative sentiment or lower ratings tended to make the model less confident the sentiment is positive. So while a positivity bias is present and very prominent, there does seem to be some ability to detect a negative sentiment.

We did have a fairly accurate model that worked a good amount of the time, but we did not save it correctly and it was corrupted. This happened fairly late in our project, we were able to capture output and data to use in the report, but that was all that was left of it. As stated in the empirical analysis, our model was not to where we would have liked.

There are lots of reasons for this, it would have likely been better to focus on actually training a good model than a tool to do so. We also had to write this report which took a large amount of the time as well. However, the tools we developed allow us to really work with the model and train it with any kind of text and rating dataset. If given an extra day or two, we would likely be able to produce a model that is of much higher quality.

The amount of evaluation we did on the model against data sets drastically improved the predictions, but not enough to make the model confidently predict negativity.

In the future, we may seek out data sets that include predominately negative reviews to even out the training and evaluation data. However, in the real world, reviews tend to already be very skewed. There is usually many high reviews, or many low reviews with not as much in-between. Thus, finding lots of good datasets prove to be difficult.

In terms of the end result, we created a model that can differentiate between a lack of negativity, and hints or presence of negativity. Overall though, we created a tool to easily make text sentiment analysis models with a variety of different datasets. Allowing the ability to train and use the model in an understandable way.

Acronyms Used

LSTM Long short term memory.
NLTK Natural language toolkit.
NLP Natural language processing.
RNN Recurrent neural network.

References

[1] K. T. Santhosh, “Natural language processing - sentiment analysis using LTSM,” Analytics Vidhya, 27-Aug-2021. [Online]. Available: https://www.analyticsvidhya.com/blog/2021/06/natural-language-processing-sentiment-analysis-using-lstm/. [Accessed: 01-May-2022].

[2] “What are recurrent neural networks?,” IBM. [Online]. Available: https://www.ibm.com/cloud/learn/recurrent-neural-networks. [Accessed: 07-May-2022].

[3] S. Li, “A Beginner’s Guide on Sentiment Analysis with RNN,” Towards Data Science. [Online]. Available: https://towardsdatascience.com/a-beginners-guide-on-sentiment-analysis-with-RNN-9e100627c02e. [Accessed: 08-May-2022].

[4] C. Olah “Understanding LSTM Networks” [Online]. Available: https://colah.github.io/posts/2015-08-Understanding-LSTMs/. [Accessed: 07-May-2022].

[5] S. Virahonda “An easy tutorial about Sentiment Analysis With Deep Learning and Keras,” Towards Data Science. [Online]. Available: https://towardsdatascience.com/an-easy-tutorial-about-sentiment-analysis-with-deep-learning-and-keras-2bf52b9cba91. [Accessed: 08-May-2022].

[6] W. Koehrsen “Neural Network Embeddings Explained,” Towards Data Science.[Online]. Available: https://towardsdatascience.com/neural-network-embeddings-explained-4d028e6f0526. [Accessed: 08-May-2022].

[7] M. Siddhartha “Natural Language Processing : Sentiment Analysis,” Kaggle. [Online]. Available: https://www.kaggle.com/code/sid321axn/natural-language-processing-sentiment-analysis. [Accessed 01-May-2022].

[8] R. Aggarwal “Amazon Mobile Phone Reviews Dataset,” Kaggle. [Online]. Available: https://www.kaggle.com/datasets/rajatagg/amazon-mobile-phone-reviews-dataset [Accessed 02-May-2022].