Aryan Kapoor

Sentiment Analysis Model Report

Abstract

The Movie Review Sentiment Analysis model leverages a Convolutional Neural Network to conduct sentiment analysis on movie reviews. Using precleaned data provided by Stanford NLP with classifications, the model is able to gather the positive and negative sentiment of each review. After fine tuning hyperparameters and levels of the Neural Network, model accuracy reached 83.75%. The model then analyzes the sentiment for ~250 reviews for any given movie, scoring the movie based on a weighted average of the sentiments. Weights are determined by the inverse of the gaussian distribution of the sentiment scores. Scores for each movie are then collected in a database and uploaded into the UI.

Problem Description

I am a huge movie fan. I watch movies very often, and want to find a way to quickly gather what the public opinion is for a certain movie. My idea is to create a sentiment analysis model that analyzes movie reviews and then is able to create a score for what the public opinion of a movie is. I believe this is much better than calculating a weighted average for movie ratings (like you would see with IMBb or Rotten Tomatoes), because ratings are rather arbitrary when every individual person has their own system for giving a rating. By gathering sentiment analysis of each review, I believe that I would be able to calculate the consensus opinion of a movie in a much more standardized and objective way.

Approach

The pipeline begins with gathering data from the precleaned database provided by Stanford NLP. There are 50,000 total data points, evenly split between positive and negative reviews. 80% (40,000) of the data points were used for model training and the remaining 20% (10,000) were used for model testing. The Convolutional Neural Network was then built and fine tuned, with the following layers and hyperparameters:

- The embedding layer establishes the vocabulary size (with performance being maximized at a size of 5000)
- A 1D convolutional layer then analyzes patterns in the input, breaking down complex relations inside the language. This step is essential in any natural language model
- A pooling layer is applied to reduce dimensions and reduce information into a singular vector
- A fourth dense layer is applied with ReLU as the activation function (sigmoid and tanh were also options but performed worse)
- A dropout layer then takes place to reduce model overfitting (with the large size of the input, the model was very susceptible to overfitting, making this layer crucial)
- The final layer performs softmax to normalize results in the range [0, 1]

After the model was built, fit, and tested, then I began scraping reviews to gather scores for each movie. I wrote a selenium script to gather ~250 reviews for any given movie, which were then fed into the model.

After gathering the sentiment for each review. I then calculated a score for the movie using a weighted average. The weights for each review were calculated by the inverse of the following gaussian function:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

The final function used in the project with the proper values of σ and μ can be shown below:

$$\Phi^{-1}(x) = \text{inverse\_gaussian\_weight}(x, \mu = 0.5, \sigma = 0.1) = 1 - e^{-0.5\left(\frac{x-\mu}{\sigma}\right)^2}$$

After taking the weighted average and scoring each movie, the results were stored in a database with all of the corresponding movie titles, posters, and scores, and were then used in the UI portion of the project.

The UI portion of the project was somewhat straightforward, with the help of streamlit. I created a page with all of the movie scores and corresponding titles/posters, a page where the user can test the model by providing a paragraph for sentiment results, a page with the sentiment analysis test results, and an about page summarizing the project report.

Data Description

There are three main data sources used/produced in the project, which can be described below:

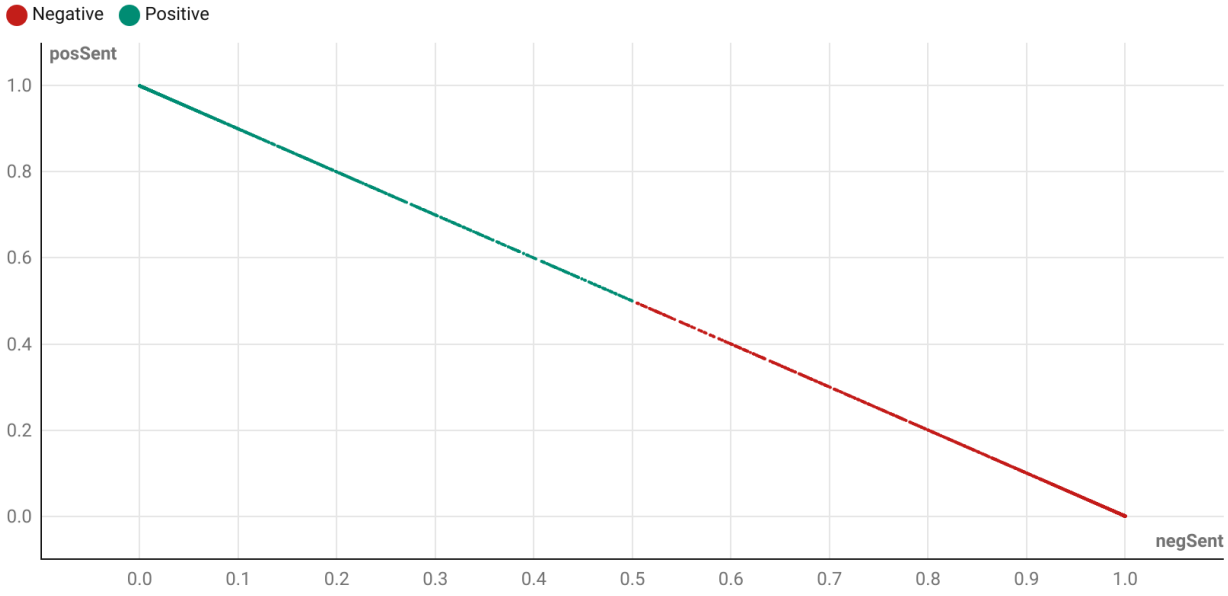
- Model Training Data: provided courtesy of Stanford NLP, containing 50,000 data points with binary outputs (positive/negative). In order to reduce model bias, only 25 reviews per movie were used.
- Model Test Results: The outcome of the model test post training is tracked on this sheet. 10,000 points were used for testing, with 83.75% accuracy.
- Movie Score Database: The list of movie scores and poster URLs on the website come from this sheet, which is constantly being updated. Scores are generated off of the top ~250 IMDb reviews

In terms of results from the model testing, I created two visualizations to show the distribution of data points after model fitting. Sentiments were plotted with negative sentiment on the x-axis and positive sentiment on the y-axis, and data points for each review had a strong linear correlation, which was ideal.

After observing the distribution of data points in the smaller test size, it became clear that there was no obvious boundary to discern between positive and negative reviews. Therefore, generating a score for a review was done with the decimal score, instead of a binary positive/negative. This led to the direction of the scoring experimentation that is described further later on in the report.

Sentiment Analysis Test Results

Results of the Sentiment Analysis Model test points, with an 83.75% accuracy

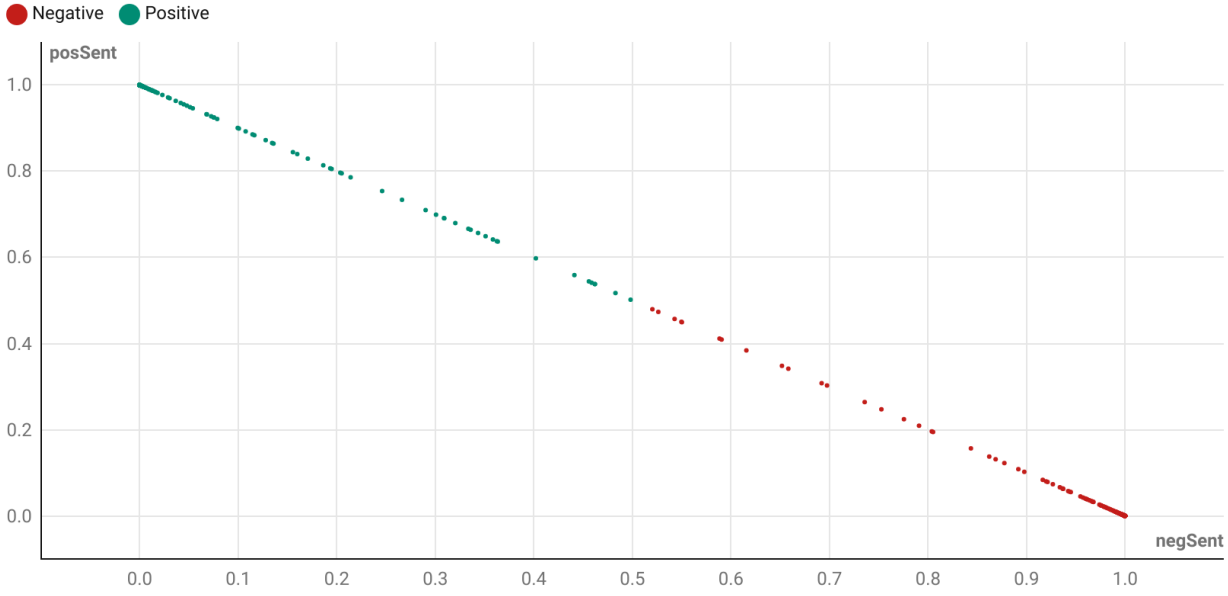


10,000 data points in total, representing 20% of the total data, with 80% of data points being used for model training.

Chart: Aryan Kapoor • Source: Link to Data • Created with Datawrapper

Sentiment Analysis Test Results (Shortened)

First 500 values of the Sentiment Analysis Model test results



First 500 of 10,000 values shown, in order to make individual datapoints discernible

Chart: Aryan Kapoor • Source: Link to Data • Created with Datawrapper

Experiments & Error Analysis

Model Experiments & Error Analysis

While coming up with the construction with the model, I played around and tested a variety of different parameters, in order to maximize the efficiency and accuracy of the model. Some parameters had large effects on the overall accuracy and runtime of the model, while other parameters played a very small part. After tokenizing and padding sequences, I settled on the structure of the Convolutional Neural Network and began testing each value of the layers.

Vocabulary size maximized model performance at a size of 5000 words. I tested ranges from 3000 to 15,000, but was only able to maximize efficiency at around ~80% with other vocab sizes. Although this metric didn't create a big difference in accuracy, 5000 words seemed like the sweet spot for the vocabulary.

Similar to the size of the vocabulary, the amount of epochs and batch size for model fitting had a similar behavior. I tried ranges of 5 to 15 epochs. Smaller amounts of epochs didn't let the model fully converge, leading to missed opportunity in increasing its accuracy. Larger amounts of epochs seemed to overfit the data quite easily, so a range of epochs around 10 seemed ideal. I started off fitting with a batch size of 32, which led to overfitting quite quickly. After some experimentation, I learned that because the training size was relatively large (40,000 values), the batch size needed to be significantly larger to maximize performance of the model. Model fits with different batch sizes/epochs can be shown below:

Batch Size	Epochs	Accuracy
2048	13	82.97%
1024	10	83.28%
600	10	83.41%
	13	83.01%
560	10	83.6%
	13	**83.75%**
550	13	83.36%
512	10	83.0%
	13	83.41%
	15	83.27%
256	10	83.41%

*Final results used in the model are bolded

I tried batch sizes in the range from 256 to 2048. While the differences in accuracy were pretty miniscule, I was determined to squeeze every point of accuracy that I physically could from the model. Once I

determined that 10/13 epochs were maximizing accuracy while preventing overfitting, I then began using a variety of batch sizes to try to see what had the greatest accuracy. From this, I eventually settled on having 13 epochs with a batch size of 560.

When determining what activation function would lead to the best results, one function clearly outperformed the rest. Sigmoid led to the worst results, while softplus was also a bit lackluster. ReLU was by far the most efficient and accurate, which goes in line for the majority of datasets. In order to normalize results after processing data in the model, softmax was the clear option to establish sentiments within the range of [0, 1].

In terms of optimizers, results were also similar to the activation function. AdaGrad was positive in preventing overfitting, but was far too slow and made the model take very long to converge. Other optimizers performed very poorly, which made Adam easily the best option as an optimizer.

Scoring Experiments & Error Analysis

After looking at the distribution of data points from the model test results, It was clear that there was significant clustering in the extreme regions, whether it was extremely negative sentiment or extremely positive sentiment. In order to reflect this when coming up with the general score for the movie, I determined the best way to do this was by taking a weighted average of the sentiments.

When deciding how exactly to weight sentiment scores for each individual review, I tried to score movies based on a variety of functions. I wanted to check scores by having two movies to monitor, a positive movie control and a negative movie control. I settled on using The Shawshank Redemption (1994) as the positive movie and Ghosted (2023) as the negative movie.

I started by trying to distribute scores based on the softmax function. However, because the final layer of the CNN model already distributed data through the softmax function, redoing this function did not lead to correct results. It ended up having the opposite effect, leading to scores that are stuck generally in a range of 60-80 for movies, which made softmax a bad candidate.

The sigmoid function then caught my attention, because of the horizontal asymptotes towards the extreme positive and negative regions. However, even after adjusting the function several times, I noticed very inconsistent results with movie scores. It seemed like weighting with sigmoid made results quite random, and was definitely not a good option as a weighting system.

After reading some theory, it became clear that the gaussian function has a similar distribution to what I was looking for. However, the normal distribution is the exact opposite of what I'm looking for, as scores in the middle have a higher value and scores on the ends have very small values. Therefore, I would need to generate scores with the inverse of this function. This ended up being the quantile function (Φ^{-1}), which, after testing through the controls and other movies with various sentiments, ended up leading to the results that I was looking for. Furthermore, weighting scores towards the middle (the area where results would be ambiguous and what led to the majority of error in model evaluation) as lesser would help minimize the error in the scores themselves. Conversely, weighing extreme scores helped create a larger distribution of scores for movies, which is what I was looking for.

After adjusting the weights of the σ and μ values of the quantile function (0.5 and 1, respectively), I finalized the scoring system, and was able to start generating scores for movies.

Conclusion

In conclusion, the Movie Review Sentiment Analysis model aims to create an objective, standardized system for scoring movies, capturing the public consensus opinion using review sentiment. This model leverages a Convolutional Neural Network and a scoring system based on the quantile function, in order to properly score movies as a percentage.

References

- Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. (2011). Learning Word Vectors for Sentiment Analysis. The 49th Annual Meeting of the Association for Computational Linguistics (ACL 2011).