COPS Summer of Code 2025

Intelligence Guild

NLP Track: Sequence Modeling

2 – 10 June 2025

RNN vs LSTM Analysis

Suyash Ranjan
24154022

1. Introduction

This report outlines the approach taken to solve the binary sentiment classification task as part of the CSoC Intelligence Guild track. The goal was to predict whether a given product review is **positive** or **negative** using Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) models. I explored pre-processing techniques, word embeddings, and sequential architectures, and evaluated them on real-world review data.
I also learned an important insight about Data-analysis and how a simple idea could save a lot of memory and time while also giving a stagnant increase in your model strength. I have discussed about that later in this report.

2. Dataset Description

We used the Amazon Reviews dataset containing :Amazon Reviews Dataset

- **polarity** – 1 (Negative), 2 (Positive)
- **title** – Title of the review
- **text** – Full product review

3. Pre-processing

The pre-processing steps included:

- Maped 1 as 0 and 2 as 1
- Lowercasing all text
- Removing punctuation, special characters, stopwords and also numbers.
- Tokenizing text into word sequences
- Padding sequences to uniform length

4. Word Embeddings

We experimented with:

- I didn't use any pre-trained embeddings, I did the embedding layer on the tokenized dataset

5. Model Architectures

5.1 RNN Model

- Embedding layer with 128-dimensional vectors
- One-layer RNN with 128 hidden units
- Fully connected dense output layer with Sigmoid activation

5.2 LSTM Model

- Embedding layer with 128-dimensional vectors
- One-layer RNN with 128 hidden units
- Fully connected dense output layer with Sigmoid activation

6. Evaluation Metrics

We evaluated the models using the following metrics:

- **Accuracy**
- **F1 Score**
- **Confusion Matrix**

7. Results and Analysis

At first I began with a standard approach. I formed y_train and y_val using the 'polarity' column and X_train, X_val using the 'review' column. But this was not that easy since the dataset contains 3.6 Million rows and it's not possible to load the whole dataset in Google Colab free version, which only provides 16 GB of RAM with a limited time of access(it gets reset in some hrs).

I tried a lot of things splitting the dataset into a fraction of 0.15 or 0.08 and then load them. While minimizing the dataset also not a good thing to do since the more proper dataset you have the better it is(Currently i am using the 0.08 of the original dataset).

It resulted in not so good numbers. Then i added the title and the review into a single column, and the numbers improved a little bit by doing this. As you can see yourself

```
Epoch 1/4
2250/2250 ━━━━━━━━━━━━━━━━━━━━ 568s 251ms/step - accuracy: 0.5695 - loss: 0.6671 - val_accuracy: 0.7735 - val_loss: 0.4904
Epoch 2/4
2250/2250 ━━━━━━━━━━━━━━━━━━━━ 613s 247ms/step - accuracy: 0.7852 - loss: 0.4735 - val_accuracy: 0.7794 - val_loss: 0.4643
Epoch 3/4
2250/2250 ━━━━━━━━━━━━━━━━━━━━ 547s 241ms/step - accuracy: 0.7911 - loss: 0.4592 - val_accuracy: 0.7761 - val_loss: 0.4799
Epoch 4/4
2250/2250 ━━━━━━━━━━━━━━━━━━━━ 562s 241ms/step - accuracy: 0.8017 - loss: 0.4429 - val_accuracy: 0.7746 - val_loss: 0.4819

<keras.src.callbacks.history.History at 0x7cc6ed5ffe10>
```

Figure 1: Training in RNN (Title+Review)

```
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 20s 20ms/step
              precision    recall  f1-score   support

           0       0.79      0.75      0.77     16000
           1       0.76      0.80      0.78     16000

    accuracy                           0.77     32000
   macro avg       0.78      0.77      0.77     32000
weighted avg       0.78      0.77      0.77     32000
```

Figure 2: Errors Metrics in RNN (Title+Review)

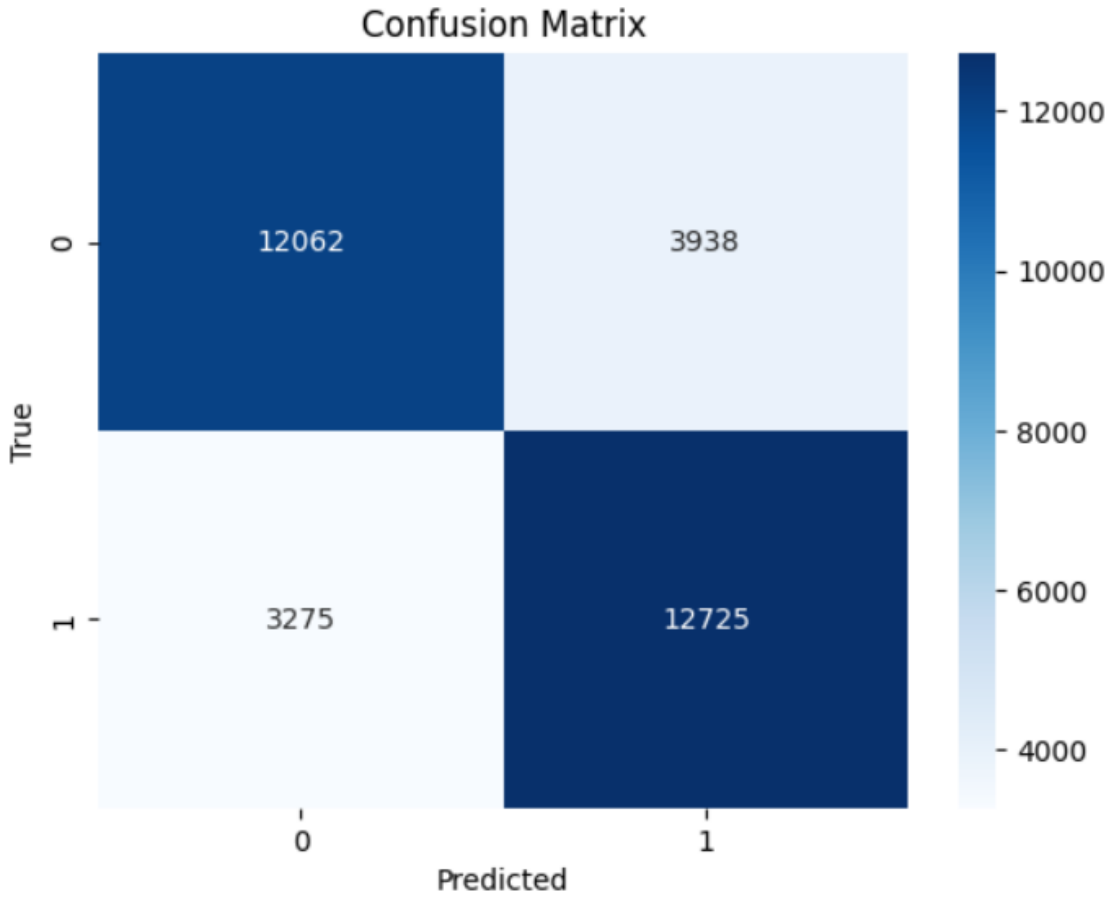

Figure 3: Confusion Matrix in RNN (Title+Review)

Now this doesn't look bad at all, i had my numbers between 0.75 to 0.80. But i wanted to experiment more with the data that i left and one more thing that is the title, i mean the 'title' only without the review paragraph. Why? Because whenever we write a review for the paragraph, we make sure that the user gets it whether to but the product or not just by reading the paragraph. So we basically write the summary of our long review and most of the time that summary will be in few words(5-9) and are made up of the common words. This is what i think and was the reason for me to experiment with the title only.

I removed the review column and loaded my current dataset again.

Now i calculated the number of unique words and the length of titles people have written for 90%,95% and 99% of the total of my current dataset. This was for choosing the number fitted for the vocab_size and Max_len to use in my embedding layer.

The results are the following:

Total unique words in corpus: 50k approx

90th percentile: 8.0, 95th percentile: 9.0, 99th percentile: 12.0

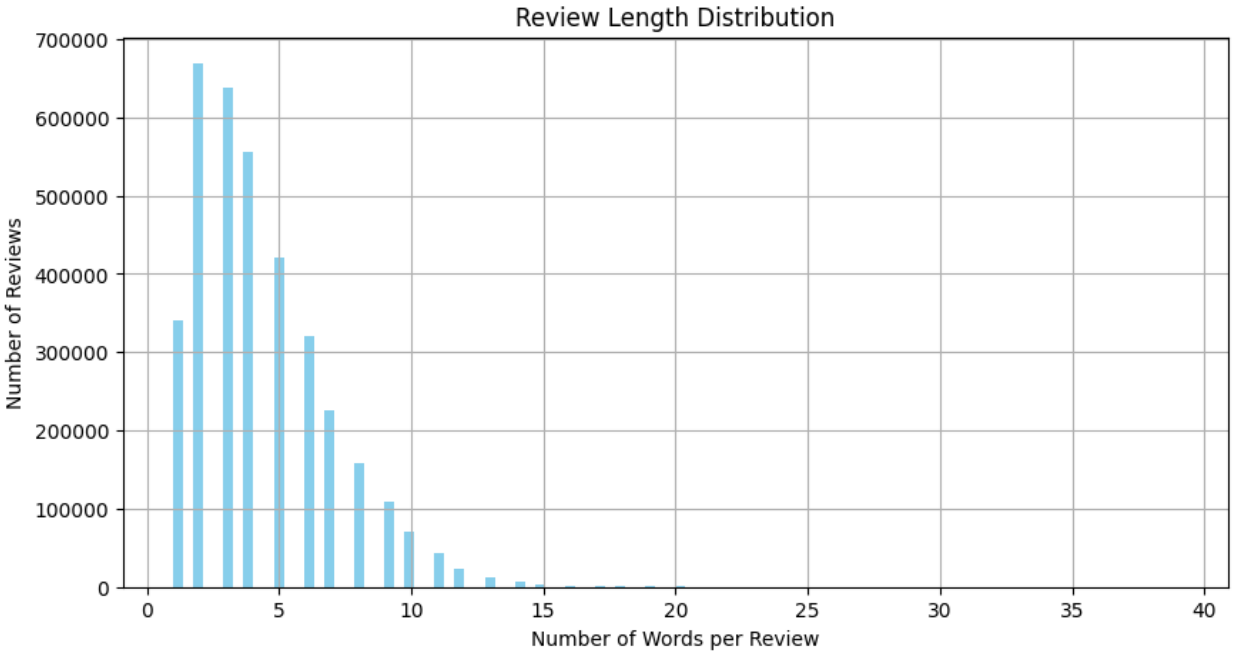

Figure 4: Review length distribution

I chose my vocab_size = 5000 and Max_len = 9.

These are the results only considering the 'title' as the input to my model.

```
Epoch 1/5
2250/2250 ━━━━━━━━━━━━━━━━━━━━ 47s 20ms/step - accuracy: 0.7939 - loss: 0.4302 - val_accuracy: 0.8261 - val_loss: 0.3769
Epoch 2/5
2250/2250 ━━━━━━━━━━━━━━━━━━━━ 81s 20ms/step - accuracy: 0.8399 - loss: 0.3462 - val_accuracy: 0.8333 - val_loss: 0.3504
Epoch 3/5
2250/2250 ━━━━━━━━━━━━━━━━━━━━ 44s 20ms/step - accuracy: 0.8523 - loss: 0.3181 - val_accuracy: 0.8293 - val_loss: 0.3611
Epoch 4/5
2250/2250 ━━━━━━━━━━━━━━━━━━━━ 82s 20ms/step - accuracy: 0.8650 - loss: 0.2984 - val_accuracy: 0.8333 - val_loss: 0.3551
Epoch 5/5
2250/2250 ━━━━━━━━━━━━━━━━━━━━ 82s 20ms/step - accuracy: 0.8754 - loss: 0.2796 - val_accuracy: 0.8352 - val_loss: 0.3713

<keras.src.callbacks.history.History at 0x78c829c092d0>
```

Figure 5: Training in RNN (Title)

```
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step
              precision    recall  f1-score   support

           0       0.83      0.84      0.84     16000
           1       0.84      0.83      0.83     16000

    accuracy                           0.84     32000
   macro avg       0.84      0.84      0.84     32000
weighted avg       0.84      0.84      0.84     32000
```

Figure 6: Errors Metrics in RNN (Title)

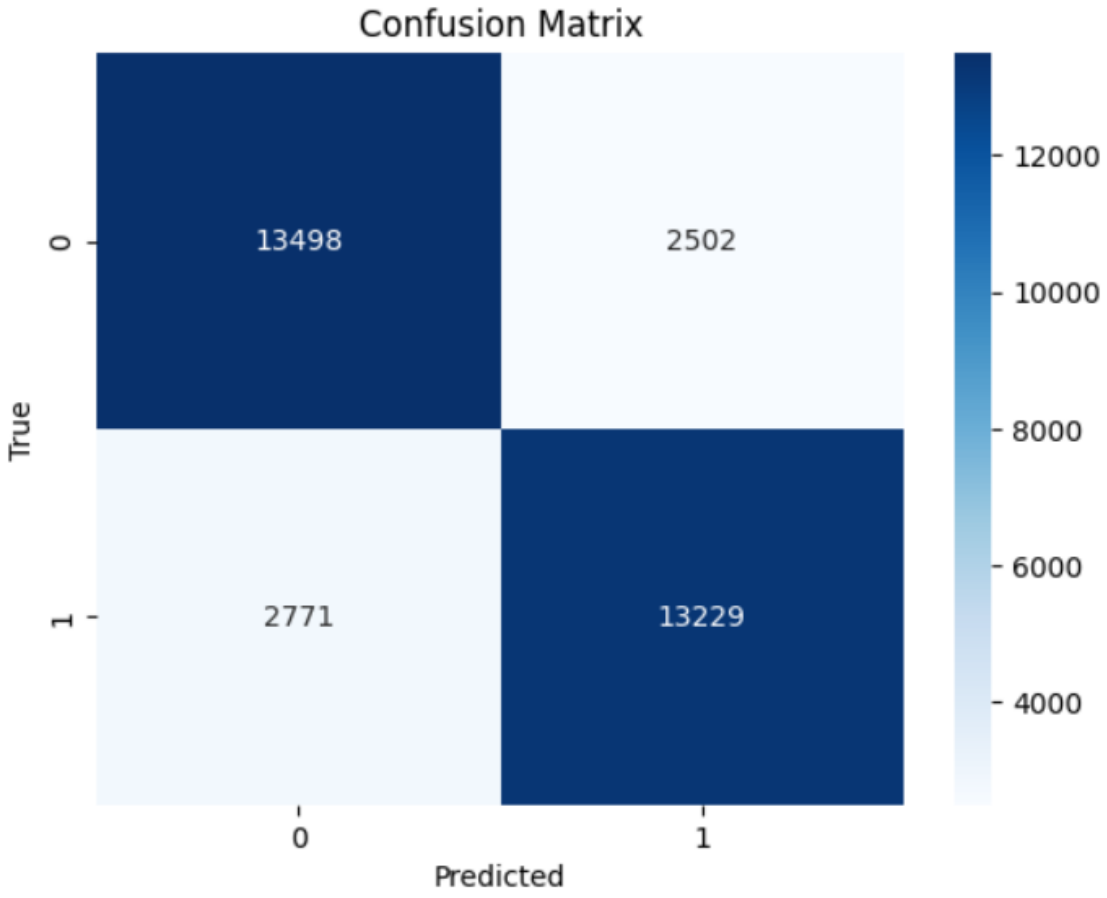

Figure 7: Confusion Matrix in RNN (Title)

The performance **improved significantly** and this proves my assumption about people writing their title. Now this doesn't only solves the numbers but also **saves the memory**. Before this the whole memory was getting consumed and now even if i load the whole original dataset (3.6 Million), it will use even **less than 1/4th of the Colab RAM (16 GB)**.

The last change i did to my model was i loaded the original dataset and used them all for my model, i increased my **vocab_size to 15000** and **Max_len to 16**.

The results are following for both RNN and LSTM Model:

```
Epoch 1/4
2009/2009 ━━━━━━━━━━━━━━━━━━━━ 383s 189ms/step - accuracy: 0.8338 - loss: 0.3586 - val_accuracy: 0.8665 - val_loss: 0.2950
Epoch 2/4
2009/2009 ━━━━━━━━━━━━━━━━━━━━ 387s 192ms/step - accuracy: 0.8705 - loss: 0.2878 - val_accuracy: 0.8691 - val_loss: 0.2928
Epoch 3/4
2009/2009 ━━━━━━━━━━━━━━━━━━━━ 379s 188ms/step - accuracy: 0.8797 - loss: 0.2708 - val_accuracy: 0.8719 - val_loss: 0.2856
Epoch 4/4
2009/2009 ━━━━━━━━━━━━━━━━━━━━ 388s 193ms/step - accuracy: 0.8862 - loss: 0.2583 - val_accuracy: 0.8712 - val_loss: 0.2858

<keras.src.callbacks.history.History at 0x7dbd7d5f2990>
```

Figure 8: Training in RNN (Title)

```
12500/12500 ━━━━━━━━━━━━━━━━━━━━ 44s 3ms/step
              precision    recall  f1-score   support

           0       0.90      0.84      0.87    200000
           1       0.85      0.91      0.88    200000

    accuracy                           0.87    400000
   macro avg       0.87      0.87      0.87    400000
weighted avg       0.87      0.87      0.87    400000
```

Figure 9: Errors Metrics in RNN (Title)

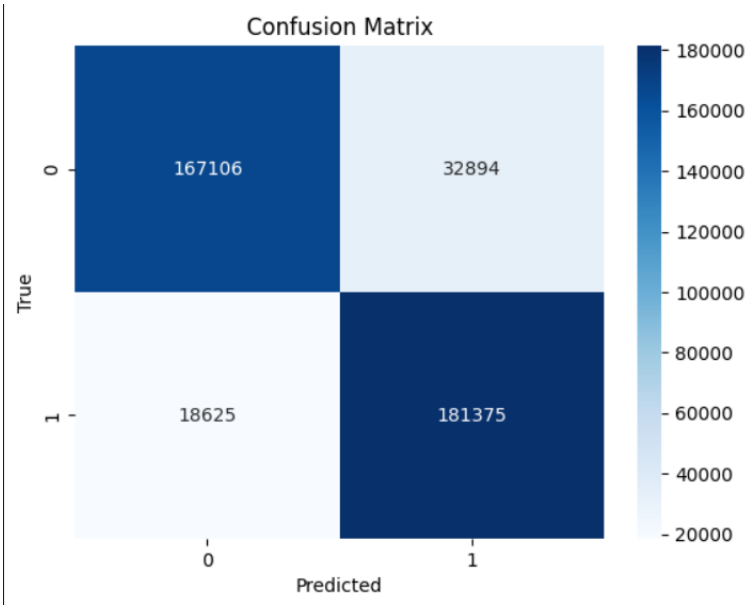

Figure 10: Confusion Matrix in RNN (Title)

```
Epoch 1/4
2009/2009 ━━━━━━━━━━━━━━━━━━━━ 1354s 672ms/step - accuracy: 0.8250 - loss: 0.3634 - val_accuracy: 0.8648 - val_loss: 0.2980
Epoch 2/4
2009/2009 ━━━━━━━━━━━━━━━━━━━━ 1415s 678ms/step - accuracy: 0.8706 - loss: 0.2860 - val_accuracy: 0.8727 - val_loss: 0.2838
Epoch 3/4
2009/2009 ━━━━━━━━━━━━━━━━━━━━ 1381s 668ms/step - accuracy: 0.8794 - loss: 0.2699 - val_accuracy: 0.8761 - val_loss: 0.2761
Epoch 4/4
2009/2009 ━━━━━━━━━━━━━━━━━━━━ 1339s 666ms/step - accuracy: 0.8857 - loss: 0.2574 - val_accuracy: 0.8772 - val_loss: 0.2753

<keras.src.callbacks.history.History at 0x7e7e9d120dd0>
```

Figure 11: Training in LSTM (Title)

```
12500/12500 ━━━━━━━━━━━━━━━━━━━━ 131s 10ms/step
              precision    recall  f1-score   support

           0       0.89      0.87      0.88    200000
           1       0.87      0.89      0.88    200000

    accuracy                           0.88    400000
   macro avg       0.88      0.88      0.88    400000
weighted avg       0.88      0.88      0.88    400000
```

Figure 12: Errors Metrics in LSTM (Title)

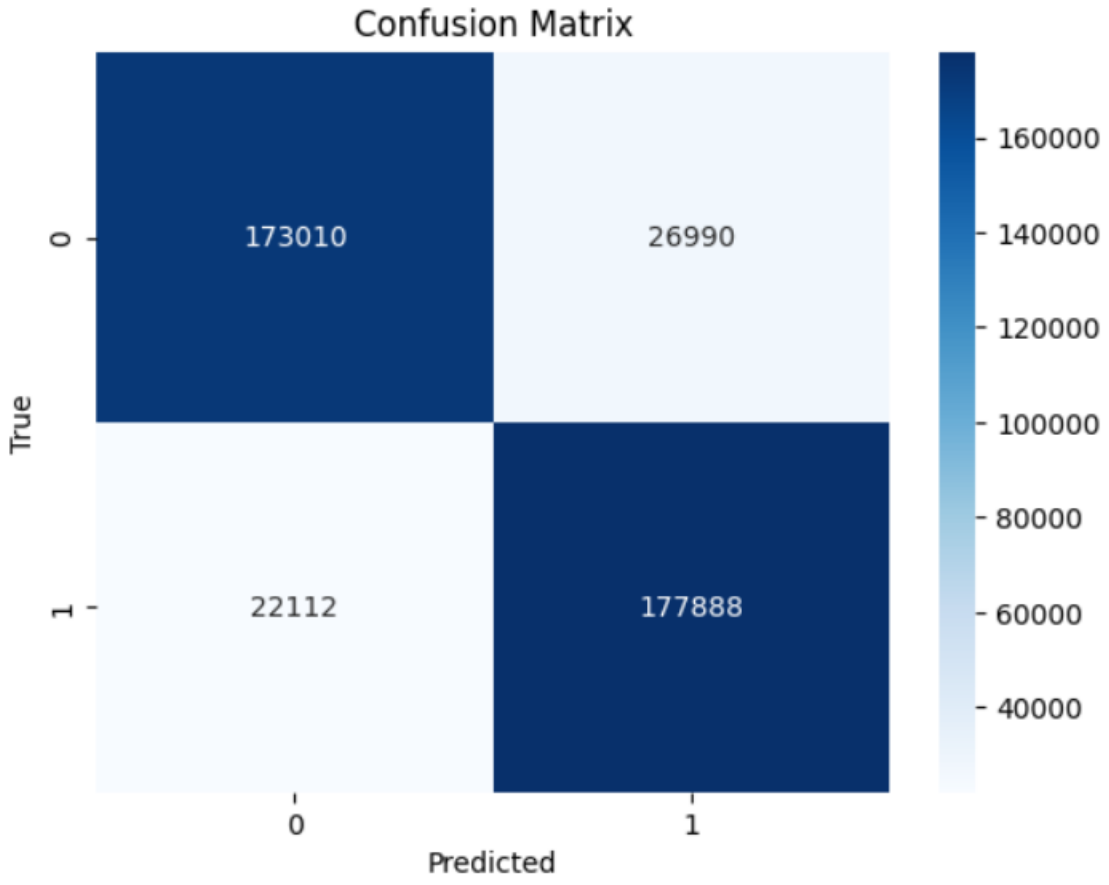

Figure 13: Confusion Matrix in LSTM (Title)

8. Minimal Clue Challenge

How can my model be fooled:

- Since my model only takes the title as input, a title like this " The game started pretty lame and the storyline was not that good in the first mission though, but after all the missions are amazing. It's an OG game!! Since my Max_len of a review is 16 so it will ignore the later words which might actually represent the real review.
- If someone uses the title as "Never seen anything like this" in a positive way and in the review they applaud the product. My model will classify them as negative.
- If only used emojis and number like 100% in title

Minimal Clue Challenge Example: If the user uses the words used with a negative review then there is a possibility my model will misinterpret a positive review as a negative one.

- Predicted: 0, Actual: 1, Text: SOY UN APASIONADO DEL BOX
  Spanish (not English) → tokenizer mapped most words to ¡OOV¿
- Predicted: 0, Actual: 1, Text: They'd watch it nonstop if I'd let them
  Sarcasm or subtle praise → lacks explicit sentiment words
- Predicted: 0, Actual: 1, Text: Great Beginning...poor ending
  Just the ending didn't as planned doesn't make it bad, the model marked it negative
- Predicted: 0, Actual: 1, Text: No another grill like this...
  Model might have learned that phrases starting with "no" often lead to negative labels (like "no value", "not good", "no support"), it'll assumed negative here too.
- Predicted: 0, Actual: 1, Text: There is no way anyone with a soul could ever give this a negative review....
  I guess my model has a negative soul, anyways Heavy use of negation + sarcasm
- Predicted: 0, Actual: 1, Text: Creepy
  Ambiguous tone — could be good for horror films/games like product, but usually negative
- Predicted: 1, Actual: 0, Text: WOW!
  Could be sarcastic or surprised — no polarity word

9. Conclusion

Improvements that might i should consider :

1. Add more training data with:
    - Sarcasm
    - Short but strong phrases
    - Double negatives
2. Use pretrained embeddings (GloVe or Word2Vec) to improve rare word handling.
3. Add punctuation and emoji awareness back to tokenizer.