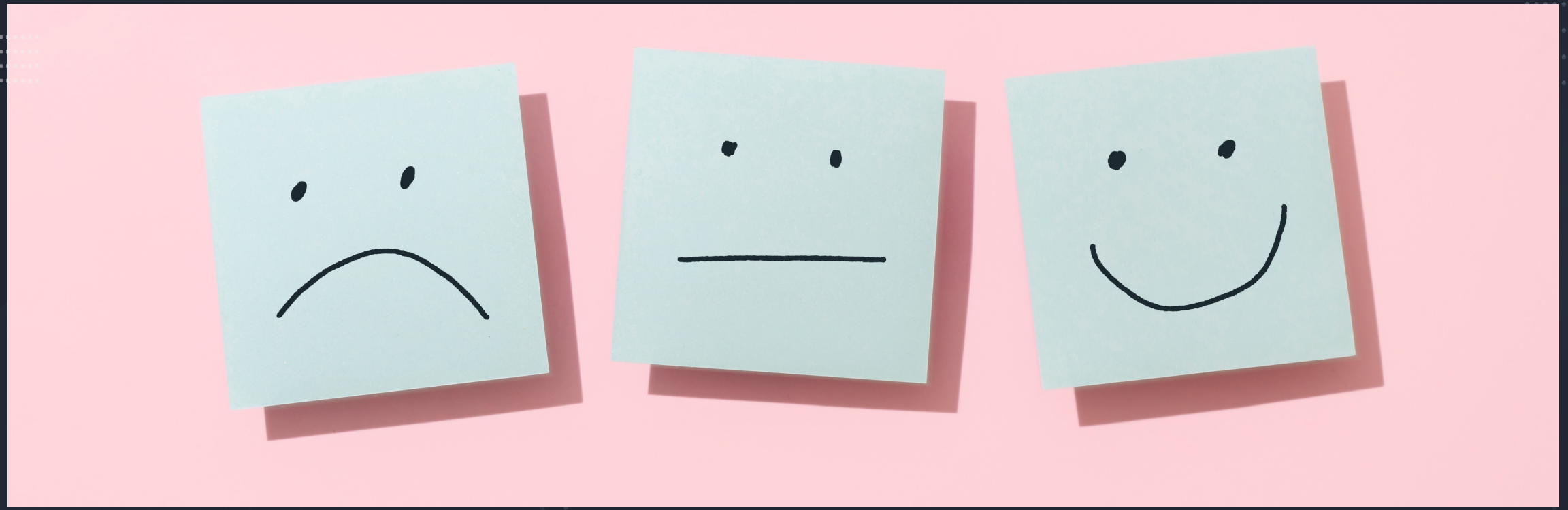
Twitter Sentiment Analysis: Classification Vs Deep Learning Approach

Himani Kaushik



Need:

- Implement a sentiment analysis model
  - Utilize tweets
  - Predict positive or negative sentiment
- Detect angry customers or negative emotions before escalation
- Select better performing model:
  - Naive Bayes
  - LSTM

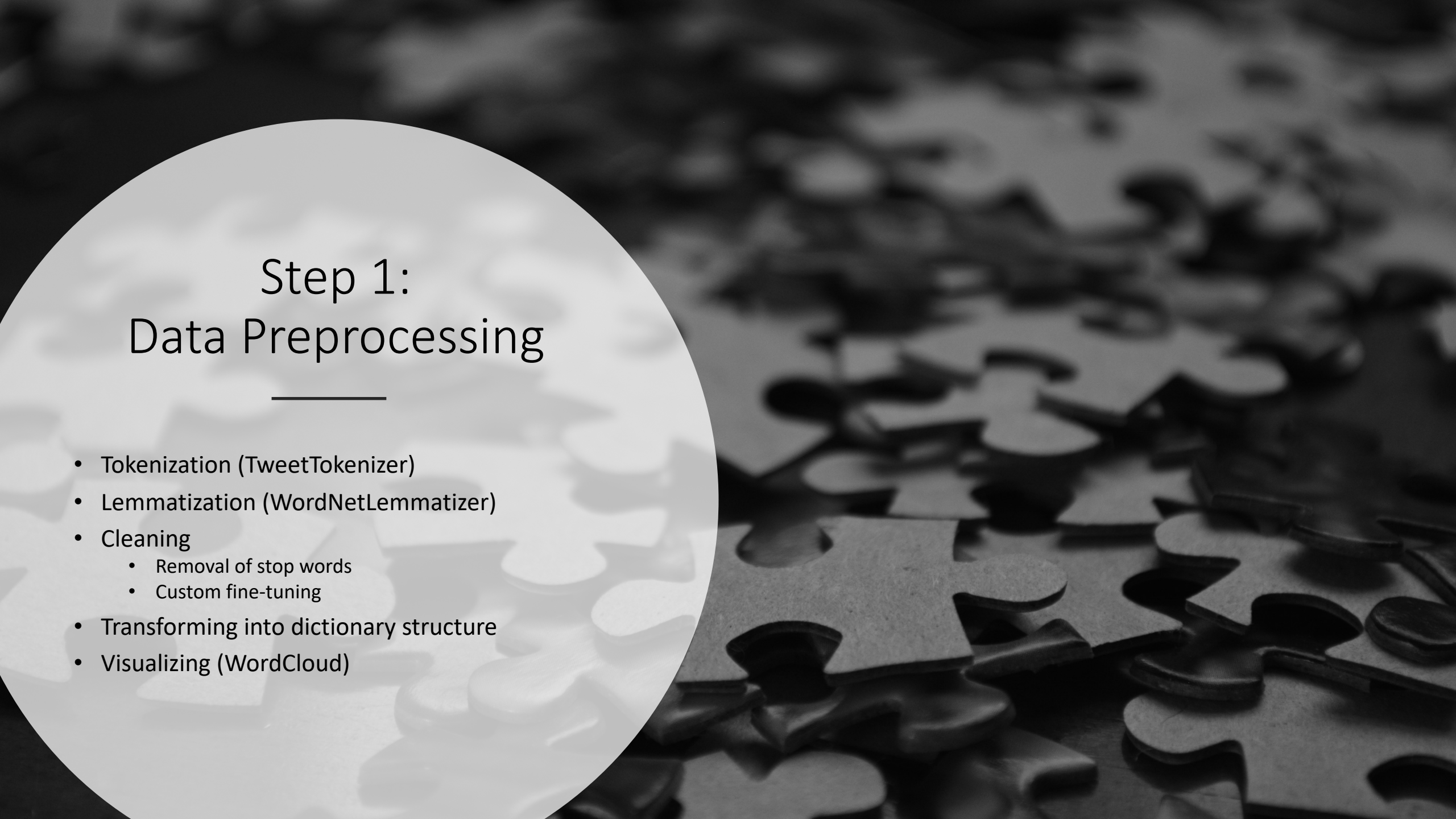
Data:

- 'Sentiment 140’ Kaggle dataset
- 1.6 million tweets
- Dataset trimmed to 1/4th
  - Data balance maintained

Tools:

- Pandas and Numpy
- Keras, NLTK, scikit-learn
- Matplotlib, Wordcloud

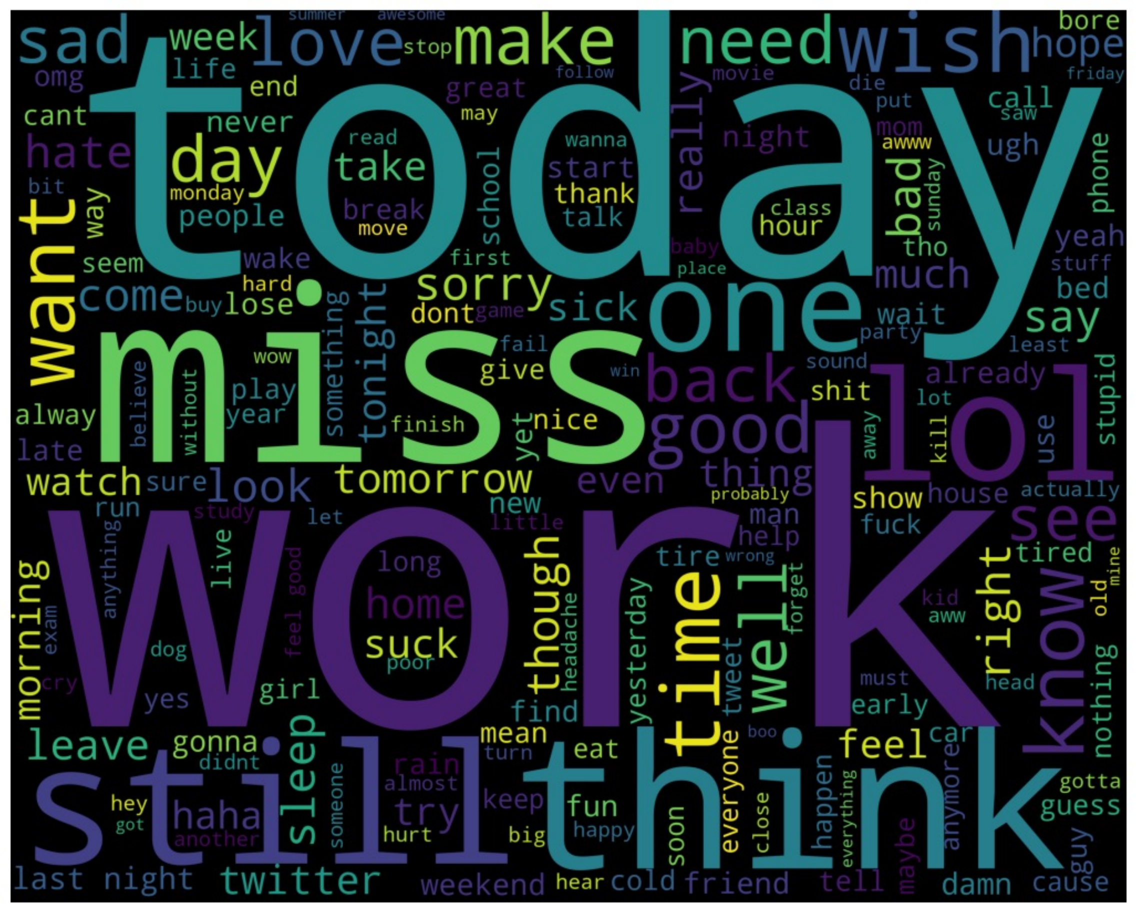
Methodology



Step 1:
Data Preprocessing

- Tokenization (TweetTokenizer)
- Lemmatization (WordNetLemmatizer)
- Cleaning
  - Removal of stop words
  - Custom fine-tuning
- Transforming into dictionary structure
- Visualizing (WordCloud)

Frequent Words: Positive Vs Negative

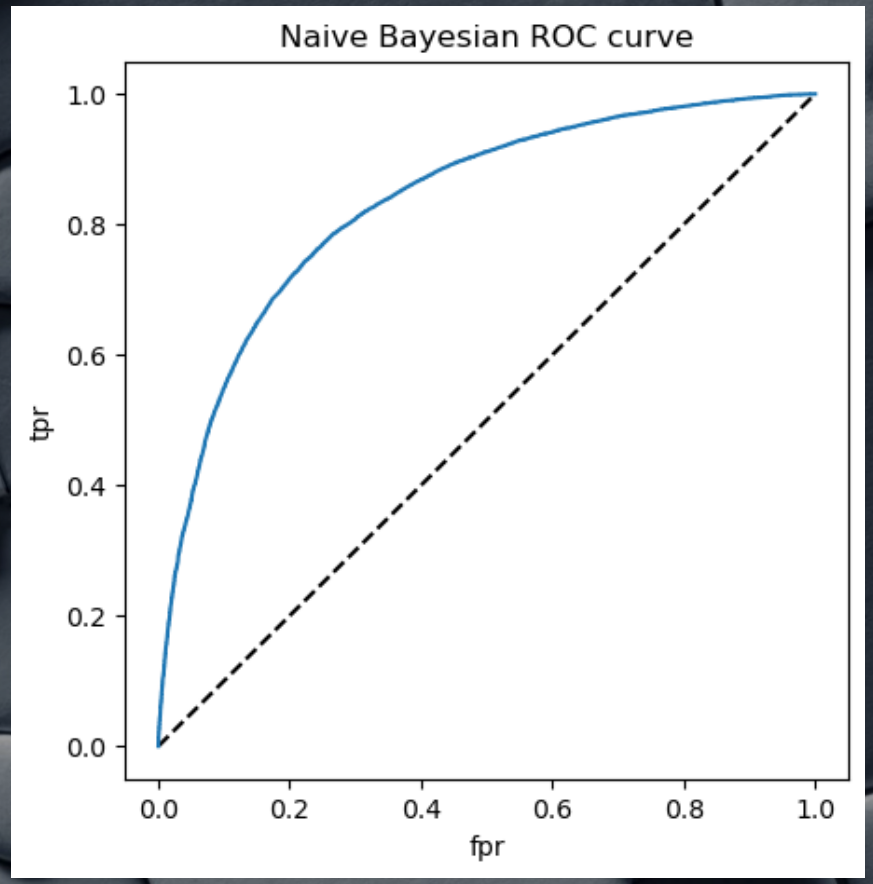


Step 2:
Naive Bayes Model

Results:

- Accuracy on train data: 0.81
- Accuracy on test data: 0.76
- AUC: 0.83

- *High volume of negative vs positive informative features*



Limitations :

- Evaluates at an independent word level
- Performs poorly for
  - negations and multi-words constructs
  - sarcasm
- Fails at generalizing

```
custom_tweet = "Good"

custom_tokens = remove_noise(tk.tokenize(custom_tweet))

print(classifier.classify(dict([token, True] for token in custom_tokens)))
```

```
1
```

```
custom_tweet = "Not good"

custom_tokens = remove_noise(tk.tokenize(custom_tweet))

print(classifier.classify(dict([token, True] for token in custom_tokens)))
```

```
1
```

```
custom_tweet = "The reward for good work is more work!"

custom_tokens = remove_noise(tk.tokenize(custom_tweet))

print(classifier.classify(dict([token, True] for token in custom_tokens)))
```

```
1
```

Step 3: LSTM

- Data Transformation
- LSTM Model
- Regularization
- Results
- Predictions

Data Transformation:

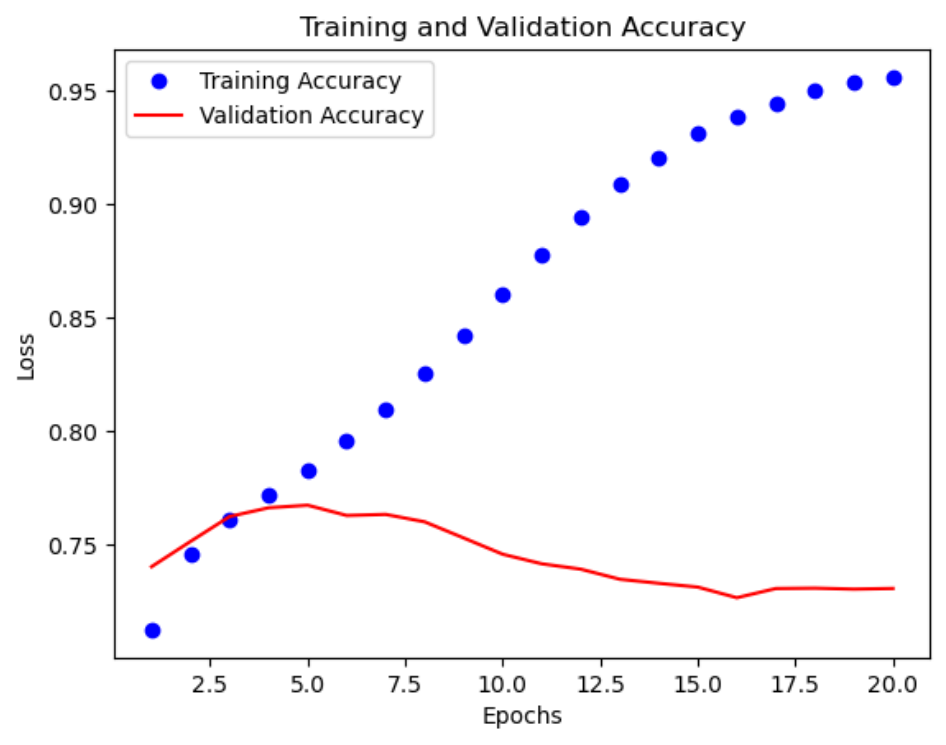
- Word Embeddings using GloVe
- Data Padding

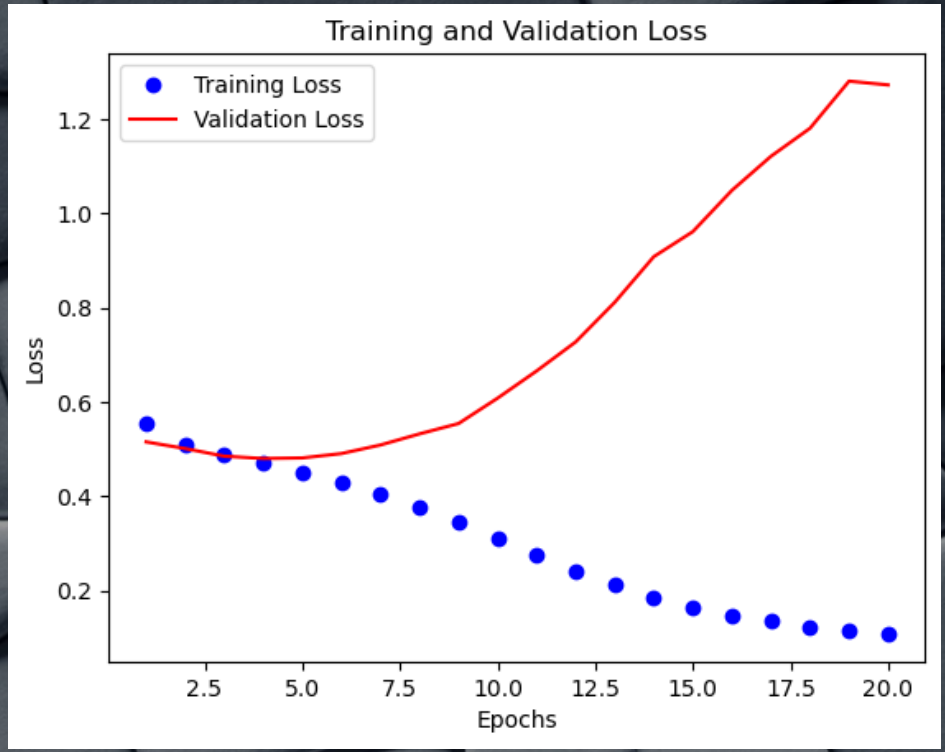
LSTM Model:

- Sequential model
  - Embedding layer
  - Pair of Bidirectional LSTMs
  - Sigmoid layer
  - Binary cross-entropy loss function
  - ‘Adam’ optimizer
  - Accuracy metric
- Training Model
  - Batch size of 20 and epoch size of 20

Initial Results:

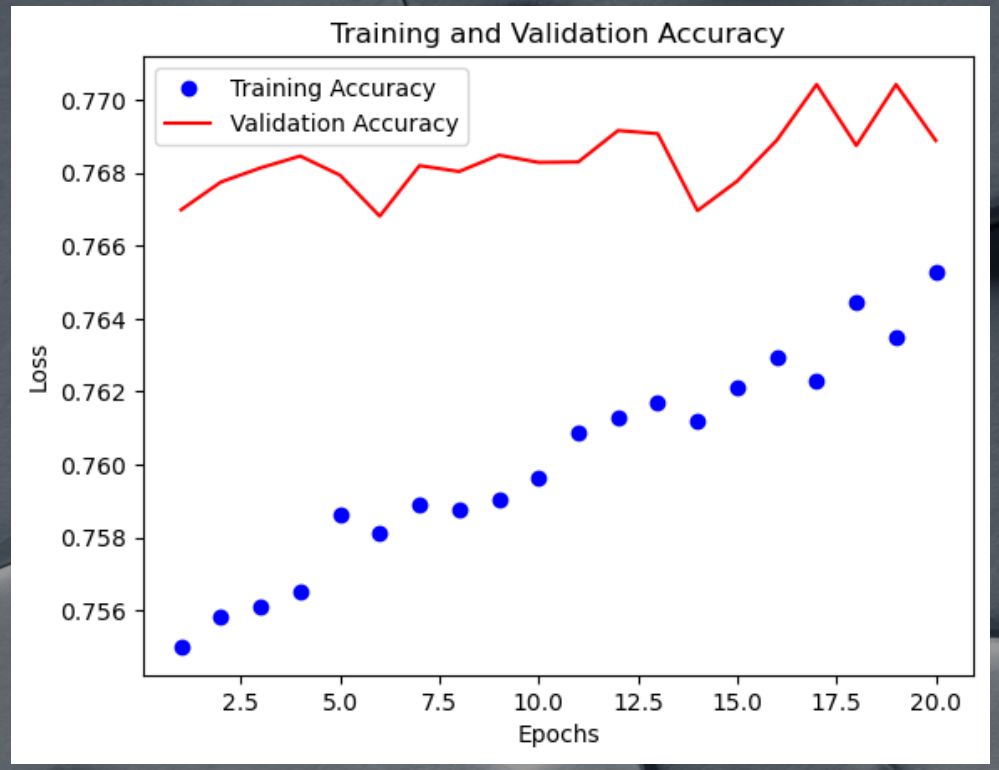
Over-fitting

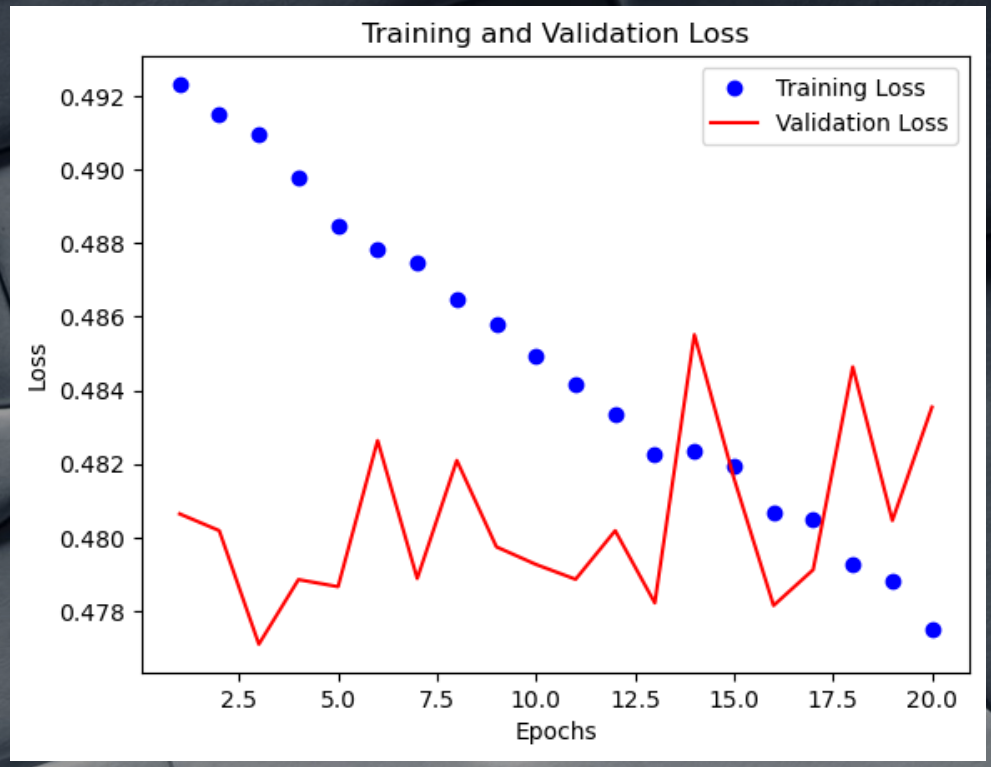




Regularization:

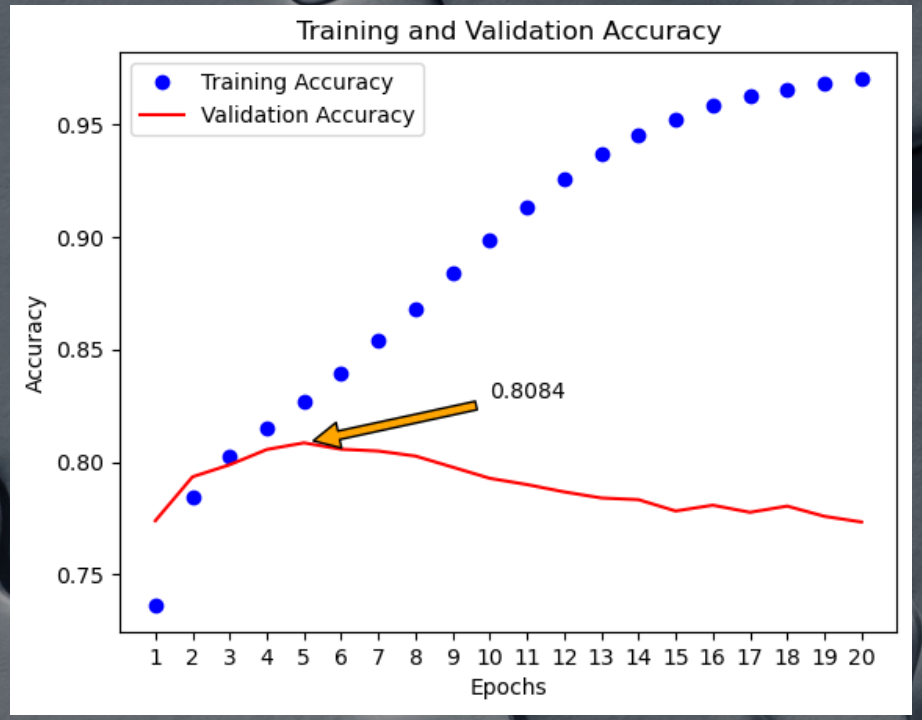
- Dropout
  - LSTM model with a dropout rate of 40%
  - Two rounds of training
  - Tiny improvement after adding dropout

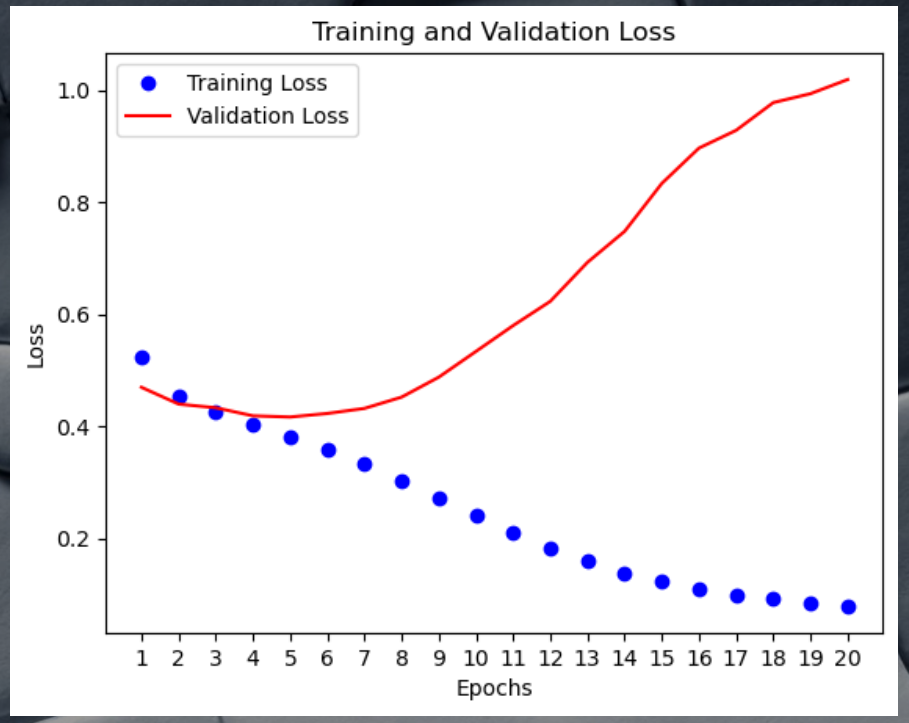




Regularization (contd):

- Inspecting Unknown Words
  - 198, 378 unknown words
- Further data cleaning
  - 119,802 unknown words
- Model built and trained on cleaned data
  - 81% validation accuracy on the 6th epoch





Predictions:

Model can understand relationship between words

```
predict_custom_tweet_sentiment("I'm happy you're here!")
```

```
1.0
```

```
predict_custom_tweet_sentiment("I'm not happy you're here!")
```

```
4.628090664482443e-06
```

```
predict_custom_tweet_sentiment("I disliked his attitude...")
```

```
0.0022062957286834717
```

Model can predict sentiment of unseen words

```
predict_custom_tweet_sentiment("I'm infatuated with you")
```

```
0.7147030830383301
```

```
(negative_words + positive_words).count('love')
```

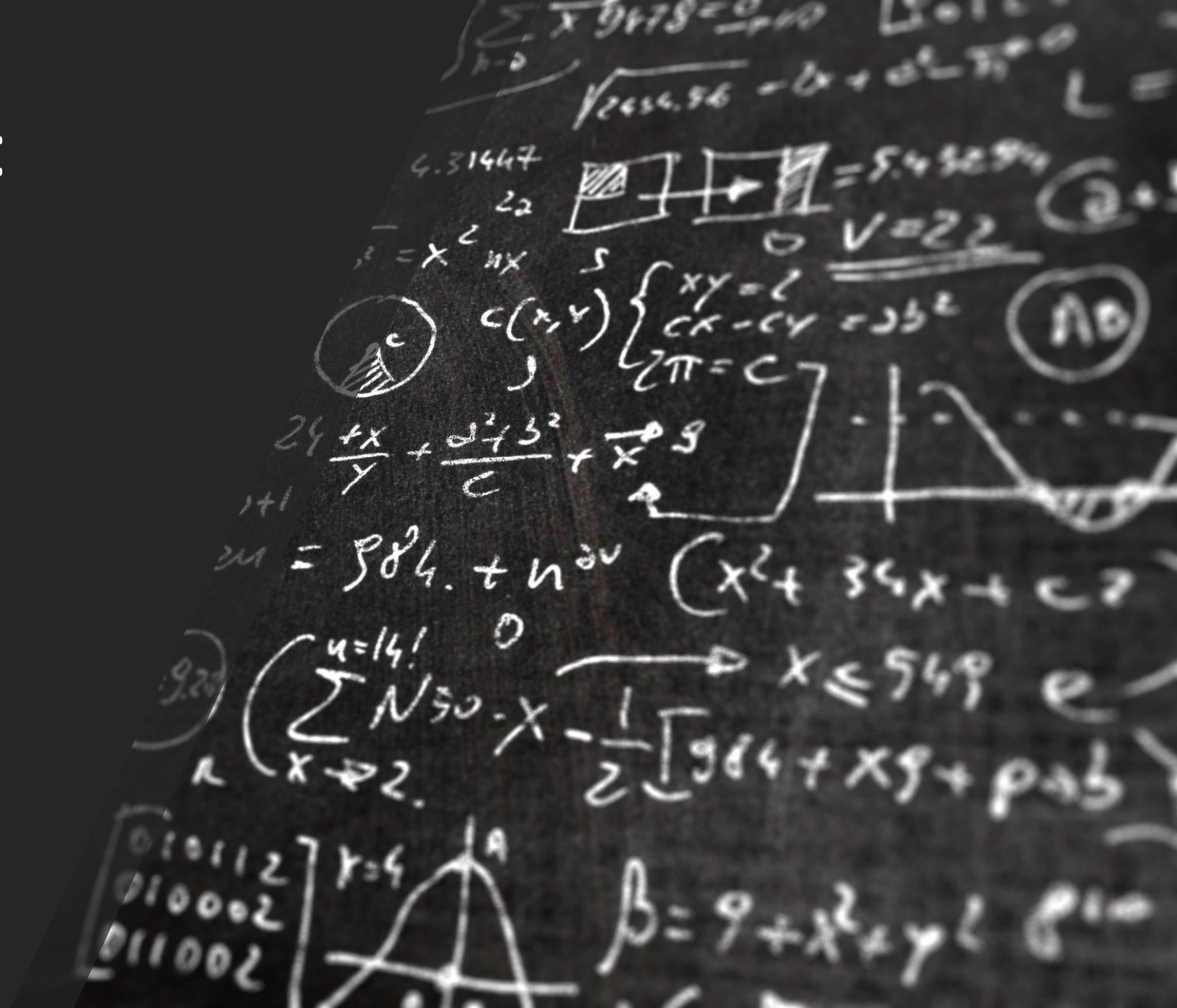
```
17636
```

```
(negative_words + positive_words).count('infatuated')
```

```
0
```

Further work:

- Intensive data cleaning
- Accurate re-labelling
  - After inspecting wrongly predicted data
- Create neutral class
- Word embeddings
- Model architectures




Thank you