Shahid Beheshti University
Data Science (2023-2024)

Final report

Author 1
Department of Computer Science
Shahid Beheshti University
pooriawmlk@gmail.com

Author 2
Department of Computer Science
Shahid Beheshti University
farimaherashidiii@gmail.com

Abstract

This dataset contains the data of multiple News headlines and we are required to build a model to predict if the headline is sarcastic or non-sarcastic.

Introduction

This project is **Sarcasm Detection**. We have a dataset which contains 3 columns and 28619 rows. The columns are:

- **Is_sarcastic:** this is the target which is **0** if the headline is not sarcastic, otherwise it is **1**.
- **Headline:** this column contains a headline, made from a few words which may or may not be sarcastic.
- **Article_link:** a link to the headline for reference.

The dataset is in **json** format.

So, the data seems pretty simple but actually it comes with many complex problems.

With limited data, the models may struggle with sparsity issues, leading to poor model performance.

Another problem that we may encounter is overfitting. Overfitting is a common concern, especially when the amount of training data is small. Models may memorize the limited examples rather than learning generalizable patterns.

NLP models, especially deep learning models, rely on complex feature representation. So, with this much data the NLP models may not be able to learn complex features.

With this limited data, we may not be able to make a practical model that works for every sentence that we give it. In other words it may struggle with new problems outside of the given data.

So, with all that being said, we need to test different models to reach an optimal solution. With this approach we can make the right decision about this problem.

The models we are going to use are as follows:

- Naive Bayes
- Logistic Regression
- Different LSTM architectures

Related Work/Background

Model 1:

This model has an interesting structure since it uses two functions to work as generators in the training loop to provide batches of data to the model during training.
It is using the "**SentenceTransformer"** library to encode text data, while the LSTM-based model involves padding or truncating sequences and converting them into numerical representations of the data.
This model is a sequential neural network.
This is the model's structure:

1. An embedding layer that Converts integer-encoded vocabulary indices into dense vectors.
2. Long Short-Term Memory (LSTM) layer to capture contextual information from past experiences and future experiences.
3. A dense layer with 1000 neurons and ReLU activation for non-linearity.
4. Dropout layer to reduce overfitting.
5. dense layer with 2 neurons and softmax activation for binary classification.

This model works quite well and it also reaches 98 percent accuracy for train data and 94 percent for the test data.

Model 2:

Another model which may seem interesting is the following model which uses tokenization to prepare the data for the training phase.
It also uses early stopping to prevent overfitting. This is the model's structure:

1. An embedding layer which Converts integer-encoded words into dense vectors of fixed size.
2. Long Short-Term Memory (LSTM) layer to capture sequential patterns in the input data.
3. a dense layer with 25 neurons and ReLU activation for non-linearity.
4. A dropout layer to reduce overfitting.
5. A dense layer with 1 neuron and sigmoid activation for binary classification.

Surprisingly, this model acts poorly and only reaches 52 percent accuracy. Many of the neural network models have high potential but a crucial factor is hyperparameter tuning and an effective architecture.

So, with that being said, we need to build a model which has an optimal structure and we need to preprocess the data to achieve better results.

Proposed methods

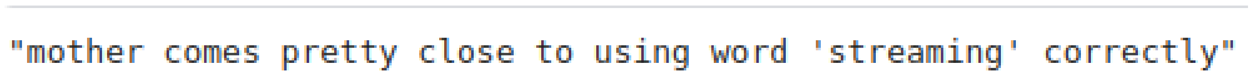
First we need to prepare the data and understand how the data works.

Preprocessing

NLP models are very tricky to work with. There are unlimited ways to approach them but only a few of them work.

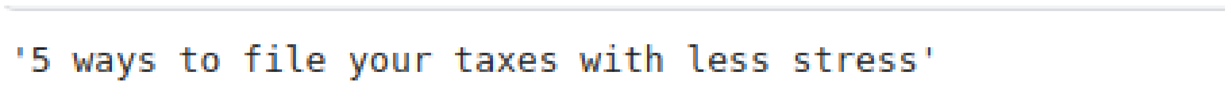
First, let’s see a few examples of the data.

The below sentence is sarcastic which tries to say that parents usually do not know the new words and figure of speeches.



```
"mother comes pretty close to using word 'streaming' correctly"
```

On the other hand, the below sentence is not sarcastic since it is pretty clear about its purpose.



```
'5 ways to file your taxes with less stress'
```

stop words are words that are commonly used in a language but generally don't contribute much to the meaning of a sentence. They are often removed from texts during the preprocessing phase because they can introduce noise and don't carry significant information. For example "the," "and," "is," "in," and "to."

For this data, we do as follows:

1. Extracting the text from all rows. There may be some anomalies so we use html parser to make sure all the current data is text.
2. Removing \[[^]]*\] from the text since they do not contribute to data and increase sparsity.
3. Removing links from the data.
4. Splitting the data and putting each word next to each other.

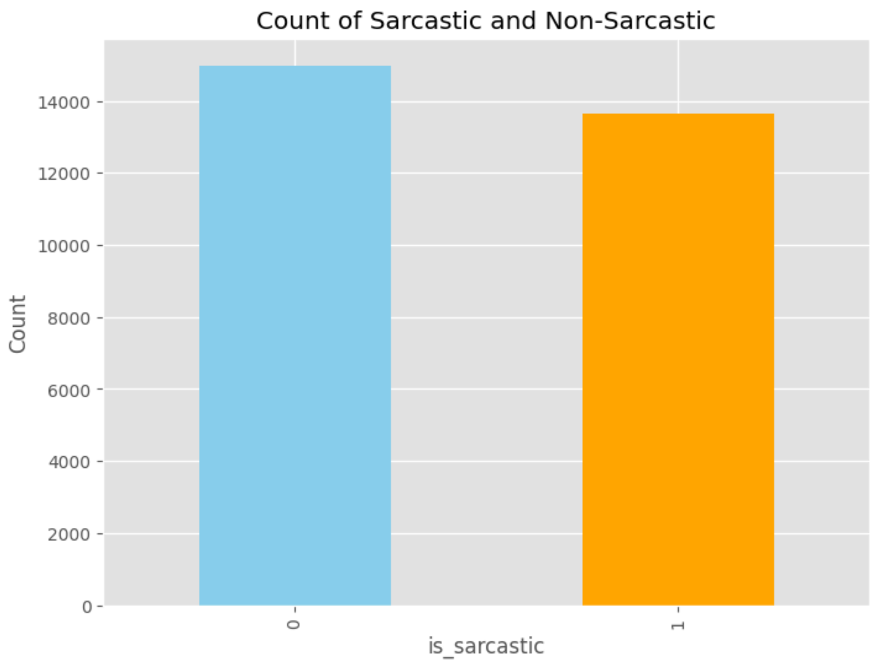
By doing these steps we have cleaned the data and it is ready to use.

We do not have any missing values so no further preprocessing is needed for now. although we need much more processing before training the data and feeding it to the model. But we'll get there.

Data Analysis

The data has a very simple structure. Only two columns are needed to be analyzed. So, we have a simple job here.

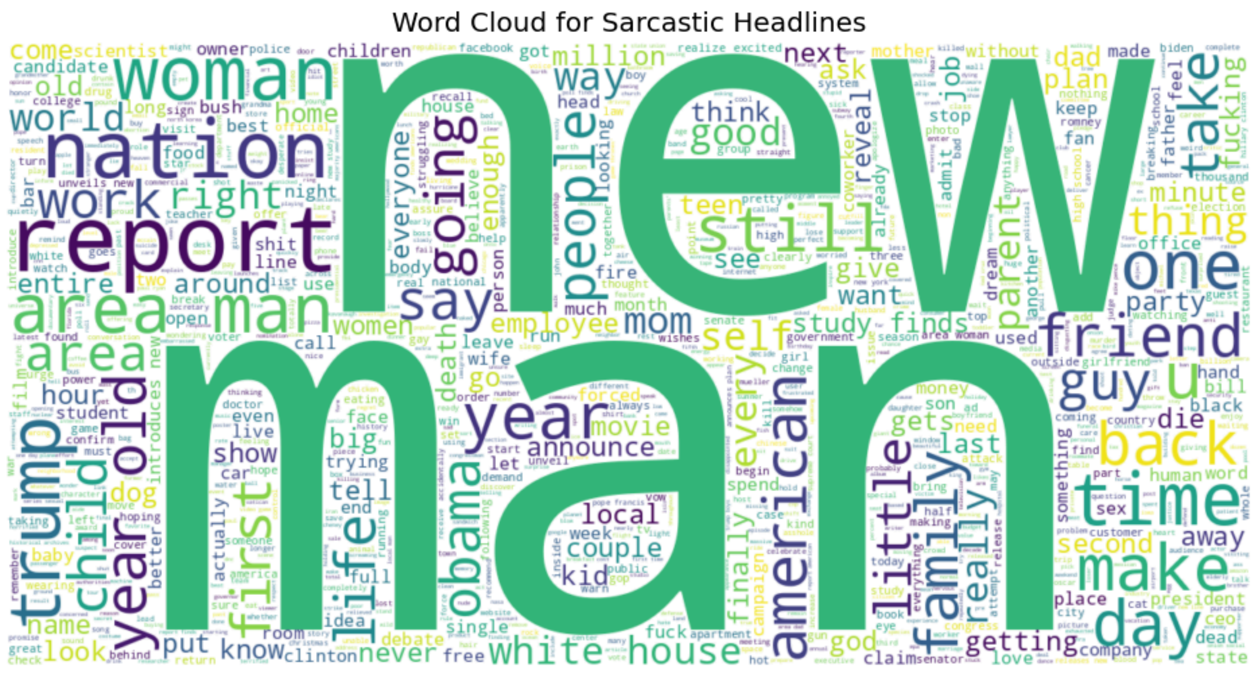
First let's have a look at the **is_sarcastic** column. The below diagram shows the barplot for this column.



According to this plot, the data is well distributed and the two classes have about the same number of rows.

To be more precise, we have 14985 rows from non-sarcastic class and 13634 rows from sarcastic class.

We have cleaned the data before. Now let’s learn more about the data. The below plot shows the word cloud plot for sarcastic words:

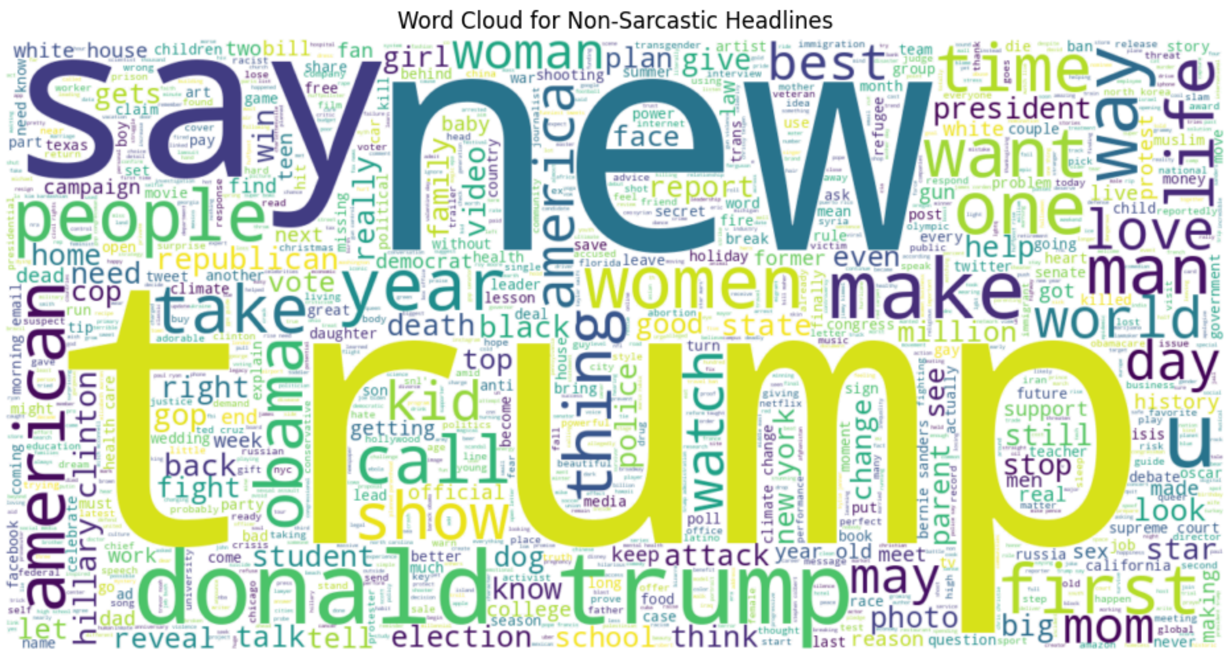


It is clear that some words are by far more frequent than others. We need to check if this is the case in non-sarcastic headlines and if so which words are more common in that class? We need to check this matter with the same plot but for non-sarcastic headlines.

According to this plot top three words are:

- New
- Man
- report

So, with that being said, let’s see about non-sarcastic features. The below diagram shows the word cloud plot for non-sarcastic features.



It seems that more common words are different in this class.

According to the results **man** is very common in sarcastic headlines and **trump** is common in non-sarcastic headlines.

And the word **New** is common in both classes.

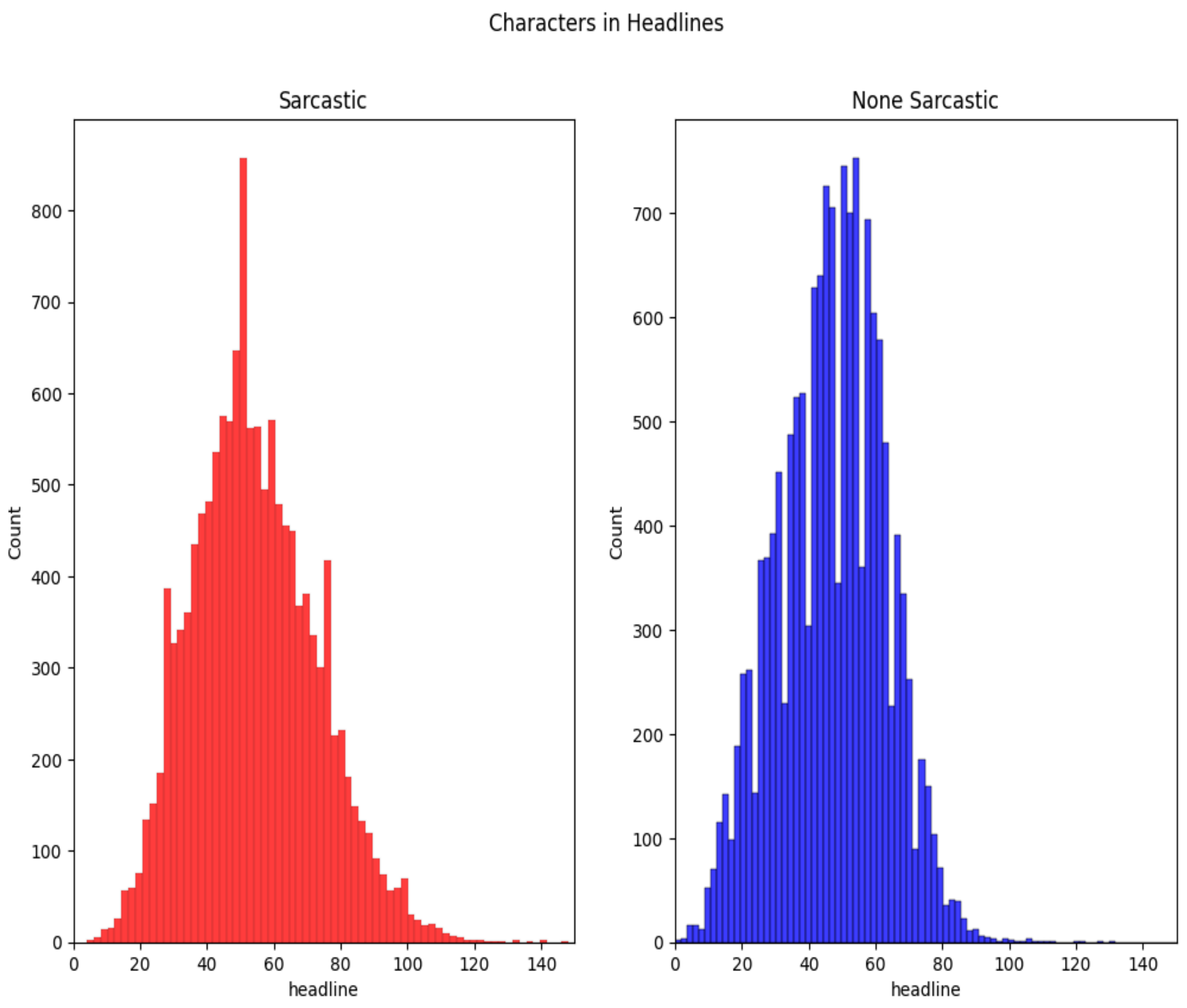
Top three words in non-sarcastic headlines are:

- Trump
- New
- say

One of the biggest differences between these two classes is their content. But can we find a pattern in their length?

Let's see the number of characters in each class. Maybe they have a difference in their length.

The below diagram shows the histogram for number of characters in each class:



The number of characters is limited to 150 since there are some extreme outliers that affect the plot.

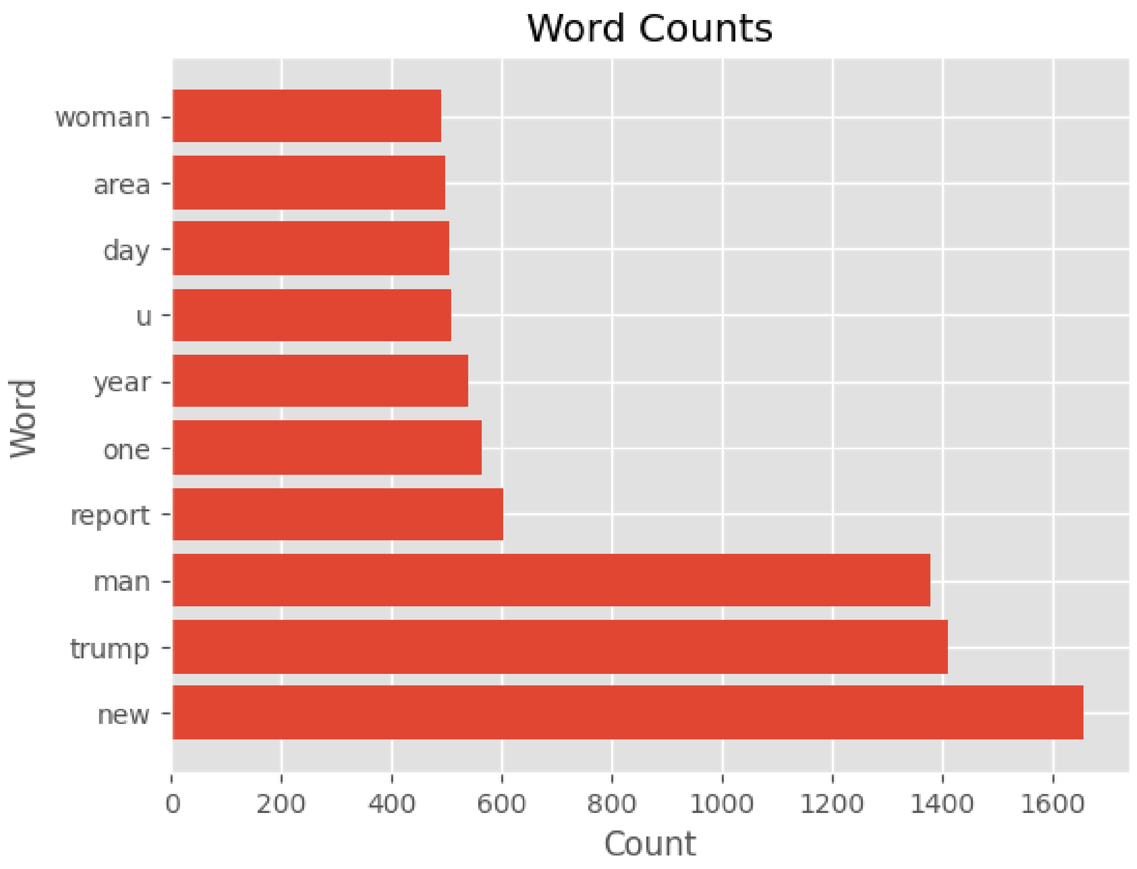
According to this plot, these two classes have almost the same distribution and we can not make a practical decision based on their number of characters.

Another thing is that most of the headlines are between 20 and 80 characters in both classes.

Currently, the number of unique words in our dataset is around 30 thousand. We may need to use PCA to reduce the dimension of our dataset.

The common words in both datasets is as follows:

```
[('new', 1658),
 ('trump', 1412),
 ('man', 1379),
 ('report', 605),
 ('one', 566),
 ('year', 541),
 ('u', 509),
 ('day', 507),
 ('area', 500),
 ('woman', 493)]
```



The word **New** has the highest number with 1658.

Methods

Logistic Regression

First, let’s start with a simple model. Text data is sparse, and logistic regression can handle sparse data well. Since this problem is a binary classification problem, we can use logistic regression.

Naive Bayes

Naive Bayes is a probabilistic model that calculates the probability of a given instance belonging to a particular class based on the occurrence of features.

Naive Bayes is computationally efficient and is particularly useful when dealing with large datasets.

For preprocessing, first we convert the data into arrays and then, apply **Countvectorizer** to it.

Countvectorizer transforms a collection of text documents into a matrix of token counts. The output is a sparse matrix where each row corresponds to a document, and each column corresponds to a word. The matrix contains the counts of each word in the respective documents.

For the model, we use **Bernoulli Naive Bayes**. It is Suited for binary data, where features represent the presence or absence of certain characteristics. It is also better for text classification problems compared to **Naive Bayes**.

LSTM - Model 1

Now, we are going to build a neural network model to make a stronger and more practical model. The first model is a Recurrent neural network (RNN) model built by the Keras library.

This is the model’s structure:

1. An embedding layer that converts integer-encoded words into dense vectors.
2. An LSTM layer for capturing sequential patterns.
3. Another LSTM layer.
4. Two dense layers for further processing and the final binary classification output.

To prepare the data for the training phase and feed it to the model, we use **Tokenizer** for this purpose.

Tokenization enables the creation of numerical representations for words, which is essential for training machine learning models on text data.

LSTM - Model 2

Since the previous model wasn't so promising, we take a different approach and change our structure a bit.

Here is another structure for the model:

1. The first layer is an Embedding layer to map each word in the dataset to a high-dimensional vector.
2. The second layer is a Bidirectional LSTM or Long Short-Term Memory layer. Bidirectional LSTMs process the input sequence in both forward and backward directions, capturing information from both past and future states. This will help the model to better understand important features.
3. The third layer is the Flatten layer. It is used to flatten the output from the Bidirectional LSTM layer into a one-dimensional array. It prepares the data to be passed to a fully connected layer.

4. This layer is the Dropout layer which randomly sets a fraction of input units to zero at each update during training time. This will help to prevent overfitting since the chance of overfitting is high.

5. The fifth layer is a dense layer with ReLU activation function.

6. The final layer is a Dense layer with a Sigmoid activation function. The output represents the probability of the input belonging to the positive class.

Again, we use **Tokenization** to prepare the data for the training phase and feed it to the model.

LSTM - Model 3

We have made good progress so far. Now, let’s make another model to see if we can make more powerful models and reduce overfitting.

This is the third model’s structure:

1. The first layer is an Embedding layer. This layer is used to convert integer-encoded vocabulary indices into dense vectors of fixed size.

2. Now, for the second layer, we use a pooling layer. We use it to reduce the dimensionality of the sequence data. It calculates the average value for each feature across all time steps.

3. For the third layer, we use a Dense layer with ReLU activation function. We use ReLU to increase non-linearity.

4. The final layer is another Dense layer with a Sigmoid activation function. We use this for binary classification.

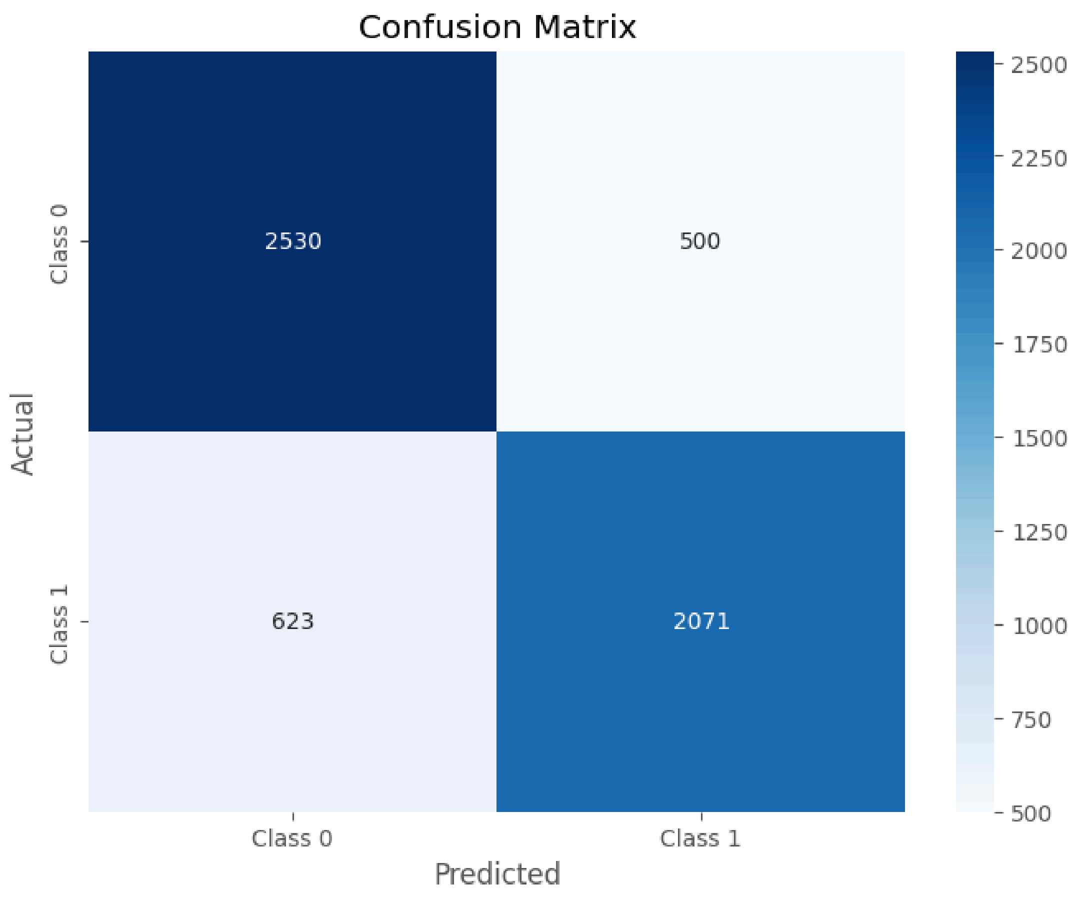
Now, we use **Tokenization** to feed it to the model.

Results

Logistic Regression

This model ran for less than a minute and the accuracy of the model is about 80 percent which is very impressive.

The below diagram shows the confusion matrix for the test data:



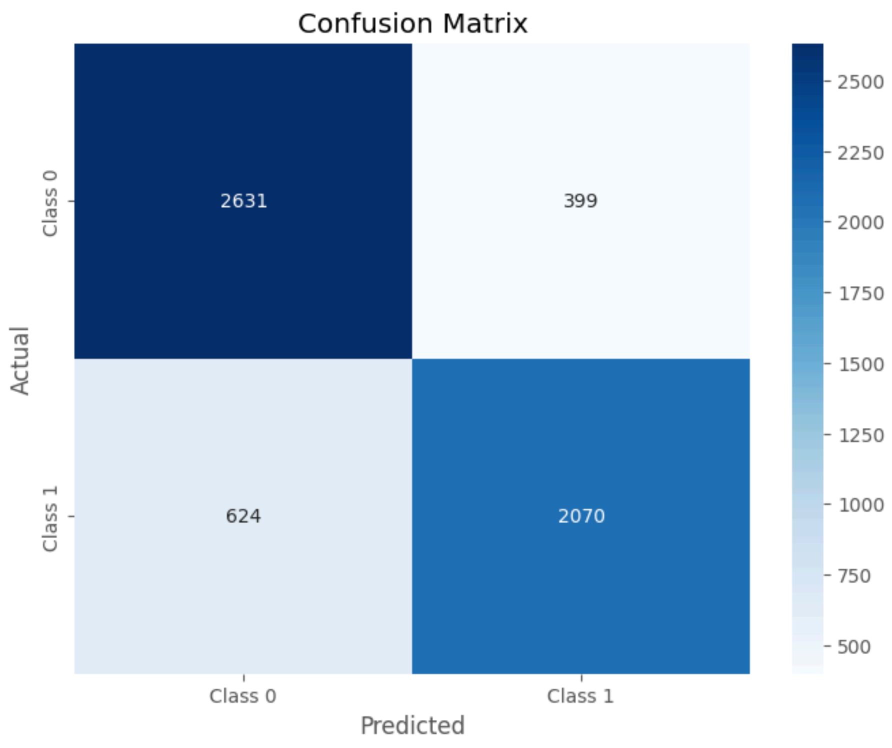
Naive Bayes

Here are the results for this model:

```
Bernoulli Accuracy: 0.8212788259958071

              precision    recall  f1-score   support

           0       0.81      0.87      0.84      3030
           1       0.84      0.77      0.80      2694

    accuracy                           0.82      5724
   macro avg       0.82      0.82      0.82      5724
weighted avg       0.82      0.82      0.82      5724
```

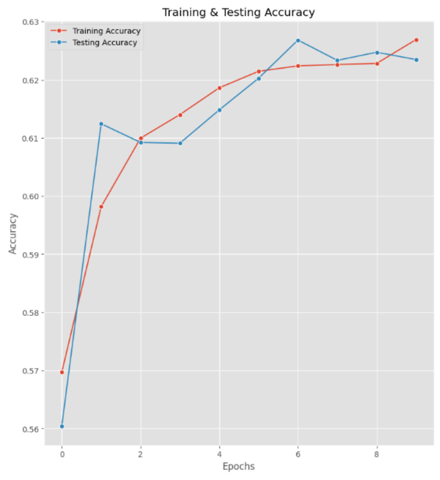
We have reached 82 percent accuracy which is promising. Also, this model only ran for less than a minute which is an important factor.

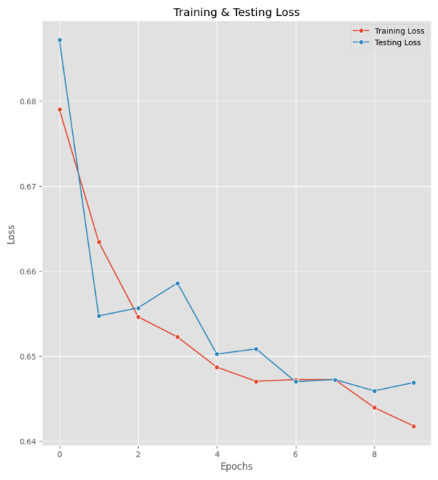
The below image shows the confusion matrix for this model:



LSTM - Model 1

Now, we feed this data to our model, and here are the results:

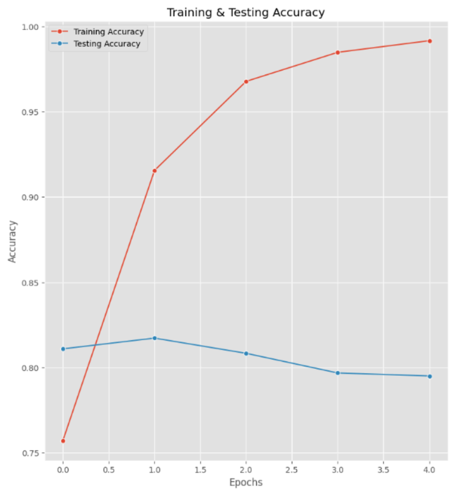


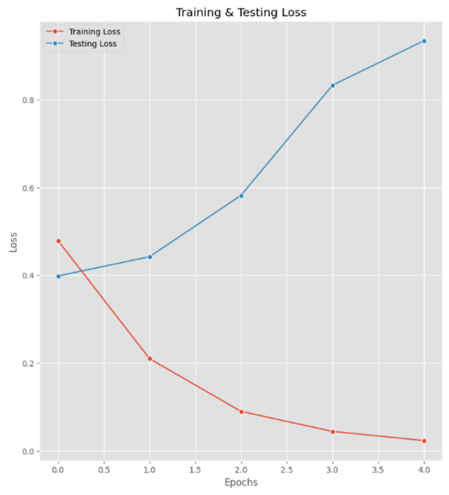


We trained the model for 10 epochs which took around 30 to 60 minutes. The test accuracy and train accuracy are 63 and 62 percent which is a very poor result for this much time and resources.

LSTM - Model 2

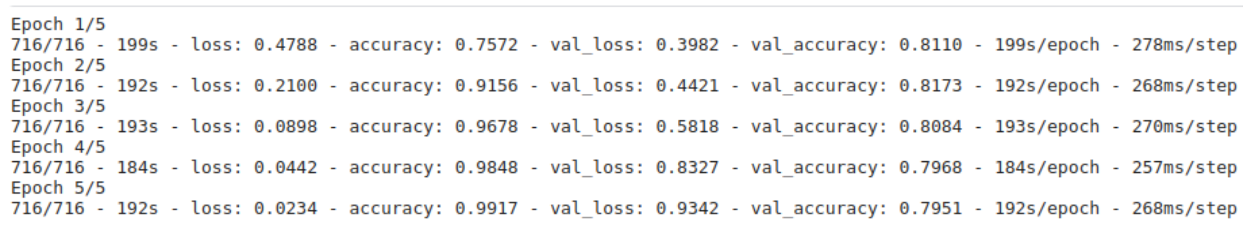
We train the model for 5 epochs to reduce the chance of overfitting. Here are the results in each epoch:





We can see that test accuracy has dropped after the second epoch which may be a sign of overfitting.

This is the result for each epoch:

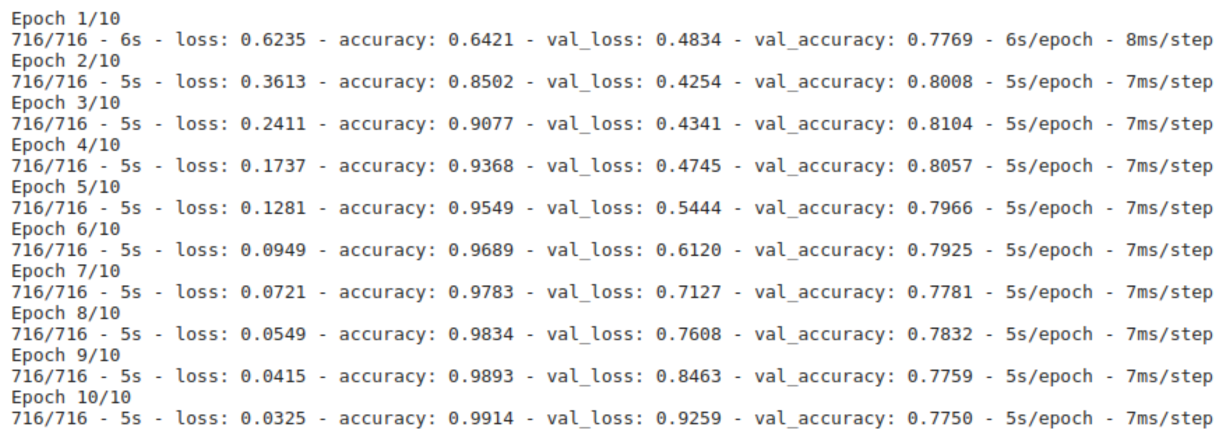


```
Epoch 1/5
716/716 - 199s - loss: 0.4788 - accuracy: 0.7572 - val_loss: 0.3982 - val_accuracy: 0.8110 - 199s/epoch - 278ms/step
Epoch 2/5
716/716 - 192s - loss: 0.2100 - accuracy: 0.9156 - val_loss: 0.4421 - val_accuracy: 0.8173 - 192s/epoch - 268ms/step
Epoch 3/5
716/716 - 193s - loss: 0.0898 - accuracy: 0.9678 - val_loss: 0.5818 - val_accuracy: 0.8084 - 193s/epoch - 270ms/step
Epoch 4/5
716/716 - 184s - loss: 0.0442 - accuracy: 0.9848 - val_loss: 0.8327 - val_accuracy: 0.7968 - 184s/epoch - 257ms/step
Epoch 5/5
716/716 - 192s - loss: 0.0234 - accuracy: 0.9917 - val_loss: 0.9342 - val_accuracy: 0.7951 - 192s/epoch - 268ms/step
```

The train accuracy is 99 and the test accuracy is 80 percent which is still a good result since the model only took a few minutes to run.

LSTM - Model 3

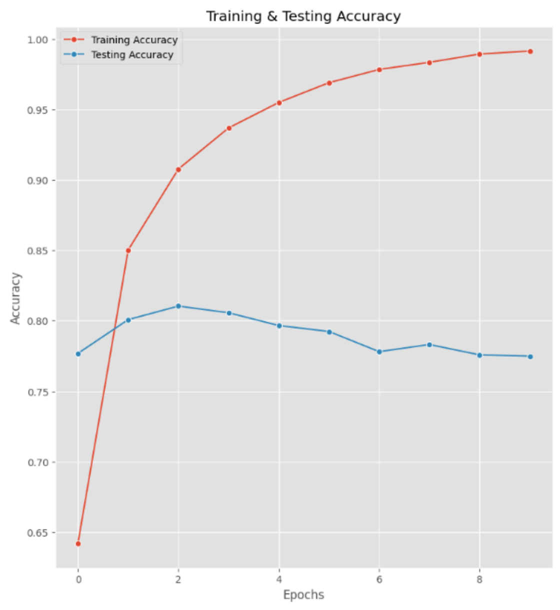
Here are the results for 10 epochs, the model ran for a few minutes:

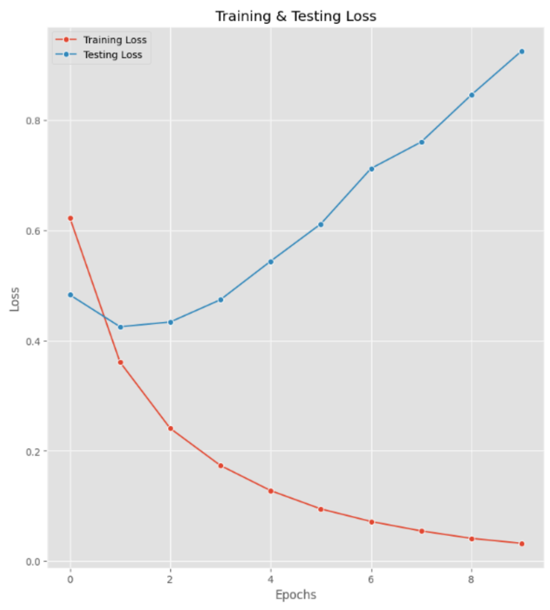


```
Epoch 1/10
716/716 - 6s - loss: 0.6235 - accuracy: 0.6421 - val_loss: 0.4834 - val_accuracy: 0.7769 - 6s/epoch - 8ms/step
Epoch 2/10
716/716 - 5s - loss: 0.3613 - accuracy: 0.8502 - val_loss: 0.4254 - val_accuracy: 0.8008 - 5s/epoch - 7ms/step
Epoch 3/10
716/716 - 5s - loss: 0.2411 - accuracy: 0.9077 - val_loss: 0.4341 - val_accuracy: 0.8104 - 5s/epoch - 7ms/step
Epoch 4/10
716/716 - 5s - loss: 0.1737 - accuracy: 0.9368 - val_loss: 0.4745 - val_accuracy: 0.8057 - 5s/epoch - 7ms/step
Epoch 5/10
716/716 - 5s - loss: 0.1281 - accuracy: 0.9549 - val_loss: 0.5444 - val_accuracy: 0.7966 - 5s/epoch - 7ms/step
Epoch 6/10
716/716 - 5s - loss: 0.0949 - accuracy: 0.9689 - val_loss: 0.6120 - val_accuracy: 0.7925 - 5s/epoch - 7ms/step
Epoch 7/10
716/716 - 5s - loss: 0.0721 - accuracy: 0.9783 - val_loss: 0.7127 - val_accuracy: 0.7781 - 5s/epoch - 7ms/step
Epoch 8/10
716/716 - 5s - loss: 0.0549 - accuracy: 0.9834 - val_loss: 0.7608 - val_accuracy: 0.7832 - 5s/epoch - 7ms/step
Epoch 9/10
716/716 - 5s - loss: 0.0415 - accuracy: 0.9893 - val_loss: 0.8463 - val_accuracy: 0.7759 - 5s/epoch - 7ms/step
Epoch 10/10
716/716 - 5s - loss: 0.0325 - accuracy: 0.9914 - val_loss: 0.9259 - val_accuracy: 0.7750 - 5s/epoch - 7ms/step
```

It seems that 3 epochs are optimal for this model since after the third epoch, the test accuracy has dropped while the train accuracy is increasing.

So, it is safe to say that after that, the model has memorized the data rather than generalization rather of the data. The below diagram shows the accuracy and loss for train and test data:





It is clear that test accuracy is decreasing after the third epoch.

Discussion

So far, we made different models and considered various factors such as time, memory, accuracy, and practicality. So let's recap.

Logistic Regression and Naive Bayes

- Computationally efficient
- Quick training

Promising accuracyThese models are suitable for situations where model interpretability is crucial.

LSTM Models

- Complex representations of sequential data.

- Model 2 had strong accuracy with reasonable training time.

Considering the trade-offs, Naive Bayes may be preferred for its balance between accuracy and computational efficiency. However, depending on the specific requirements and the trade-off between complexity and performance, other models like Logistic Regression or the LSTM-based Model 2 are outstanding options.

In conclusion, the choice of the best model depends on the priorities. The presented models provide a wide range of options.

References

https://www.kaggle.com/code/msvrao/bert-pretrained-vs-lstm
https://medium.com/@nguyenduchuyvn/sarcasm-detection-with-machine-learning-92538da893ec
https://www.geeksforgeeks.org/sarcasm-detection-using-neural-networks/
https://www.hindawi.com/journals/wcmc/2022/1653696/
https://www.kaggle.com/code/alnourabdalrahman9/distilbert-transformer-for-sarcasm-detection
https://www.kaggle.com/code/tabualkher/sarcasm-using-nn-with-gap1d-layer
https://www.kaggle.com/code/ernestglukhov/getting-embeddings
https://www.kaggle.com/code/osamaabobakr/sarcasm-detection-train-99-9-testing-81
https://www.kaggle.com/code/mjavadpur/mj-sarcasm-headlines
https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1194/reports/custom/15791781.pdf
https://www.kaggle.com/code/himanshubhenwal/sarcasm-classifier-using-tensorflow
https://www.kaggle.com/code/alperkaraca1/sarcasm-detection
https://www.kaggle.com/code/marcosgois07/sarcasm-detection-with-autokeras
https://www.kaggle.com/code/abdulbasitniazi/word2vec-glove-a-beginner-s-guide