Product Reviews Sentiment Analysis

Καρούσος Γιάννης - 20222022040171

https://github.com/GiannisKarousos/Deep_Learning_Karousos

Sentiment analysis (text-based) project for the Deep Learning course.

Exploratory Data Analysis

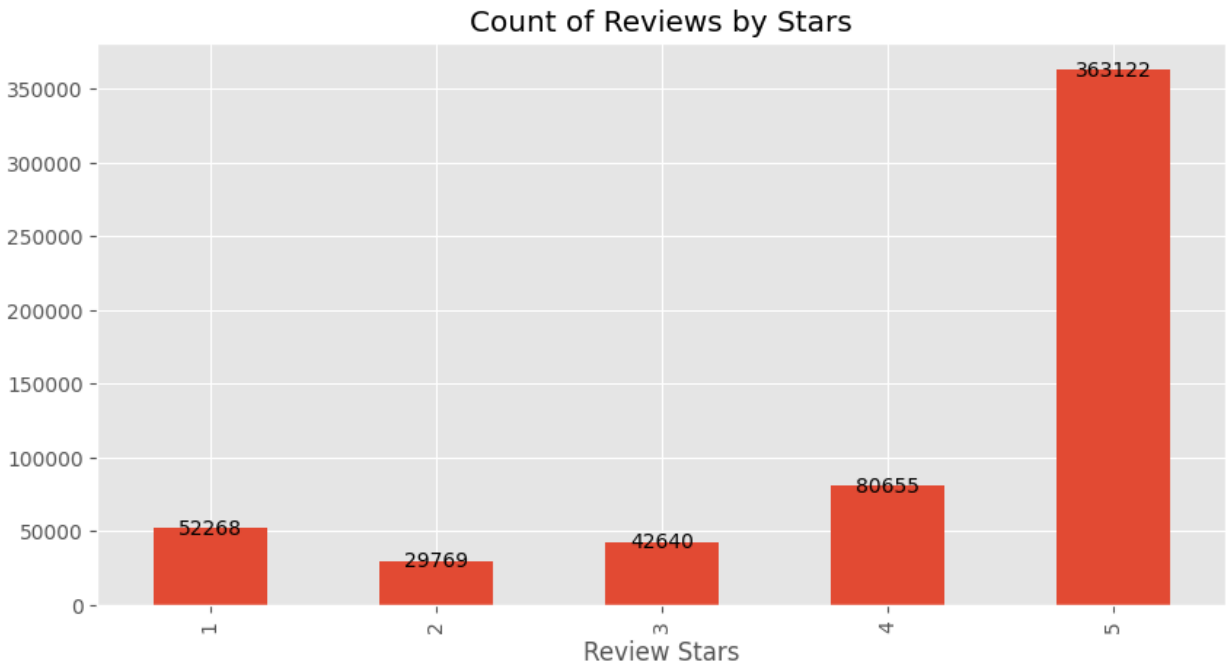
Our dataset is based on a Kaggle project (https://www.kaggle.com/datasets/snap/amazon-fine-food-reviews) and it has to do with food products. We have actually loaded it as a dataframe by reading the CSV (Reviews.csv) in the .ipynv named “EDA&VADER&Roberta”. The shape of it is (568454, 10). Below we can have a first view.

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary	Text
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	5	1303862400	Good Quality Dog Food	I have bought several of the Vitality canned d...
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	1	1346976000	Not as Advertised	Product arrived labeled as Jumbo Salted Peanut...
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	4	1219017600	"Delight" says it all	This is a confection that has been around a fe...
3	4	B000UA0QIQ	A395BORC6FGVXV	Karl	3	3	2	1307923200	Cough Medicine	If you are looking for the secret ingredient i...
4	5	B006K2ZZ7K	A1UQRSCLF8GW1T	Michael D. Bigham "M. Wassir"	0	0	5	1350777600	Great taffy	Great taffy at a great price. There was a wid...

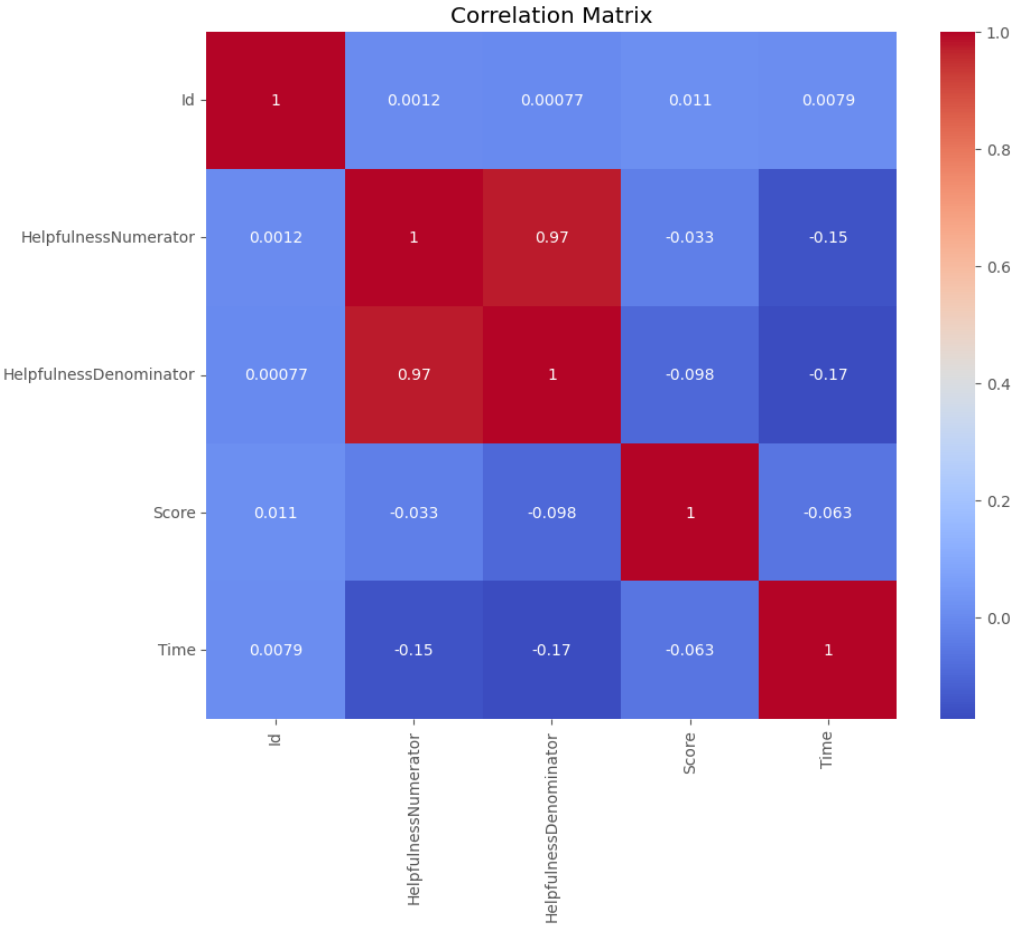
More specifically:

```
RangeIndex: 568454 entries, 0 to 568453
Data columns (total 10 columns):
 #   Column                  Non-Null Count   Dtype
---  ------                  --------------   -----
 0   Id                      568454 non-null  int64
 1   ProductId               568454 non-null  object
 2   UserId                  568454 non-null  object
 3   ProfileName             568438 non-null  object
 4   HelpfulnessNumerator    568454 non-null  int64
 5   HelpfulnessDenominator  568454 non-null  int64
 6   Score                   568454 non-null  int64
 7   Time                    568454 non-null  int64
 8   Summary                 568427 non-null  object
 9   Text                    568454 non-null  object
dtypes: int64(5), object(5)
memory usage: 43.4+ MB
```

Checking the distribution of the review column, we can clearly notice that the 5-star review count is much more than the others.



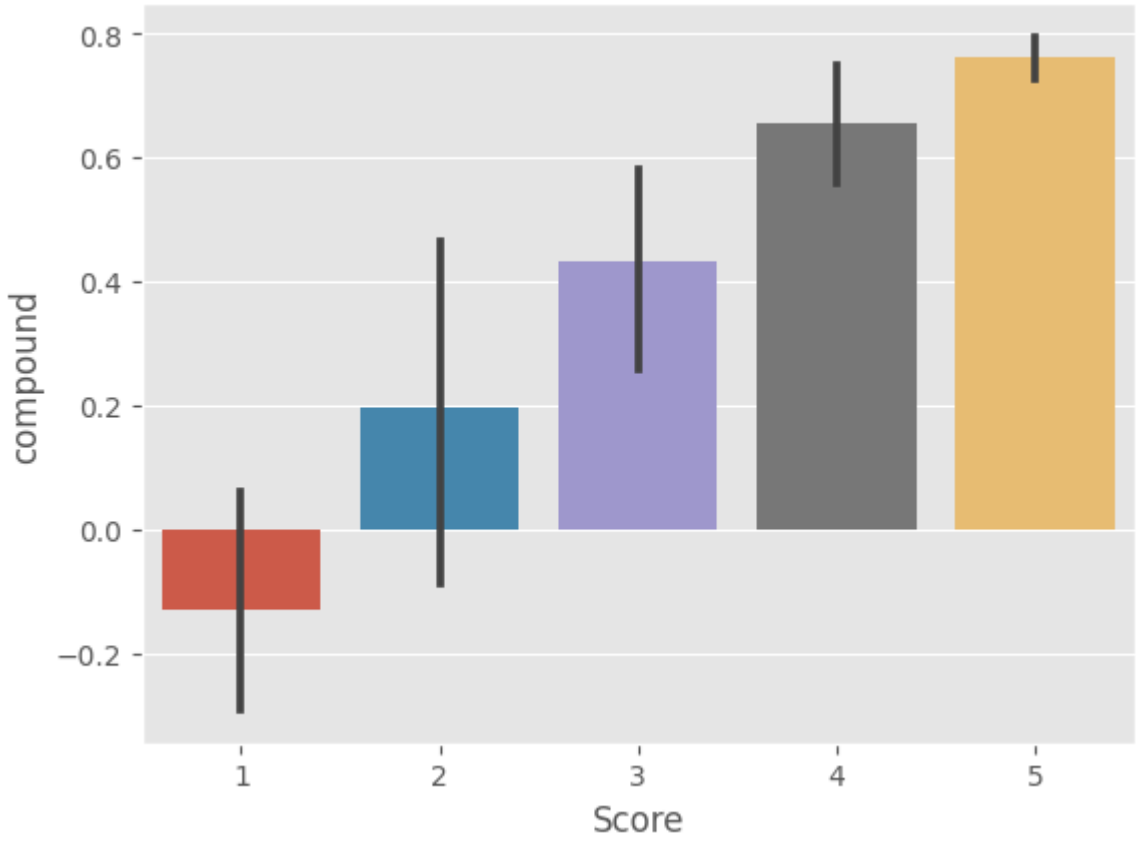
Below we can see a correlation matrix as well:



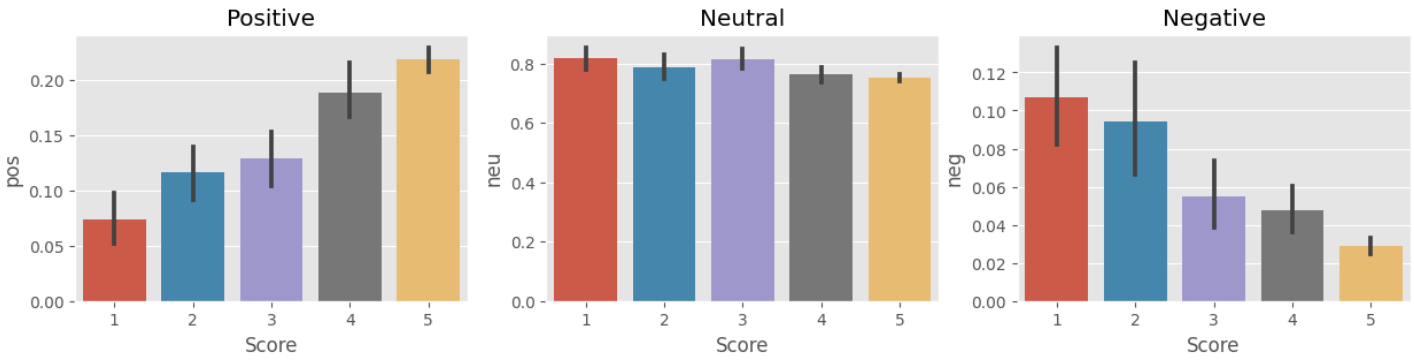
VADER - Bag of Words Approach

VADER (Valence Aware Dictionary and sEntiment Reasoner) is a rule-based sentiment analysis tool specifically designed for analyzing sentiment in textual data. It is a widely used tool in natural language processing (NLP) and is particularly known for its effectiveness in sentiment analysis of social media texts. It uses a combination of lexical and grammatical heuristics, as well as a pre-trained sentiment lexicon that contains a list of words and their associated sentiment scores. The lexicon includes both sentiment-bearing words (e.g., "happy," "sad") and intensifiers or modifiers that affect the sentiment intensity (e.g., "very," "extremely"). The sentiment scores in the lexicon are typically between -1 and +1, where negative values indicate negative sentiment, positive values indicate positive sentiment and a score of zero indicates neutral sentiment. VADER provides sentiment analysis by assigning sentiment scores to individual text units (sentences or phrases) or entire documents. These scores indicate the positivity, negativity, and neutrality of the text. The overall sentiment is often summarized using metrics like compound score (which combines the positive, negative, and neutral scores) or by classifying the sentiment as positive, negative, or neutral based on a threshold.

By applying the VADER on the first 500 rows of our dataset, we can clearly see that it makes sense and the results are as expected. In the barplot below, the compound VADER score is more positive when it comes to 4 or 5-star reviews. At the same time, the score gets lower for the 1 or 2-star reviews.



Another visualization shows the positive, neutral and negative scores of the VADER depending on the Reviews:



One notable drawback of VADER is its limited consideration of neighboring words or context when assigning sentiment scores. The tool primarily focuses on individual words and their associated sentiment scores without taking into account the influence of adjacent words. This approach may result in instances where the sentiment of a word is misinterpreted due to the lack of contextual understanding. The absence of analyzing the relationships and dependencies between neighboring words can hinder the accurate identification of sentiment nuances, making it challenging to capture the true sentiment expressed in complex and context-dependent sentences or phrases.

Roberta Pre-trained Model

RoBERTa (Robustly Optimized BERT Pretraining Approach) is a state-of-the-art pre-trained language model based on the BERT (Bidirectional Encoder Representations from Transformers) architecture. It addresses some limitations of the original BERT model by employing additional training techniques and data augmentation strategies. RoBERTa significantly improves the performance of various natural language processing (NLP) tasks, including sentiment analysis. One of the key advantages of RoBERTa is its ability to capture contextual information effectively. It considers the surrounding words and their relationships when generating word representations, enabling a more nuanced understanding of sentiment in text. This contextual understanding allows RoBERTa to better handle cases of negation, sarcasm, and subtle sentiment expressions.

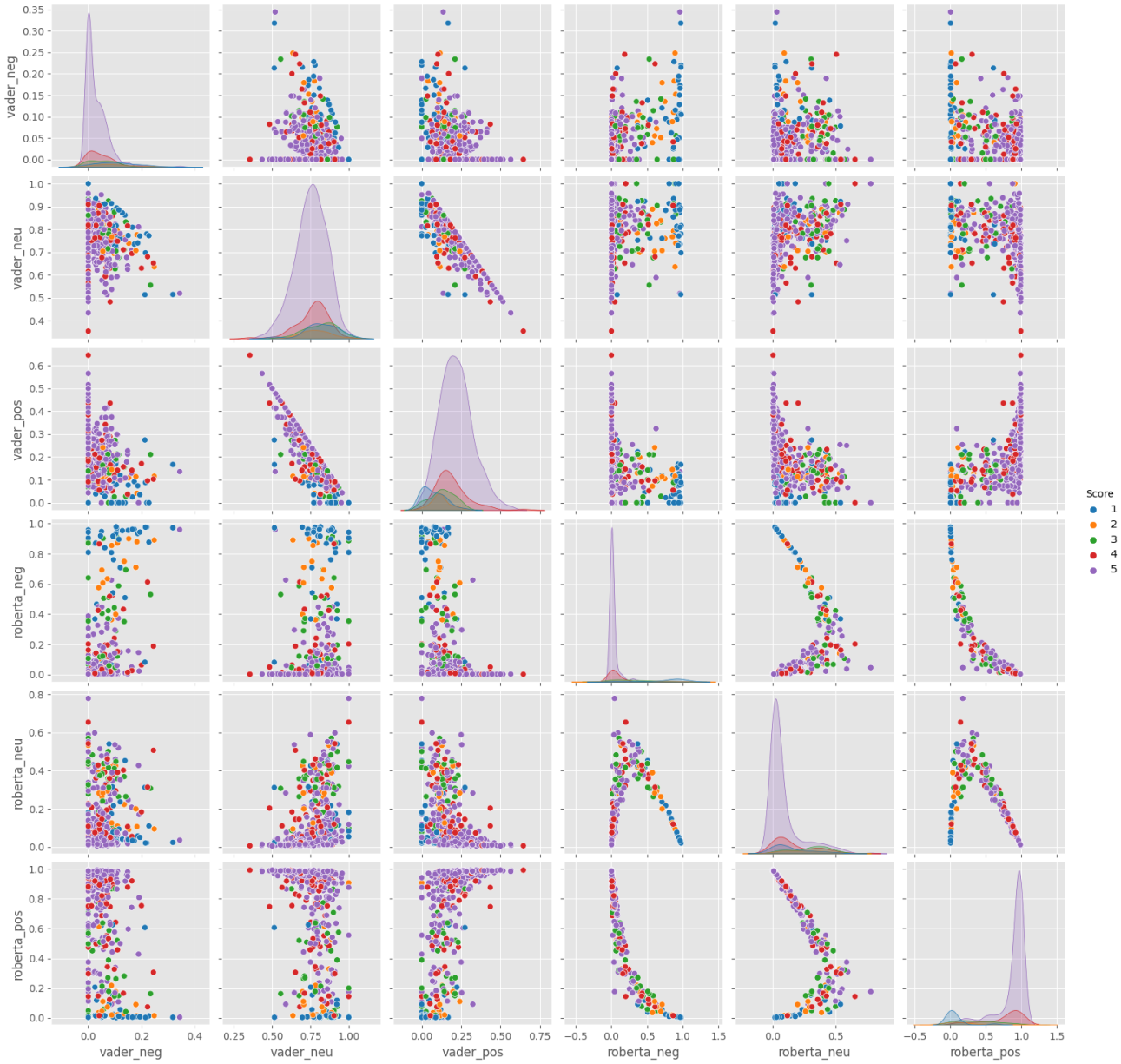
Here is an example based on a review of our dataset:

```
This oatmeal is not good. Its mushy, soft, I don't like it. Quaker Oats is the way to go.

{'neg': 0.22, 'neu': 0.78, 'pos': 0.0, 'compound': -0.5448}
```

Comparing VADER & RoBERTa

One thing we notice in the dashboard below is that there are some correlations between the two models but it seems that the VADER model is a little bit less confident and that RoBERTa model better separates the different sentiments.



Review examples for VADER and RoBERTa

Below we can see where the two models failed to identify the sentiment!

Positive RoBERTa with 1-star review:

```
df_both.query('Score == 1').sort_values('roberta_pos', ascending=False)['Text'].values[0]
```

```
'I felt energized within five minutes, but it lasted for about 45 minutes. I paid $3.99 for this drink. I could have just drunk a cup of coffee and saved my money.'
```

Positive VADER with 1-star review:

```
df_both.query('Score == 1').sort_values('vader_pos', ascending=False)['Text'].values[0]
```

```
'So we cancelled the order.  It was cancelled without any problem.  That is a positive note...'
```

Negative RoBERTa with 5-star review:

```
df_both.query('Score == 5').sort_values('roberta_neg', ascending=False)['Text'].values[0]
```

```
'this was sooooo deliscious but too bad i ate em too fast and gained 2 pds! my fault'
```

Negative VADER with 5-star review:

```
df_both.query('Score == 5').sort_values('vader_neg', ascending=False)['Text'].values[0]
```

```
'this was sooooo deliscious but too bad i ate em too fast and gained 2 pds! my fault'
```

Our Models

In this section, we are going to train a Simple Neural Network (SNN), a Convolutional Neural Network (CNN) and a Recurrent Neural Network (RNN - LSTM).

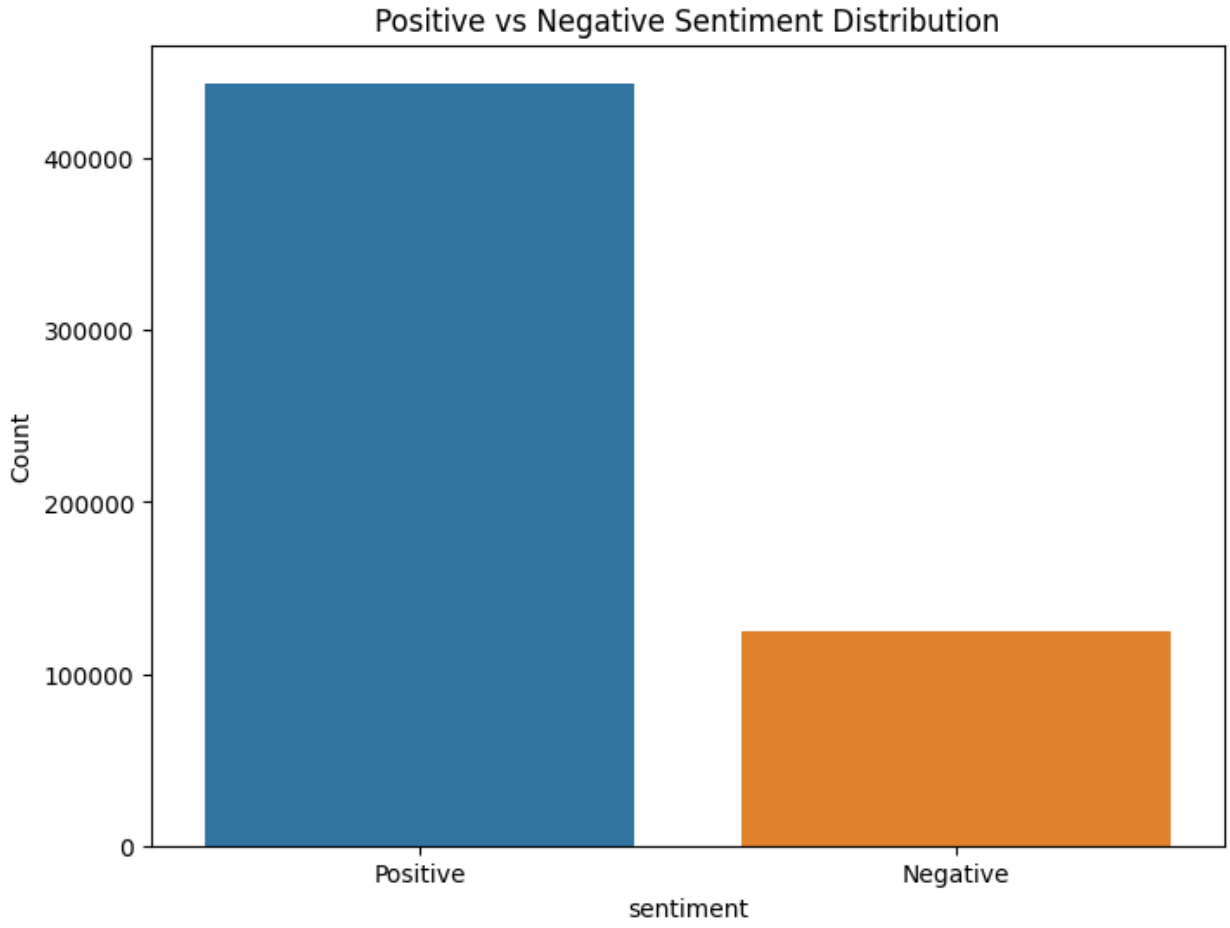
Data Resampling

The first thing we are going to do is to only keep the columns needed, so, we keep text column renamed it to review and through the score column we created the sentiment column. For the 4, 5-star reviews, we gave a positive sentiment while for the rest of the dataset (1, 2, 3-star reviews) we gave a negative sentiment.

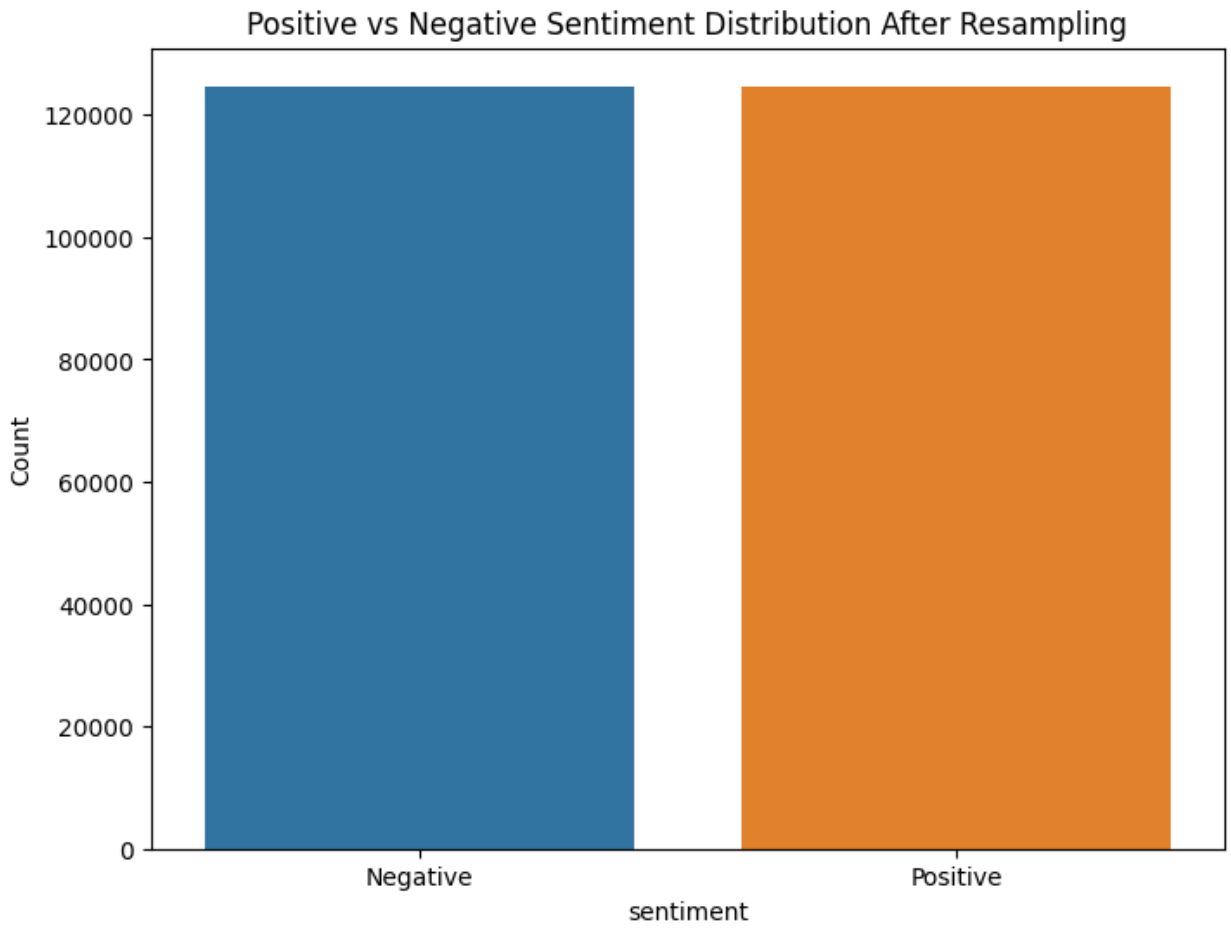
Here is a glimpse of the dataset:

	review	sentiment
0	I have bought several of the Vitality canned d...	Positive
1	Product arrived labeled as Jumbo Salted Peanut...	Negative
2	This is a confection that has been around a fe...	Positive
3	If you are looking for the secret ingredient i...	Negative
4	Great taffy at a great price. There was a wid...	Positive
5	I got a wild hair for taffy and ordered this f...	Positive
6	This saltwater taffy had great flavors and was...	Positive
7	This taffy is so good. It is very soft and ch...	Positive
8	Right now I'm mostly just sprouting this so my...	Positive
9	This is a very healthy dog food. Good for thei...	Positive

At the same time, the main issue here is the imbalance between the Positive and Negative label:



The issue of imbalanced data poses a challenge for our model's performance. Currently, our dataset exhibits a significant disparity between the number of positive and negative labels. With 443,777 instances of positive sentiment and only 124,677 instances of negative sentiment, this imbalance can skew the model's ability to accurately predict sentiments. To address this problem, we have decided to resample the positive label by downsampling the data. By reducing the number of positive instances to match the count of the negative instances, we aim to create a more balanced dataset that will enable our model to learn from a representative distribution of sentiments.



Data Preprocessing

For this one, we have created a function named “‘preprocess_text” to help us clean the reviews!

- Converting the input sentence to lowercase: By converting all the characters to lowercase, the function ensures uniformity in the text, treating uppercase and lowercase versions of the same word as identical.

- Removing HTML tags: The function utilizes a separate remove_tags function (not provided in the code snippet) to eliminate any HTML tags that might be present in the sentence. This step is crucial when dealing with text data from web pages or other HTML-based sources.

- Removing punctuations and numbers: This step utilizes regular expressions (re module) to remove any non-alphabetic characters and numbers from the sentence. By doing so, it helps to eliminate noise and non-essential information that might not contribute significantly to the semantic meaning of the text.

- Removing single characters: This step removes single characters (e.g., standalone letters) that may not carry substantial meaning on their own. The regular expression pattern \s+[^a-zA-Z]\s+ matches any single character surrounded by whitespace, and replaces it with a single space.

- Removing multiple spaces: This step utilizes another regular expression, \s+, to match and replace multiple consecutive whitespace characters with a single space. It helps to normalize the spacing within the sentence and avoid unnecessary gaps.

- Removing stopwords: Stopwords are common words in a language (e.g., "the," "is," "and") that do not typically contribute much to the overall meaning of a sentence. The code snippet uses the stopwords module from the Natural Language Toolkit (NLTK) to obtain a list of English stopwords. It then compiles a regular expression pattern based on these stopwords and removes them from the sentence, resulting in a more concise representation of the text with fewer noise words.

Lastly, we convert our sentiment column to labels, 1 for positive and 0 for negative.

Splitting the Dataset

The splitting is done with the following parameters:

- **test_size = 0.20:** This parameter determines the proportion of the dataset that will be allocated for testing. In this case, 20% of the data will be used for testing, while the remaining 80% will be used for training the model.

- **random_state = 42:** This parameter is used to set a specific random seed, ensuring the reproducibility of the data splitting process. By setting the random_state to a specific value (in this case, 42), the same train-test split will be obtained every time the code is executed, allowing for consistent comparisons and analysis.

The resulting subsets can be interpreted as follows:

- **X_train:** This subset contains a randomly selected portion (80% in this case) of the original feature dataset X (reviews). It will be used to train the machine learning model, allowing it to learn patterns and relationships within the data.
- **X_test:** This subset consists of the remaining portion (20% in this case) of the original feature dataset X (reviews). It serves as a holdout or unseen dataset that was not used during training. The purpose of the X_test is to evaluate the performance of the trained model on unseen data and assess its generalization capabilities.
- **y_train:** This subset contains the corresponding labels for the samples in X_train. It represents the ground truth or known outcomes that the model will learn from during training.
- **y_test:** This subset includes the corresponding labels for the samples in X_test. It serves as the reference or expected outcomes against which the predictions of the trained model will be compared and evaluated.

Embedding Layer

Converting text to numeric

```
word_tokenizer = Tokenizer()
word_tokenizer.fit_on_texts(X_train)

X_train = word_tokenizer.texts_to_sequences(X_train)
X_test = word_tokenizer.texts_to_sequences(X_test)
```

Converting textual review data, represented by X_train and X_test, to numeric form is necessary for several reasons. By utilizing the code snippet provided, we employ various techniques from the TensorFlow and Keras libraries to accomplish this transformation. Firstly, the Tokenizer class is employed to tokenize the text, splitting it into individual words or tokens. This step enables the creation of a vocabulary where each unique word is assigned a numerical index. By calling the fit_on_texts method on word_tokenizer with X_train, the tokenizer learns the vocabulary based on the training data. Next, the texts_to_sequences method is applied to both X_train and X_test, which converts the text into sequences of corresponding numerical indices based on the learned vocabulary. This process represents each word in the reviews with its

respective index. Converting the text data to a numeric format is crucial because machine learning models typically operate on numerical inputs. By transforming the text into a numerical representation, we enable the utilization of various machine learning algorithms and neural network architectures that require numeric inputs. This allows us to extract meaningful features and patterns from the text data and train models to make predictions or perform other text-based tasks effectively.

There is also time to create a vocabulary bucket of the total number of unique words in the text data. The '**word_tokenizer.word_index'** attribute returns a dictionary-like object that maps each word in the vocabulary to its corresponding numerical index. The +1 is added to account for an additional index that represents out-of-vocabulary or unknown words. Therefore, vocab_length represents the total number of unique words in the text data, including the additional index.

```
vocab_length = len(word_tokenizer.word_index) + 1
```

Padding the Sequences

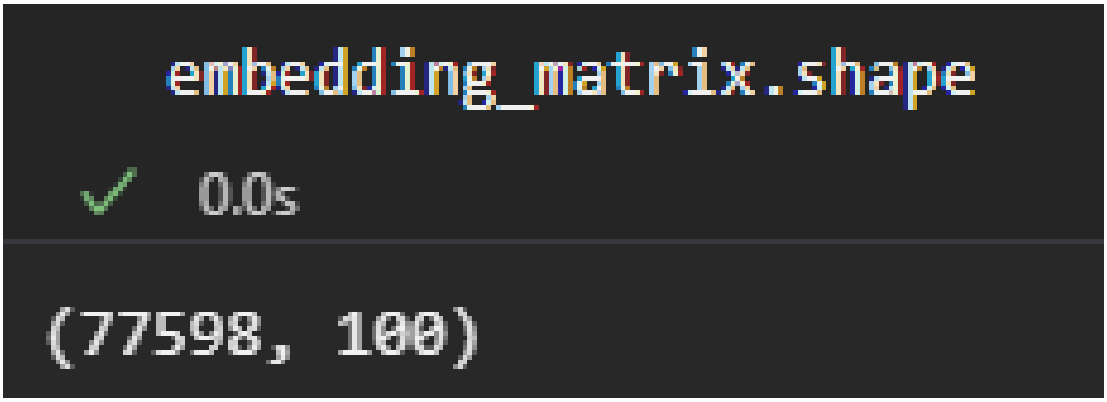
Padding is necessary because many deep learning models require fixed-size inputs. For example, when training a recurrent neural network (RNN) or a convolutional neural network (CNN), the input data is typically organized into batches, and all sequences within a batch need to have the same length.

```
maxlen = 100

X_train = pad_sequences(X_train, padding='post', maxlen = maxlen)
X_test = pad_sequences(X_test, padding='post', maxlen = maxlen)
```

Glove Word Embeddings and Embeddings Dictionary

Embeddings, transform words into dense and lower-dimensional vector representations. These vectors are learned through training on a large corpus of text data using techniques like GloVe. By training on a large amount of text, the embeddings capture the context and meaning of words based on their surrounding words. GloVe, which stands for Global Vectors for Word Representation, is an unsupervised learning algorithm used to generate word embeddings.



Models Training

(Training_Models.ipynb)

Simple Neural Network

Here is a simple sequential SNN model with an architecture with an embedding layer, a flatten layer, and a dense layer. The model takes input sequences, applies an embedding transformation, flattens the output, and passes it through a dense layer to produce a single output representing the probability of the input belonging to a particular class.

Here's a breakdown of the compilation settings:

- **optimizer='adam'**: This specifies the optimizer to be used during training. In this case, the Adam optimizer is chosen, which is a popular optimization algorithm in deep learning. Adam dynamically adjusts the learning rate during training and is known for its efficient and effective performance.

- **loss='binary_crossentropy':** This defines the loss function to be used during training. Since the SNN model is designed for binary classification (where the output is either 0 or 1), binary cross-entropy loss is commonly used. This loss function is suitable for problems where there are two classes and the goal is to maximize the prediction accuracy for each class.

- **metrics=['acc']:** This specifies the metric(s) to be computed during training and evaluation. In this case, 'acc' represents accuracy, which is a commonly used metric for classification tasks. Accuracy measures the percentage of correctly classified samples compared to the total number of samples.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 100, 100)          7759800

 flatten (Flatten)           (None, 10000)             0

 dense (Dense)               (None, 1)                 10001

=================================================================
Total params: 7769801 (29.64 MB)
Trainable params: 10001 (39.07 KB)
Non-trainable params: 7759800 (29.60 MB)
_________________________________________________________________
None
```

Here's also a breakdown of the training process:

- **batch_size=128:** This specifies the number of samples per gradient update. The training data will be divided into batches of size 128, and the model's weights will be updated after each batch.

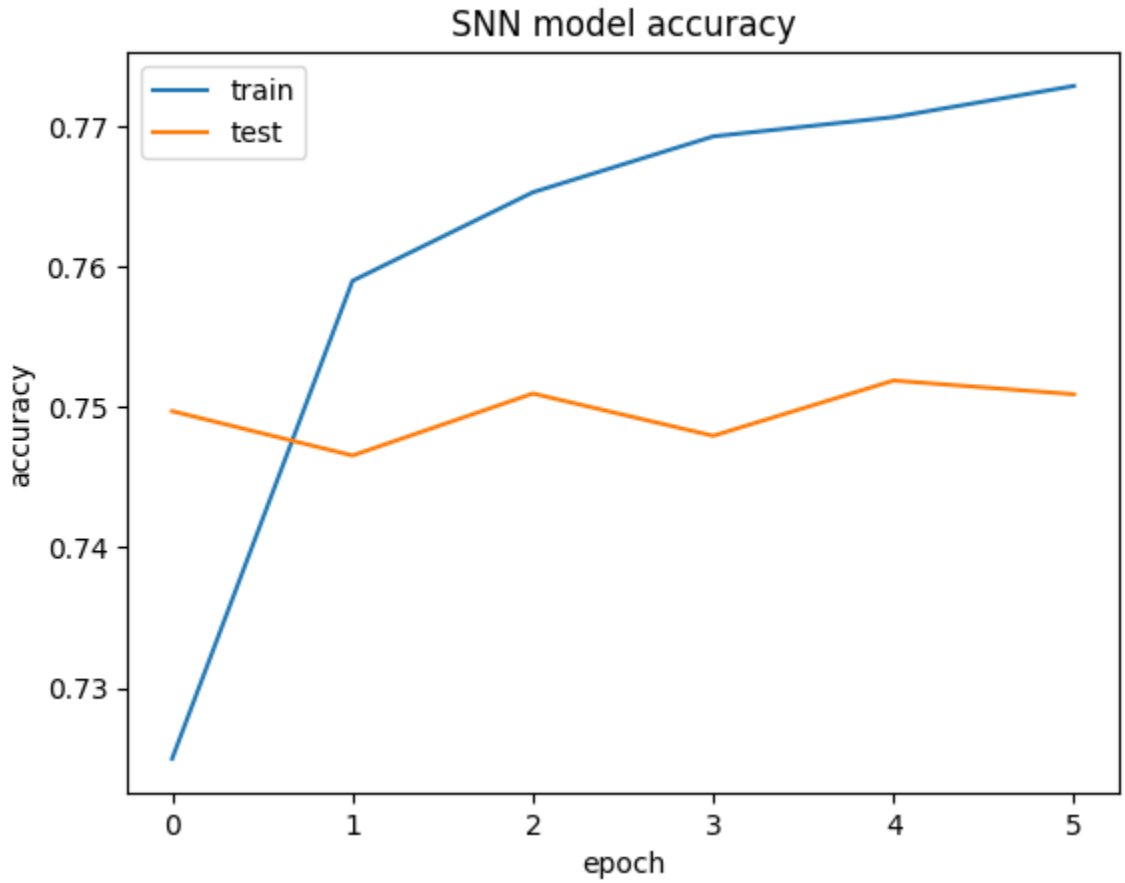
- **epochs=6:** This determines the number of times the entire training dataset will be passed through the network during training. In this case, the training process will iterate over the dataset six times.

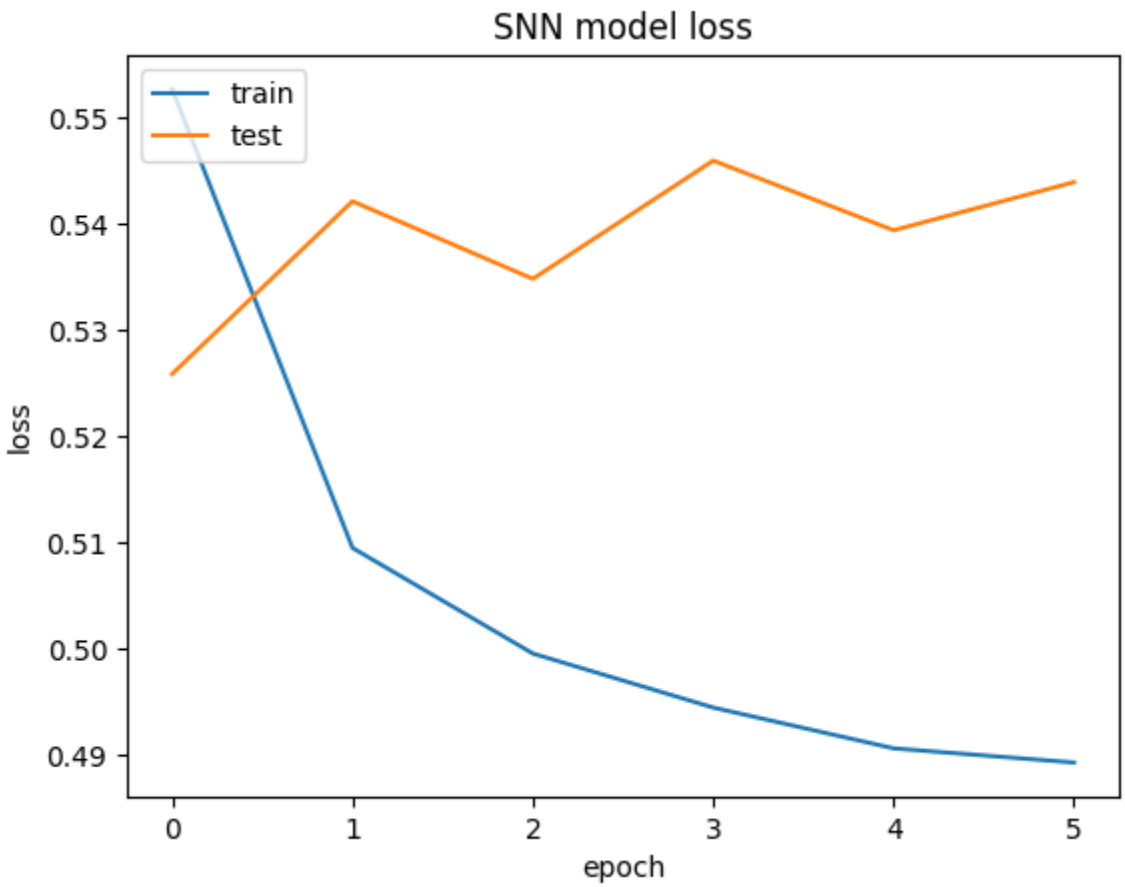
```
snn_model_history = snn_model.fit(X_train, y_train, batch_size=128, epochs=6, verbose=1, validation_split=0.2)
✓ 1m 12.4s

Epoch 1/6
1247/1247 [==============================] - 13s 10ms/step - loss: 0.5527 - acc: 0.7250 - val_loss: 0.5259 - val_acc: 0.7497
Epoch 2/6
1247/1247 [==============================] - 11s 9ms/step - loss: 0.5095 - acc: 0.7590 - val_loss: 0.5421 - val_acc: 0.7465
Epoch 3/6
1247/1247 [==============================] - 12s 9ms/step - loss: 0.4996 - acc: 0.7653 - val_loss: 0.5348 - val_acc: 0.7510
Epoch 4/6
1247/1247 [==============================] - 12s 10ms/step - loss: 0.4945 - acc: 0.7693 - val_loss: 0.5460 - val_acc: 0.7480
Epoch 5/6
1247/1247 [==============================] - 12s 9ms/step - loss: 0.4907 - acc: 0.7706 - val_loss: 0.5394 - val_acc: 0.7519
Epoch 6/6
1247/1247 [==============================] - 12s 9ms/step - loss: 0.4893 - acc: 0.7729 - val_loss: 0.5439 - val_acc: 0.7509
```

SNN output

```
Test Score: 0.5445172190666199
Test Accuracy: 0.7470072507858276
```





Convolutional Neural Network

This CNN model, has an embedding layer, a 1D convolutional layer, a global max pooling layer, and a dense layer. The model takes input sequences, applies an embedding transformation, performs convolutions, applies global max pooling, and passes the output through a dense layer to produce a single output representing the probability of the input belonging to a particular class.

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, 100, 100)          7759800

 conv1d (Conv1D)             (None, 96, 128)           64128

 global_max_pooling1d (Glob  (None, 128)               0
 alMaxPooling1D)

 dense_1 (Dense)             (None, 1)                 129

=================================================================
Total params: 7824057 (29.85 MB)
Trainable params: 64257 (251.00 KB)
Non-trainable params: 7759800 (29.60 MB)
_________________________________________________________________
None
```

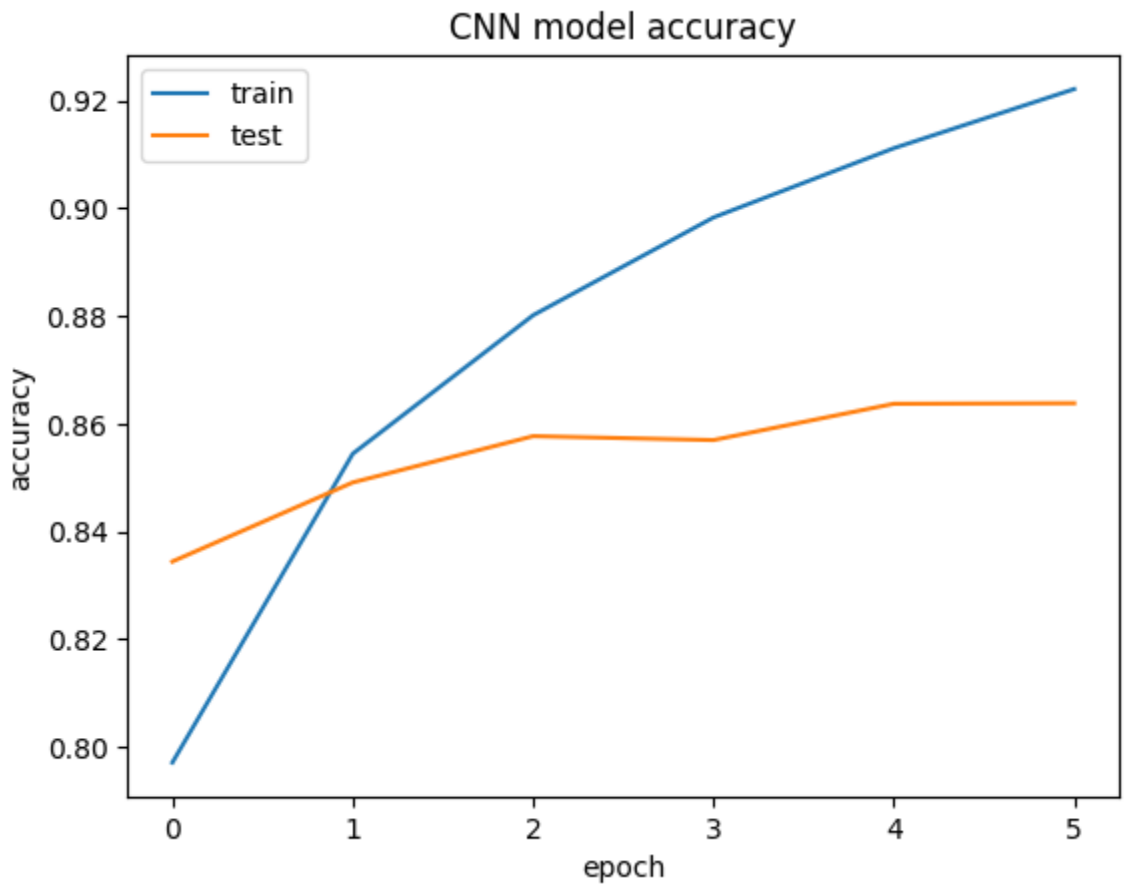
During training:

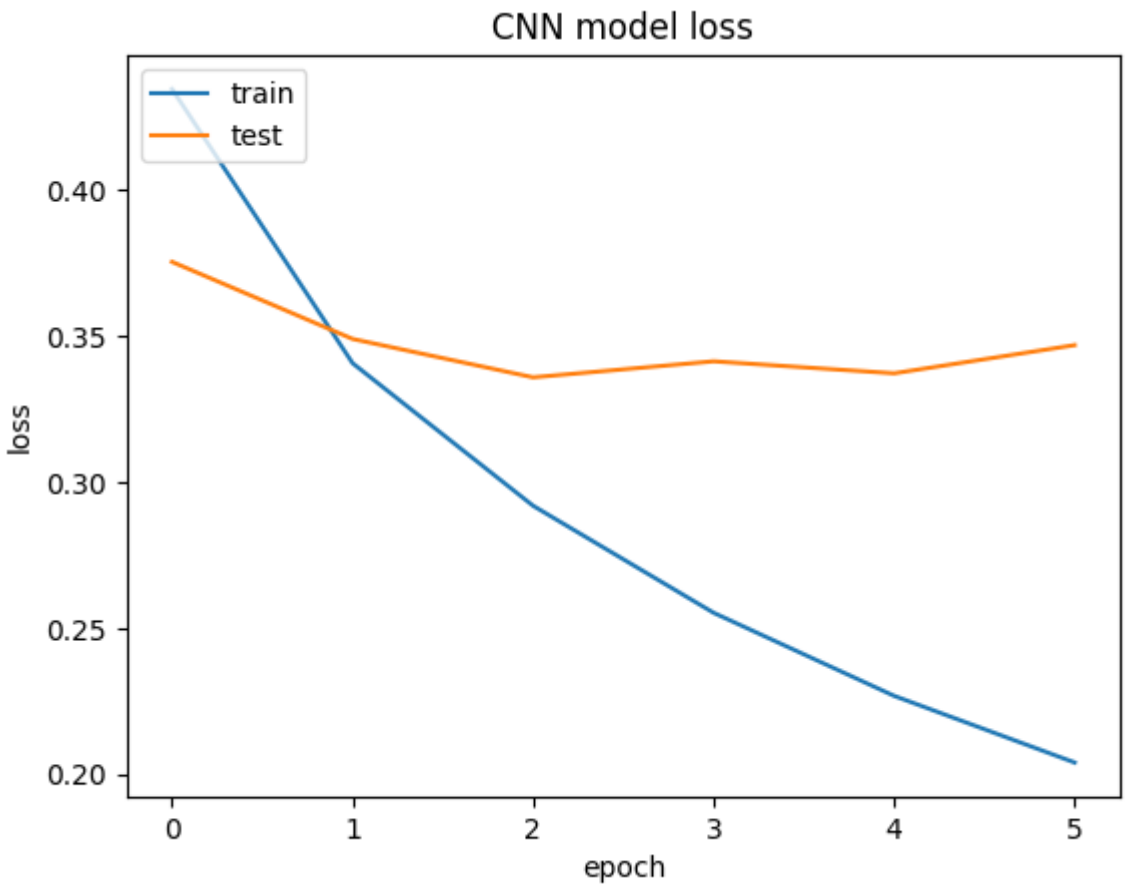
```
cnn_model_history = cnn_model.fit(X_train, y_train, batch_size=128, epochs=6, verbose=1, validation_split=0.2)
✓ 9m 59.3s

Epoch 1/6
1247/1247 [==============================] - 102s 81ms/step - loss: 0.4347 - acc: 0.7970 - val_loss: 0.3755 - val_acc: 0.8344
Epoch 2/6
1247/1247 [==============================] - 106s 85ms/step - loss: 0.3408 - acc: 0.8544 - val_loss: 0.3491 - val_acc: 0.8491
Epoch 3/6
1247/1247 [==============================] - 106s 85ms/step - loss: 0.2919 - acc: 0.8801 - val_loss: 0.3359 - val_acc: 0.8577
Epoch 4/6
1247/1247 [==============================] - 104s 84ms/step - loss: 0.2552 - acc: 0.8983 - val_loss: 0.3414 - val_acc: 0.8569
Epoch 5/6
1247/1247 [==============================] - 94s 75ms/step - loss: 0.2267 - acc: 0.9111 - val_loss: 0.3373 - val_acc: 0.8637
Epoch 6/6
1247/1247 [==============================] - 88s 70ms/step - loss: 0.2039 - acc: 0.9221 - val_loss: 0.3470 - val_acc: 0.8638
```

CNN output

```
Test Score: 0.3490977883338928
Test Accuracy: 0.8622847199440002
```


CNN model accuracy
train
test
0.92
0.90
0.88
0.86
0.84
0.82
0.80
accuracy
0
1
2
3
4
5
epoch



Recurrent Neural Network (LSTM)

A Recurrent Neural Network (RNN) with Long Short-Term Memory (LSTM) is a type of neural network commonly used for sequence data processing, such as natural language processing, speech recognition, and time series analysis. LSTM is an extension of the RNN architecture that addresses the vanishing gradient problem and allows the network to capture long-term dependencies in the data.

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_2 (Embedding)     (None, 100, 100)          7759800

 lstm (LSTM)                 (None, 128)               117248

 dense_2 (Dense)             (None, 1)                 129

=================================================================
Total params: 7877177 (30.05 MB)
Trainable params: 117377 (458.50 KB)
Non-trainable params: 7759800 (29.60 MB)
_________________________________________________________________
None
```

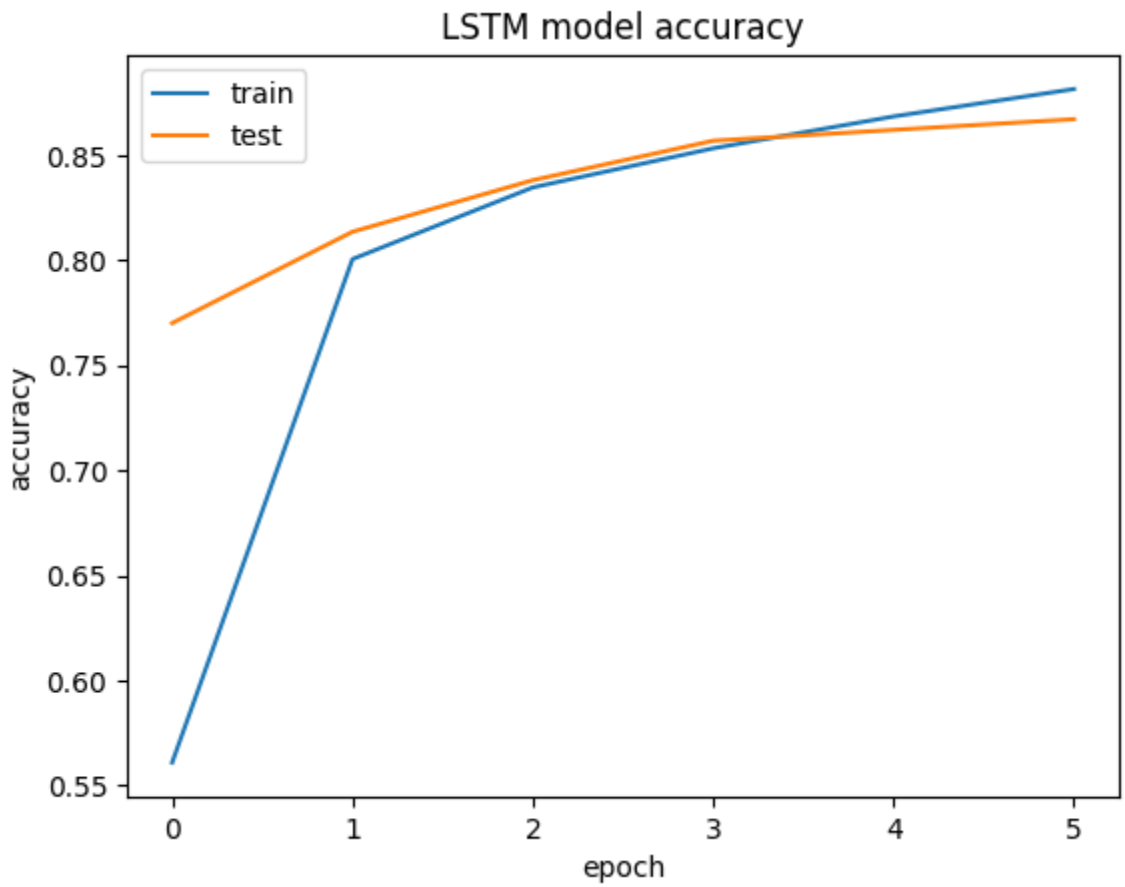
During training:

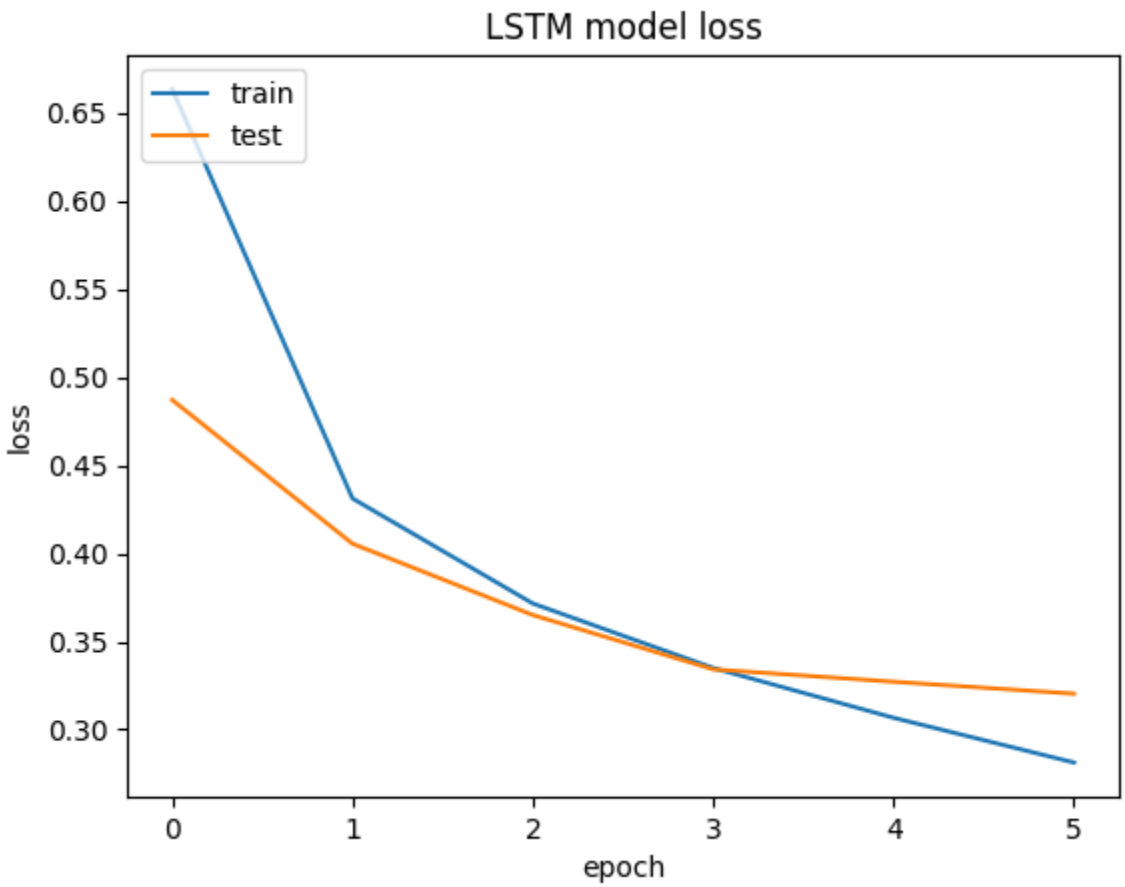
```
lstm_model_history = lstm_model.fit(X_train, y_train, batch_size=128, epochs=6, verbose=1, validation_split=0.2)
✓ 37m 45.5s

Epoch 1/6
1247/1247 [==============================] - 399s 311ms/step - loss: 0.6634 - acc: 0.5608 - val_loss: 0.4871 - val_acc: 0.7701
Epoch 2/6
1247/1247 [==============================] - 390s 312ms/step - loss: 0.4311 - acc: 0.8006 - val_loss: 0.4055 - val_acc: 0.8137
Epoch 3/6
1247/1247 [==============================] - 379s 304ms/step - loss: 0.3715 - acc: 0.8348 - val_loss: 0.3650 - val_acc: 0.8383
Epoch 4/6
1247/1247 [==============================] - 377s 303ms/step - loss: 0.3350 - acc: 0.8534 - val_loss: 0.3342 - val_acc: 0.8571
Epoch 5/6
1247/1247 [==============================] - 367s 294ms/step - loss: 0.3067 - acc: 0.8686 - val_loss: 0.3271 - val_acc: 0.8622
Epoch 6/6
1247/1247 [==============================] - 353s 283ms/step - loss: 0.2813 - acc: 0.8817 - val_loss: 0.3204 - val_acc: 0.8673
```

LSTM output

```
Test Score: 0.31545889377593994
Test Accuracy: 0.8691624402999878
```


LSTM model accuracy
train
test
accuracy
epoch
0.85
0.80
0.75
0.70
0.65
0.60
0.55
0
1
2
3
4
5



Live Representation

The LSTM model provides the best results concerning this sentiment project, thus we saved it into an h5 doc to for future use!

```
lstm_model.save(f"./c1_lstm_model_acc_{round(score[1], 3)}.h5", save_format='h5')
```

After that, we're going to load the model on the 'test_model&crawling.ipynb' for a live DEMO.

In this demo we start a browser with selenium and we actually crawl the top reviews for a product! After that we're using the model to get some results!