Comparison of algorithms in Amazon reviews sentiment analysis

Victor Luis Collado Ureta
CWID: A20522008
Illinois Institute of Technology
Chicago, United States of America
vcolladoureta@hawk.iit.edu

Daniel Pizarro Caro
CWID: A20521462
Illinois Institute of Technology
Chicago, United States of America
dpizarrocaro@hawk.iit.edu

ABSTRACT

This project consists of a Sentiment Analysis of Amazon reviews of musical instruments. A series of specific methods such as Linear Regression, Naive Bayes, SVC and Decision Tree have been used to perform this analysis. The results of the different methods have been compared with each other to see which is the most efficient and graphs have been used to illustrate them.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning**; **Machine learning approaches**; **Classification and regression trees**; **Modeling methodologies**; **Information extraction**; **Lexical semantics**; **Supervised learning by regression**; **Learning from critiques**.

KEYWORDS

Datasets, sentiment analysis, machine learning methods, Linear Regression, Decision Tree, Support Vector Classifier, Naive Bayes

ACM Reference Format:
Victor Luis Collado Ureta, CWID: A20522008 and Daniel Pizarro Caro, CWID: A20521462. 2023. Comparison of algorithms in Amazon reviews sentiment analysis. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (IL).* ACM, New York, NY, USA, 7 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Sentiment analysis, also known as opinion mining, is a subfield of natural language processing that has gained significant attention in recent years due to the explosion of social media and online review platforms. It involves analyzing and categorizing the sentiments expressed in a text, whether positive, negative, or neutral. With the increasing availability of large datasets and advancements in machine learning techniques, sentiment analysis has become more sophisticated, enabling businesses to gain insights into customer attitudes and opinions on their products or services.

In this paper, we investigate the performance of machine learning methods in sentiment analysis. We explore various feature extraction techniques, and compare the performance of different classifiers, such as Logistic Regression, Naive Bayes, Decision Trees, Support Vector Machine and KNN. Our study aims to provide insights into the strengths and limitations of machine learning methods in sentiment analysis and to identify best practices for developing accurate and reliable sentiment analysis models.

2 PROBLEM DESCRIPTION

Amazon is currently one of the world's leading online shopping stores. There are many people who use Amazon as a store to buy specific items and others who use it as a trusted store on a daily basis. Some of these users tend to leave their opinion about the products through reviews, which can be positive, negative or even neutral. With so much information it is difficult to analyze whether a product or several products are worth buying or not based on this opinions. In order to solve this problem, this project aims to perform a sentiment analysis on Amazon reviews.

2.1 Description of the Data

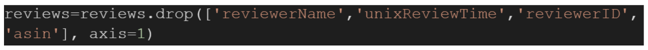
The dataset chosen for this project consists of a series of musical instrument reviews. It is a table with 9 columns where each row corresponds to a user and his evaluation of a specific instrument. The descriptions of each of the columns are as follows[1]:

- `reviewerID`: Contains the ID of the user making the review.
- `asin`: Contains the ID of the instrument.
- `reviewerName`: Contains the nickname of the reviewer.
- `helpful`: Helpfulness rating of the review.
- `reviewText`: Contains the text of the review.
- `overall`: Rating of the product.
- `summary`: A brief summary of the review.
- `unixReviewTime`: Time and date of the review in UNIX format.
- `reviewTime`: Time and date of the review.

3 DATA PRE-PROCESSING

Prior to any analysis, it is necessary to carry out the task of data pre-processing of the original dataset[1].

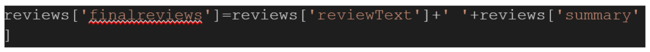
Firstly, we drop all the unnecessary features from the dataset.



```
reviews=reviews.drop(['reviewerName','unixReviewTime','reviewerID',
'asin'], axis=1)
```

Figure 1: Drop unnecessary features

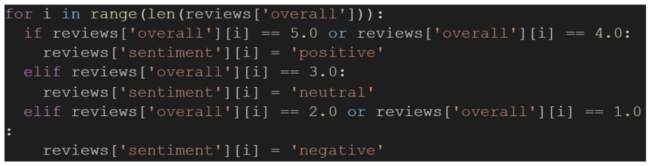
The next step is putting together the text from the review and the text from the summary of the review to get all the possible information about each review for the analysis.



```
reviews['finalreviews']=reviews['reviewText']+' '+reviews['summary'
]
```

Figure 2: Join review and summary of the review texts

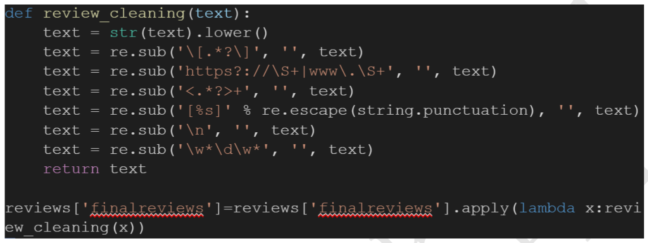
Then, we need to put the target variable in the correct format. Since Amazon rates the products from 1.0 to 5.0 stars, we decide that if the rating is 1.0 or 2.0 stars, we consider it negative. If the rating is 3.0 stars, we consider it neutral. Finally, if the rating is 4.0 or 5.0, we consider it positive.



```
for i in range(len(reviews['overall'])):
  if reviews['overall'][i] == 5.0 or reviews['overall'][i] == 4.0:
    reviews['sentiment'][i] = 'positive'
  elif reviews['overall'][i] == 3.0:
    reviews['sentiment'][i] = 'neutral'
  elif reviews['overall'][i] == 2.0 or reviews['overall'][i] == 1.0
:
    reviews['sentiment'][i] = 'negative'
```

Figure 3: Rating intervals

The next step is to make a general cleaning of the text such as making everything lowercase, removeing links or removing punctuation.

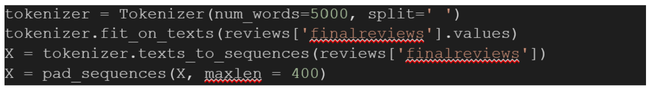


```
def review_cleaning(text):
    text = str(text).lower()
    text = re.sub('\[.*?\]', '', text)
    text = re.sub('https?://\S+|www\.\S+', '', text)
    text = re.sub('<.*?>+', '', text)
    text = re.sub('[%s]' % re.escape(string.punctuation), '', text)
    text = re.sub('\n', '', text)
    text = re.sub('\w*\d\w*', '', text)
    return text

reviews['finalreviews']=reviews['finalreviews'].apply(lambda x:revi
ew_cleaning(x))
```

Figure 4: Removing punctuation

After that, we need to convert the text into numerical sequences that can be fed into the neural network.



```
tokenizer = Tokenizer(num_words=5000, split=' ')
tokenizer.fit_on_texts(reviews['finalreviews'].values)
X = tokenizer.texts_to_sequences(reviews['finalreviews'])
X = pad_sequences(X, maxlen = 400)
```

Figure 5: Text into numerical sequences

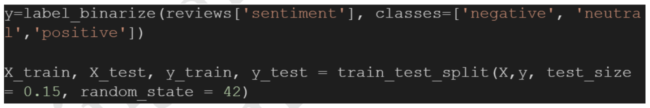
The Tokenizer class tokenizes the text by splitting it into individual words, and then maps each word to a unique integer index. The num_words parameter specifies the maximum number of words to keep, based on word frequency. In this case, it is set to 5000, which means that only the top 5000 most frequent words will be kept, and all other words will be discarded.

The fit_on_texts method of the Tokenizer class is then used to fit the tokenizer on the preprocessed reviews in the finalreviews column of the reviews dataframe. This method updates the internal vocabulary of the tokenizer based on the frequency of words in the reviews.

The texts_to _sequences method is then used to convert the preprocessed reviews into numerical sequences based on the tokenizer's vocabulary. This method replaces each word in the reviews with its corresponding integer index in the vocabulary.

Finally, the pad_sequences function from Keras is used to pad all the sequences to a fixed length of 400. This is necessary because neural networks typically require inputs to be of the same length. The maxlen parameter specifies the maximum length of the sequences, and any sequences shorter than this length will be padded with zeros at the end to make them the same length as the longest sequence. The resulting X matrix contains the numerical sequences of the reviews, with each row representing a single review and each column representing a word in the review.

Finally, the last step of the pre-processing is to one-hot encode the target variable and split the dataset into training/test samples.



```
y=label_binarize(reviews['sentiment'], classes=['negative', 'neutra
l','positive'])

X_train, X_test, y_train, y_test = train_test_split(X,y, test_size
= 0.15, random_state = 42)
```

Figure 6: Text into numerical sequences

4 PRIOR ANALYSIS OF THE DATA

We have taken the chosen dataset and created a series of preliminary graphs to explain how the information is distributed and what data, patterns can be observed prior to analysis.

4.1 Rating Distribution

This plot shows the number of each rating in the dataset. As you can see, most of the people have rated the products with 5 stars. Therefore, the majority of the reviews will have positive sentiment.

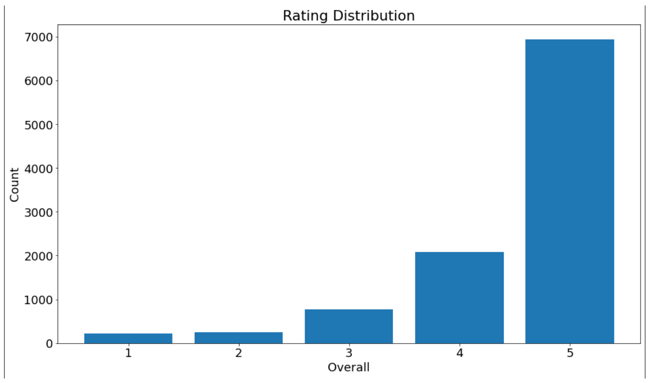


Figure 7: Rating Distribution

4.2 Reviews Helpfulness

This violin plot shows, for each sentiment, the helpful rate of each review. The helpful rate goes from 0 to 1 and it is computed dividing the number of users that found the review helpful by the number of users that rated the helpfulness of the review. As you can see in the plot, the most helpful reviews are the positive ones.

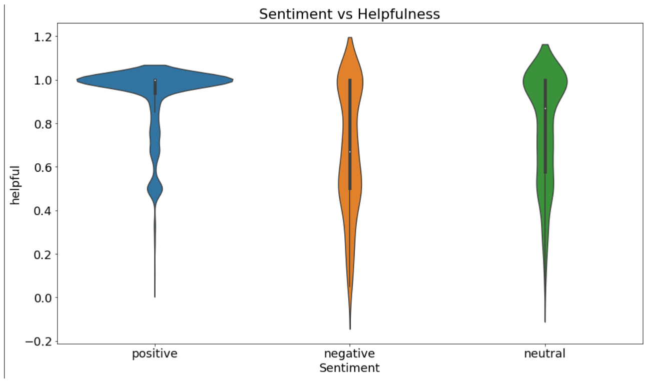


Figure 8: Sentiment vs Helpfulness

4.3 Ngram analysis

We did an analysis of the unigrams, bigrams and trigrams that most appear in positive, negative and neutral reviews. It is important to mention that for this analysis we have dropped the stopwords. Stopwords are words that appear a lot in common text and do not provide any information about its sentiment. An example of stopwords could be ‘a’, ‘me’, ‘the’, and a lot more.

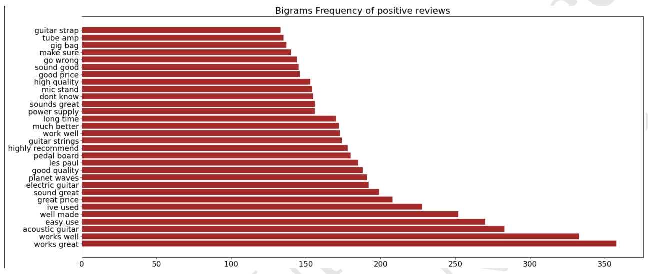


Figure 9: Bigrams frequency of positive reviews

In the plot you can see the frequency of bigrams in positive reviews. The two words that most appear together are works great and works well. Also good quality, sounds great and easy used are mentioned a lot in positive reviews. However, you can see bigrams such as ‘go wrong’ that do not seem to fit here. These words appear because they are most of the time followed by a negative word such as ‘nothing go wrong’ giving them a positive connotation.

4.4 Word Cloud

A word cloud consists of a set of words that are arranged in a visually appealing way, with the size of each word indicating its frequency or importance in the text. The most frequent words appear larger than the less frequent ones. With this plot, we can easily see what are the words most frequently used in positive, negative and neutral reviews.

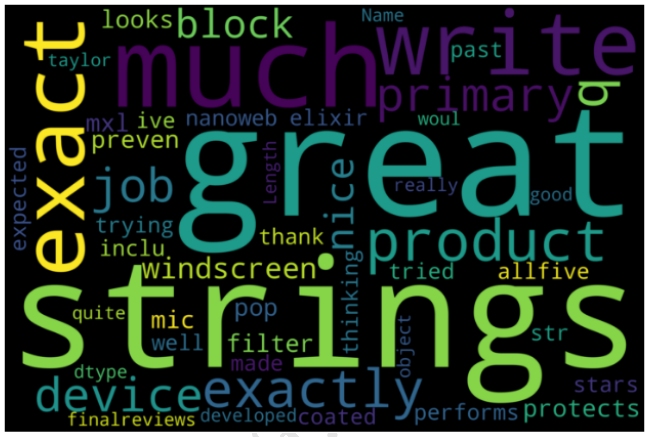


Figure 10: Word cloud

5 ANALYSIS OF THE DATA

Once the previous step of Data pre-processing was finished, the next step was to carry out the analysis. For this purpose, it was decided to use two alternative methods:

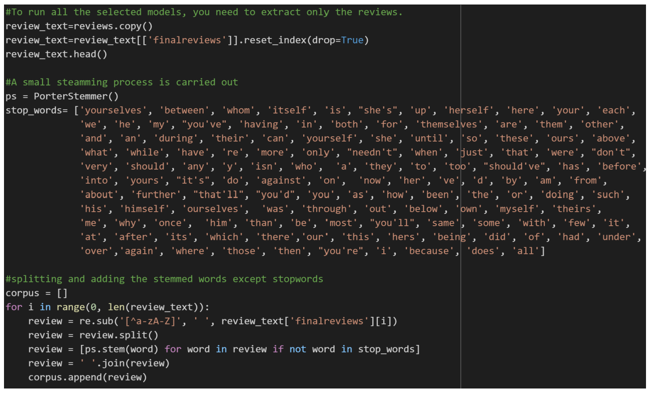
(1) On the one hand, the data were analyzed using 5 different models: Logistic Regression, Naive Bayes test, Decision Trees, Support Vector Machine and KNN.
(2) On the other hand, the data were analyzed using a Recurrent Neural Network.

5.1 Analysis using models

Several models have been chosen for the analysis of the data:

- Logistic Regression is a supervised learning algorithm that is used to model the probability of a binary response variable, given a set of independent variables.[2]
- Naive Bayes is a probabilistic machine learning algorithm that is used for classification tasks. It is based on Bayes’ theorem, which states that the probability of a hypothesis (in this case, a classification) can be updated given new evidence.[3]
- Decision Trees are a popular type of machine learning algorithm that is used for both classification and regression tasks. They are based on a tree-like model, where each internal node represents a test on a feature, each branch represents the outcome of the test, and each leaf node represents a class label or a numerical value.[4]
- Support Vector Machines (SVMs) are a type of supervised learning algorithm that can be used for both classification and regression tasks. SVMs aim to find the best hyperplane that separates the data points into different classes.[5]
- K-Nearest Neighbors (KNN) is a non-parametric machine learning algorithm that is used for classification and regression tasks. KNN is based on the idea that similar data points tend to belong to the same class. The algorithm finds the K nearest data points to a given data point and assigns the majority class label to it.[6]

In order to carry out the analysis using the models, a previous Stemming step had to be performed. The main objective is to get the root of all the words in the reviews.



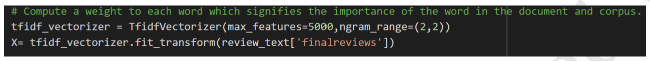
```
#To run all the selected models, you need to extract only the reviews.
review_text=reviews.copy()
review_text=review_text[['finalreviews']].reset_index(drop=True)
review_text.head()

#A small steamming process is carried out
ps = PorterStemmer()
stop_words= ['yourselves', 'between', 'whom', 'itself', 'is', "she's", 'up', 'herself', 'here', 'your', 'each',
             'we', 'he', 'my', "you've", 'having', 'in', 'both', 'for', 'themselves', 'are', 'them', 'other',
             'and', 'an', 'during', 'their', 'can', 'yourself', 'she', 'until', 'so', 'these', 'ours', 'above',
             'what', 'while', 'have', 're', 'more', 'only', "needn't", 'when', 'just', 'that', 'were', "don't",
             'very', 'should', 'any', 'y', 'isn', 'who',  'a', 'they', 'to', 'too', "should've", 'has', 'before',
             'into', 'yours', "it's", 'do', 'against', 'on',  'now', 'her', 've', 'd', 'by', 'am', 'from',
             'about', 'further', "that'll", "you'd", 'you', 'as', 'how', 'been', 'the', 'or', 'doing', 'such',
             'his', 'himself', 'ourselves',  'was', 'through', 'out', 'below', 'own', 'myself', 'theirs',
             'me', 'why', 'once',  'him', 'than', 'be', 'most', "you'll", 'same', 'some', 'with', 'few', 'it',
             'at', 'after', 'its', 'which', 'there','our', 'this', 'hers', 'being', 'did', 'of', 'had', 'under',
             'over','again', 'where', 'those', 'then', "you're", 'i', 'because', 'does', 'all']

#splitting and adding the stemmed words except stopwords
corpus = []
for i in range(0, len(review_text)):
    review = re.sub('[^a-zA-Z]', ' ', review_text['finalreviews'][i])
    review = review.split()
    review = [ps.stem(word) for word in review if not word in stop_words]
    review = ' '.join(review)
    corpus.append(review)
```

Figure 11: Stemming Process

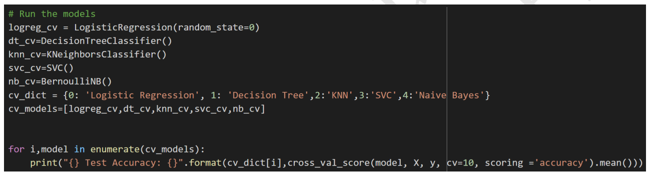
The next step is to use the TFIDF technique. This method quantifies words in documents by assigning each word a weight that represents its significance to the document and the corpus as a whole. In this case, we break the sentence into two bigrams and weigh them individually.We are just using the top 5000 words from the reviews as well.



```
# Compute a weight to each word which signifies the importance of the word in the document and corpus.
tfidf_vectorizer = TfidfVectorizer(max_features=5000,ngram_range=(2,2))
X= tfidf_vectorizer.fit_transform(review_text['finalreviews'])
```

Figure 12: TFIDF technique

Finally, in order to see which model works best, we used cross validation and perform the model selection process.

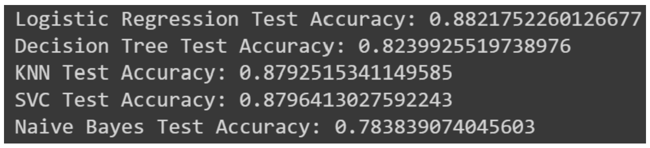


```
# Run the models
logreg_cv = LogisticRegression(random_state=0)
dt_cv=DecisionTreeClassifier()
knn_cv=KNeighborsClassifier()
svc_cv=SVC()
nb_cv=BernoulliNB()
cv_dict = {0: 'Logistic Regression', 1: 'Decision Tree',2:'KNN',3:'SVC',4:'Naive Bayes'}
cv_models=[logreg_cv,dt_cv,knn_cv,svc_cv,nb_cv]


for i,model in enumerate(cv_models):
    print("{} Test Accuracy: {}".format(cv_dict[i],cross_val_score(model, X, y, cv=10, scoring ='accuracy').mean()))
```

Figure 13: Models execution

The results show that logistic regression outperformed the other algorithms, and almost all of the accuracy levels are more than 80%.



```
Logistic Regression Test Accuracy: 0.8821752260126677
Decision Tree Test Accuracy: 0.8239925519738976
KNN Test Accuracy: 0.8792515341149585
SVC Test Accuracy: 0.8796413027592243
Naive Bayes Test Accuracy: 0.783839074045603
```

Figure 14: Models accuracy

5.2 Analysis using RNN

Recurrent Neural Networks (RNNs) are a type of neural network that is particularly well-suited for processing sequential data, such as natural language. Unlike feedforward neural networks, which process each input independently, RNNs maintain a hidden state that is updated with each new input. This hidden state allows the network to take into account the context of each input in the sequence, making them ideal for tasks like sentiment analysis, where the sentiment of a piece of text is dependent on the sequence of words.

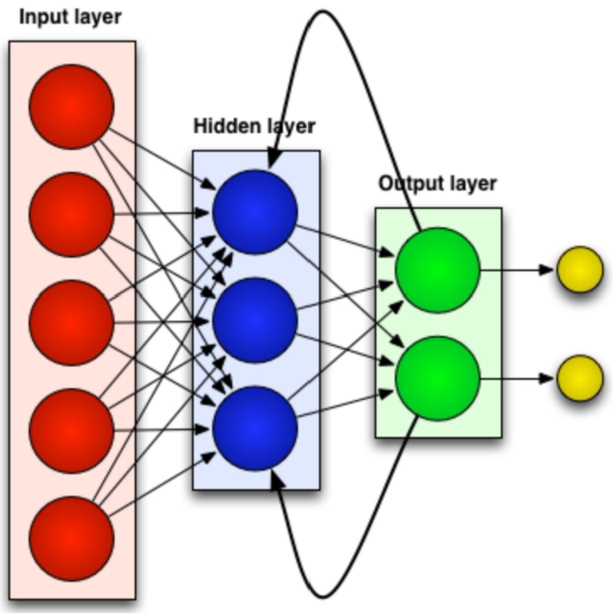


Figure 15: RNN structure

RNNs can be used for a wide variety of tasks, including machine translation, speech recognition, and image captioning. However, in the context of sentiment analysis, the goal is to classify a piece of text as positive, negative, or neutral based on the sentiment expressed in the text.

The process of using RNNs for sentiment analysis typically involves training the network on a dataset of labeled examples, where each example consists of a piece of text and its associated sentiment label. During training, the RNN learns to recognize patterns in the text that are indicative of positive or negative sentiment. Once trained, the RNN can be used to classify the sentiment of new, unseen text.[7]

There are several different types of RNNs, each with its own strengths and weaknesses. One of the most commonly used RNN architectures for sentiment analysis is the Long Short-Term Memory (LSTM) network. LSTMs are a type of RNN that are designed to address the problem of vanishing gradients, which can occur when training RNNs on long sequences of data. LSTMs use a series of gates to control the flow of information into and out of the hidden state, allowing them to remember or forget information as needed.

In addition to LSTMs, there are other types of RNNs that can be used for sentiment analysis, such as Gated Recurrent Units (GRUs) and Simple Recurrent Units (SRUs). Each of these architectures has its own unique properties, and the choice of architecture will depend on the specific requirements of the task at hand.

When using RNNs for sentiment analysis, it is important to preprocess the input text to ensure that it is in a suitable format for the network to process. This typically involves tokenizing the text into individual words or sub-words, and converting them into a numerical representation that can be fed into the network.

In order to improve the performance of RNNs for sentiment analysis, several techniques can be employed. One such technique is the use of pre-trained word embeddings, which are numerical representations of words that have been learned from large amounts of text data. These pre-trained embeddings can be used to initialize the weights of the RNN, allowing it to leverage the knowledge contained within the embeddings to improve its performance.

5.3 RNN Design

For this project, the type of RNN we decided to build is a LSTM. LSTM is a type of neural network that is good at understanding the meaning of a series of words. It has gates that control what information is important to remember and what can be forgotten. This is useful for sentiment analysis, which looks at the emotions conveyed by words.

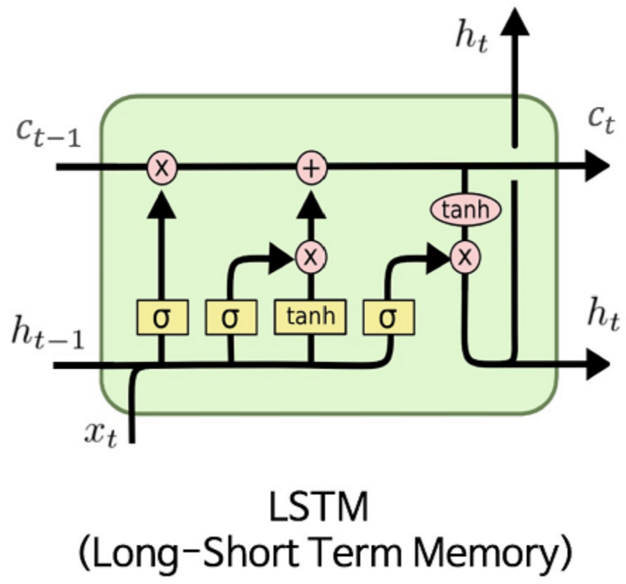


Figure 16: LSTM structure

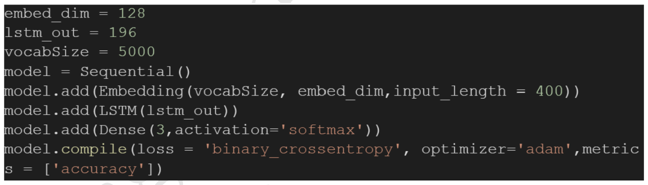
LSTM works by taking a sequence of inputs, like words, and processing them one at a time. As it processes each input, it remembers what has come before so it can understand the context of each word.

To use LSTM for sentiment analysis, you need a dataset of text that is labeled with whether it has positive or negative sentiment. The network is trained to recognize patterns in the text that are associated with positive or negative sentiment. Once trained, it can be used to analyze new text and determine if it has a positive or negative sentiment.

LSTMs are better than traditional neural networks for sentiment analysis because they can handle long sequences of text without forgetting important information. They can also selectively remember or forget information from previous inputs, so they can focus on what is important for the task at hand. Finally, LSTMs can be trained on large datasets, which is important for achieving high accuracy in sentiment analysis.[8]

5.4 RNN Architecture

To develop the RNN we follow the following architecture:



```
embed_dim = 128
lstm_out = 196
vocabSize = 5000
model = Sequential()
model.add(Embedding(vocabSize, embed_dim,input_length = 400))
model.add(LSTM(lstm_out))
model.add(Dense(3,activation='softmax'))
model.compile(loss = 'binary_crossentropy', optimizer='adam',metric
s = ['accuracy'])
```

Figure 17: RNN architecture

- **Input Layer**: It takes in sequences of words and transforms them into fixed-size dense vectors using an embedding layer with a vocabulary size of 5000 and an embedding dimension of 128.
- **LSTM Layer**: It processes the output from the embedding layer in a sequential manner to capture long-term dependencies in the input sequence. This layer has 196 LSTM units.
- **Output Layer**:It produces the probabilities for each of the three classes of a binary classification problem, using a dense layer with the softmax activation function.
- **Loss Function**: It measures the difference between the predicted probabilities and the actual classes during training. The binary cross-entropy loss function is used for this model.
- **Optimizer**: It updates the weights of the model during training to minimize the loss. The Adam optimizer is used in this architecture
- **Metrics**: It evaluates the performance of the model during training and testing. The accuracy metric is used in this architecture.

To sum up, this architecture is designed to process input sequences of length 400 and classify them into one of three classes, using an LSTM layer for sequential processing, a dense output layer for classification, and the Adam optimizer for efficient learning.[8]

5.5 Results

5.5.1 Unbalanced dataset. : Running the RNN for a few epochs we got the following results:

- VALIDATION ACCURACY: 0.891
- TESTING ACCURACY: 0.875

These accuracies are a slight improvement with respect to the previous models. In the next picture you can see a confusion matrix of the model predictions

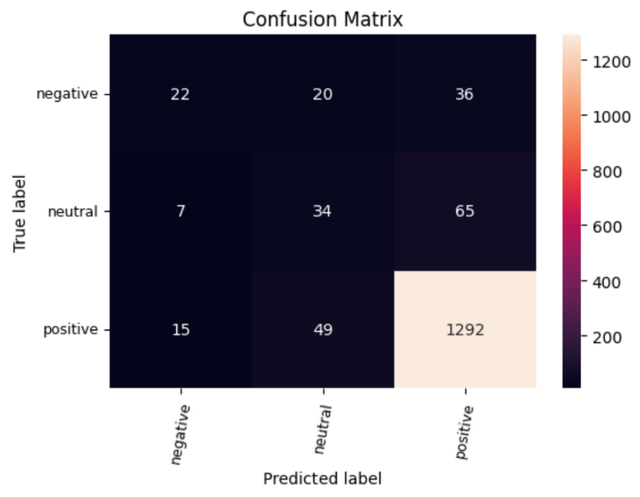


Figure 18: Confusion matrix for unbalanced data

It can be seen that both the neutral and the negative classes are badly predicted. Also, it can be view that the classes are unbalanced since the positive class is incredibly large compared to the other two classes.

The next image shows the ROC curve.

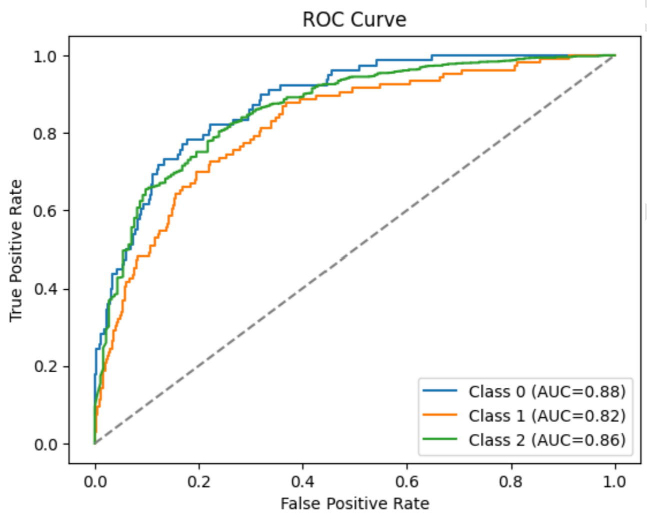
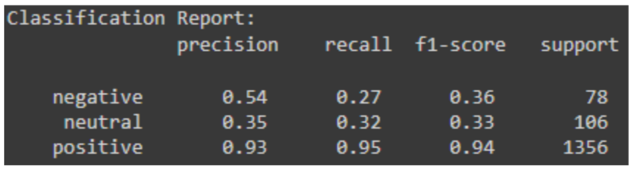


Figure 19: ROC Curve

Lastly, in the following picture it can be seen the classification report

The precision represents how the model performs on predicting positive results for each class. This means that for both negative and neutral classes the precision results are low because they are falsely predicted a lot of times compared to the times they are correctly predicted.

The recall represents how many of the actual positives our model is able to predict. This means that also in both negative and neutral classes the recall results are low because our model is correctly



```
Classification Report:
              precision    recall  f1-score   support

    negative       0.54      0.27      0.36        78
     neutral       0.35      0.32      0.33       106
    positive       0.93      0.95      0.94      1356
```

Figure 20: Classification report for unbalanced data

predicting approximately 25% of the total negative and neutral classes.

The F1 score represents the balance between precision and recall. So, in this case F1 score is also low for both negative and neutral classes.

The fact that the dataset is unbalanced might be a real problem since the model struggles a lot to predict negative sentiment cases. If you are Amazon, you would be interested to correctly capture the negative reviews in order to learn from them and fix the problems. What can we do to fix this? You might have realized that the dataset is so unbalanced. In the next part, we are using the SMOTE python library to balance the classes and try to recalculate these values to see the real performance of the recurrent neural network.[9]

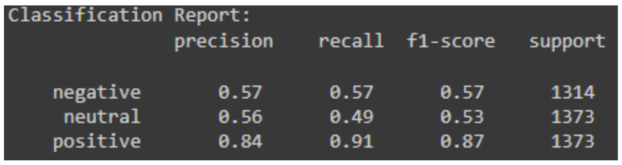
5.5.2 Balanced dataset. As we mentioned, in this part we are using the SMOTE library to equal the samples of negative, neutral, and positive sentiments. The SMOTE (Synthetic Minority Oversampling Technique) library is used to balance the imbalanced dataset by oversampling the minority class. In the given code, the target variable 'y' contains three classes: 'negative', 'neutral', and 'positive'. The Counter() function can be used to count the number of samples in each class.

SMOTE synthesizes new minority instances between existing minority instances. It generates the virtual training records by linear interpolation for the minority class. These synthetic training records are generated by randomly selecting one or more of the k-nearest neighbors for each example in the minority class. After the oversampling process, the data is reconstructed and several classification models can be applied for the processed data.

After running the balanced model, the results are the following:

- VALIDATION ACCURACY: 0.734
- TESTING ACCURACY: 0.663

If we look at the classification report



```
Classification Report:
              precision    recall  f1-score   support

    negative       0.57      0.57      0.57      1314
     neutral       0.56      0.49      0.53      1373
    positive       0.84      0.91      0.87      1373
```

Figure 21: Classification record for balanced data

It can be seen that there has been an increase in both precision and recall for the negative and neutral classes. Analyzing both the balanced and the unbalanced dataset, it can be appreciated that the tradeoff between a good global accuracy and a good performance in the negative and neutral class.

However, if we look at the confusion matrix.

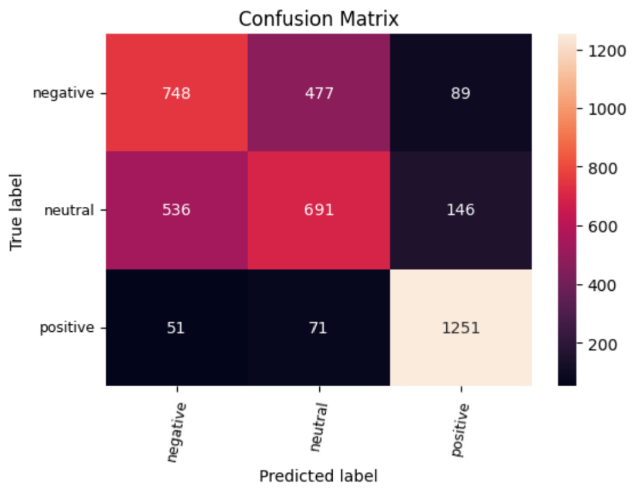


Figure 22: Confusion matrix for balanced data

Here, it can be seen that there has been a substantial decrease in the testing accuracy. This is because the way the SMOTE library makes the extra samples that are not captured so well by the model.

6 CONCLUSIONS

Through this comparison of analysis methods, it is evident that both methods offer valuable insights into sentiment analysis. Machine learning models, with their ability to classify sentiments based on various features, provide a reliable and interpretable approach. On the other hand, RNNs, with their sequential analysis of textual data, capture the temporal dependencies and contextual information more effectively. While machine learning models excel in extracting important features from reviews, RNNs showcase their strength in understanding the sentiment expressed throughout the text.[10]

REFERENCES

[1] B. Roshan, "Sentiment analysis | amazon reviews." https://www.kaggle.com/code/benroshan/sentiment-analysis-amazon-reviews#Model-Building:-Sentiment-Analysis, 2020.
[2] S. Swaminathan, "Logistic regression — detailed overview." https://towardsdatascience.com/logistic-regression-detailed-overview-46c4da4303bc, 2018.
[3] R. Gandhi, "Naive bayes classifier." https://towardsdatascience.com/naive-bayes-classifier-81d512f50a7c, 2018.
[4] C. Bento, "Decision tree classifier explained in real-life: picking a vacation destination." https://towardsdatascience.com/decision-tree-classifier-explained-in-real-life-picking-a-vacation-destination-6226b2b60575, 2021.
[5] R. Gandhi, "Support vector machine — introduction to machine learning algorithms." https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47, 2018.
[6] O. Harrison, "Machine learning basics with the k-nearest neighbors algorithm." https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761, 2018.
[7] S. Li, "A beginner's guide on sentiment analysis with rnn." https://towardsdatascience.com/a-beginners-guide-on-sentiment-analysis-with-rnn-9e100627c02e, 2018.
[8] A. K. T. Alpna Patel, "Sentiment analysis by using recurrent neural network." https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3349572#:~:text=RNN%20model%20is%20used%20to,valid%20data%20for%20test%20classifier, 2019.
[9] K. P. Shung, "Accuracy, precision, recall or f1?." https://towardsdatascience.com/accuracy-precision-recall-or-f1-331fb37c5cb9, 2018.
[10] V. L. C. U. Daniel Pizarro Caro, "Project github." https://github.com/Danipizarro04/Amazon-Reviews-Sentimental-Analysis, 2023.