Amazon Reviews: Sentiment Analysis

AAI 501 Final Project
- University of San Diego
- Yash Chaturvedi

This project is a part of AAI 501 Introduction to Artificial Intelligence offered by University of San Diego. In this project, we aim to perform sentiment analysis on Amazon product reviews for the Home Improvement category. We will be using natural language processing (NLP) techniques and machine learning algorithms to classify each review as either positive or negative.

The results of this project can be used to understand customer sentiment towards products in the Home Improvement category on Amazon and provide insights for companies to improve their products and services.

Abstract

The purpose of this project is to develop a sentiment analysis system for Amazon reviews on home improvement products. The goal of this project is to accurately classify reviews as positive or negative based on the text content of the review.

The scope of this project includes pre-processing the text data, applying different techniques such as bag-of-words, n-gram models, and deep learning models like different versions of RNNs, and evaluating the performance of these models using cross-validation and accuracy metrics.

To develop this sentiment analysis system, I relied on research papers on sentiment analysis techniques and algorithms, including "Bag of Words Meets Bags of Popcorn" by Andrew L. Maas et al., "Recurrent Neural Networks for Sentiment Analysis of Short Texts" by Guenter Klambauer et al.

In this project, I adapted published algorithms for sentiment analysis and applied them to the Amazon reviews dataset. I addressed several problems and issues, including text pre-processing, feature selection, model selection, and hyperparameter tuning.

These problems and issues are related to the techniques and ideas covered in the Natural Language Processing course, including text pre-processing, classification algorithms, and deep learning models for natural language processing.

Overall, this project aimed to provide a practical example of sentiment analysis applied to real-world data and to demonstrate the effectiveness of different techniques and algorithms for sentiment analysis.

About the Data

The dataset used in this project consists of Amazon customer reviews for Home Improvement products sold in the United States. The data was obtained from the Amazon Customer Reviews Dataset, which is publicly available on the Amazon Web Services (AWS) website. The dataset contains over 1.6 million reviews collected between May 1996 and July 2014. Each review includes the product ID, the reviewer ID, the review text, the rating given by the reviewer (on a scale of 1 to 5), and other metadata such as the date the review was written and whether or not the reviewer is a verified purchaser of the product.

What data we are using

We are accessing reviews data of the Amazon 'Home Improvement' category for customers in the U.S. To get more details about this data set from visit here:
https://s3.amazonaws.com/amazon-reviews-pds/tsv/index.txt

The files are available in a .tsv format and we will download -

amazon_reviews_us_Home_Improvement - contains the reviews.
For this project we will be working with the reviews file. The file will be attached to the project as well.

The Dataset

These are the available columns in the dataset -

- marketplace: a two-letter country code of the marketplace where the review was written
- customer_id: a random identifier that can be used to aggregate reviews written by a single author
- review_id: the unique ID of the review
- product_id: the unique product ID the review pertains to. In the multilingual dataset, the reviews for the same product in different countries can be grouped by the same product_id
- product_parent: a random identifier that can be used to aggregate reviews for the same product
- product_title: title of the product
- product_category: broad product category that can be used to group reviews (also used to group the dataset into coherent parts)
- star_rating: the 1-5 star rating of the review
- helpful_votes: number of helpful votes
- total_votes: number of total votes the review received
- vine: review was written as part of the Vine program
- verified_purchase: the review is on a verified purchase
- review_headline: the title of the review
- review_body: the review text
- review_date: the date the review was written

Data Preparation

The file is available as a .tsv format and after downloading it, and opening it while creating a pandas dataframe, we find that some records have metadata in more than the 15 columns named above. We get an error message -
"ParserError: Error tokenizing data. C error: Expected 15 fields in line 7207, saw 22" which usually occurs when there is a formatting issue with the input data file. Hence, we make sure that we are selecting only the first 15 columns.

For the purpose of text classification and sentiment analysis, we will work with only a few relevant columns to this specific problem. Here are the relevant columns and how they can be used -

review_id: A unique identifier for each review. This is useful for tracking and analysis purposes.
review_body: The text of the review itself. This is the main input data for the text classification and sentiment analysis tasks.
star_rating: The rating given by the reviewer, from 1 to 5 stars. This can be used to generate a sentiment label for each review, where ratings of 4 or 5 stars are typically considered positive and ratings of 1 to 3 stars are considered negative.
verified_purchase: A boolean indicating whether the reviewer purchased the product they are reviewing. This can be used to filter out potentially biased or fake reviews.

There are other optional columns we could have used to filter out the reviews themselves based on the usefulness by working with the following columns -

product_category: The broad category of the product being reviewed. This can be useful for analyzing sentiment and text patterns across different types of products. Though as we are considering products only from a certain category we can exclude this.
total_votes: indicates the total number of votes that a review has received and can be useful for filtering out reviews with very few votes, which may not be representative of the broader customer sentiment.
helpful_votes: indicates the number of votes that a review received indicating that it was helpful. This can be used to filter for reviews that are more likely to be informative and useful.

In this project though, we will work with all the reviews and not filter them out based on votes.

Data Cleaning

Nulls, Missing values, Erroneous Data, Duplicate Rows, Matching data types.

We perform the following steps to perform a generic data cleaning:
1. Remove any missing or null values
2. Remove any duplicates
3. Remove any irrelevant or erroneous data
4. Convert relevant columns to the correct data types
5. Perform any additional data cleaning steps specific to your task.

The process results in the following filtered data -
1. We remove any rows that contain missing or null values using the dropna() method of the DataFrame.
2. We remove any duplicate rows based on the review_id column using the drop_duplicates() method of the DataFrame.
3. We remove any rows that contain irrelevant or erroneous data. In this case, we keep only the rows that have a star_rating value between 1 and 5, and a verified_purchase value of either 'Y' or 'N'.
4. We convert the verified_purchase column to a boolean data type by mapping 'Y' to True and 'N' to False, and the review_body column to a string data type using the astype() method of the DataFrame.
5. We also see inconsistent values in the column ‘star_rating’ and therefore to fix this inconsistency in star_rating column, we use the astype() method to convert the column to integer type.
6. To make the computations faster, and spend less computing power, we will also just work with 100000 records sampled randomly. To reproduce the same result, we will assign a random_seed = ‘42’.

Tokenization & Stemming

Tokenization involves breaking the text into individual words or tokens, while stemming involves reducing words to their root form (e.g. "running" and "runner" are both stemmed to "run"). These steps can help reduce the number of unique words in the dataset and improve the accuracy of the model. We also implement these techniques to preprocess the raw text data and transform it into a format that can be used for further analysis or modeling.

Exploratory Data Analysis

We plot the star rating for verified purchase vs non verified purchases. As we can see (**Figure 1**) shows that the mean for verified purchases is higher than that of non verified purchases.

We also produce a box plot (**Figure 2**) showing the distribution of star ratings for verified and unverified purchases. The x-axis shows the two groups, and the y-axis shows the star ratings. The box plot displays the median (line in the box), the quartiles (box), and the outliers (dots outside the box) for each group. This also shows that we have outliers for verified purchases.

In text classification, outliers may refer to reviews that are unusually long, short, or have extreme ratings. It's important to carefully consider the impact of removing outliers before doing so. Removing too many reviews or removing reviews that are relevant to the task may reduce the quality of the model. On the other hand, including outliers may introduce noise and affect the performance of the model. For the scope of this project, we will consider these outliers to be relevant and therefore, not exclude any outliers.

A frequency distribution is a way of organizing and displaying data in a tabular form to show the number of times a particular value or range of values occurs in a dataset. In other words, it represents how often each value or range of values appears in a dataset. We also plot the distribution of Star Ratings (**Figure 3**). This can provide insights into how customers rate the products in the given category. This can help identify trends, such as a higher proportion of 5-star ratings or a higher proportion of 1-star ratings.

Models

These are the following models we work with -

- Bag-of-words model with Naive Bayes
- N-gram model with Naive Bayes
- Recurrent Neural Network (RNN) with word embeddings (GRU)
- Recurrent Neural Network (RNN) with word embeddings (with LSTM Layers)

Language Models

Bag-of-Words Model

A naive Bayes model based on the presence of specific words could reliably classify sentences into categories. The application of naive Bayes to strings of words is called the **bag-of-words model**. Though it falsely assumes that each word is independent of the others, and therefore it does not generate coherent English sentences, it does allow us to do classification with good accuracy using the naive Bayes formula.

We use the bag-of-words model to classify the reviews as positive or negative based on the star_rating column. We also use the star_rating column to generate sentiment labels where ratings of 4 or 5 stars are considered positive and ratings of 1 to 3 stars are considered negative.

Feature Selection

We do feature selection where we create a vector for each review using term frequency - with a value of the number of occurrences of each word in the review. We limit the max number of features to 5000 limiting ourselves to a subset of the words as features. We also drop words that are very rare (and thus have high variance in their predictive powers), as well as words that are common to all classes (such as “the”) but don’t discriminate between classes.

Create Vectors using term frequency

We use the CountVectorizer class from sklearn to create a vector for each essay. We can't use text directly with machine learning; we need to create a vector of numbers first. The CountVectorizer creates a vector with one position for each word in the corpus with a value of the number of occurrences of that word in the essay.

We create a CountVectorizer object named vectorizer with the following parameters:
1) max_features=5000: This sets the maximum number of features to be extracted to 5000, which means that only the 5000 most frequent words will be used to create the bag-of-words model.
2) stop_words='english': This tells the vectorizer to remove all English stop words (such as "the", "and", "a", etc.) from the text data, as these words are generally not informative for text classification tasks.
3) min_df=0.001: This sets the minimum document frequency to 0.001, which means that any word that appears in less than 0.1% of the documents will be excluded from the vocabulary.

By passing these parameters to the CountVectorizer constructor, we are creating a bag-of-words model that is optimized for text classification tasks

Naive Bayes

MultinomialNB is used to train a Naive Bayes model on the bag-of-words features obtained from the preprocessed review text. After the model is trained, it is used to predict the sentiment of the test data, and the accuracy of the predictions is evaluated.

We calculate the accuracy of the classifier by comparing the predicted sentiments with the actual sentiments.

Accuracy = 0.03985 = 4%

The accuracy score for the model is very low, only 0.03985, which indicates poor performance. This means that the model is not able to accurately predict the sentiment of the reviews based on the text data.
There could be various reasons why the accuracy of the model is low -

Data imbalance: If the number of positive and negative reviews in the dataset is balanced. If the dataset has more reviews of one class than the other, the model may be biased towards the majority class, resulting in lower accuracy. As we saw in (**Figure 3**), this potentially could be the issue.
Model complexity: The model may not be complex enough to capture the nuances of the language in the reviews. In such cases, we may try using more sophisticated models like neural networks or ensembles of models.
Hyperparameters: The hyperparameters of the model may not be set optimally. We can try tuning the hyperparameters using techniques like grid search or random search.

Cross-validation: Bag-of-Words
Cross-validation is performed to evaluate the performance of a machine learning model on a dataset. It involves dividing the dataset into multiple subsets, or folds, and training the model on a combination of these subsets while testing it on the remaining fold. This process is repeated multiple times, with each fold serving as the test set at least once, and the performance metrics are averaged across all the folds.

Evaluation
We performed cross-validation on our model and obtained an average accuracy score of 0.83468. The reason why we observe a high average cross-validation score, but a low accuracy score on the testing set is due to overfitting.

During training, the model learns to fit the training data very well, but may fail to generalize to new data. This means that the model may perform very well on the data that it has already seen, but poorly on new data. Cross-validation is a technique that can help to detect overfitting by splitting the data into

multiple training and testing sets and then evaluating the performance of the model on each set. By taking the average score across multiple sets, we can get a more reliable estimate of how well the model is likely to perform on new, unseen data. In contrast, when we evaluate the model on a single testing set, we are essentially testing the model's ability to generalize to new data. If the model is overfitting, it may perform poorly on the testing set, leading to a low accuracy score which is the case.

Limitations: Bag-of-Words

The bag-of-words model has several limitations, including:

1) Loss of context: The model treats each word in the text as an independent feature, without considering the context in which the word appears. This can lead to the loss of important information and nuances in the text.

2) No semantic understanding: The model does not have any semantic understanding of the text. It simply counts the occurrence of words in the text and uses that as a basis for classification. This can result in incorrect classification of text with similar words but different meanings.

3) High dimensionality: The bag-of-words model can result in high-dimensional feature vectors, particularly when dealing with large datasets. This can make the model computationally expensive and difficult to train.

4) Handling of out-of-vocabulary words: The model requires that all the words in the text are known beforehand. Out-of-vocabulary words can cause problems in the classification process and may need to be handled separately.

5) Lack of information about word order: The bag-of-words model does not take into account the order of words in the text. This can be problematic for tasks that require understanding the structure of the text, such as sentiment analysis.

For now we will work with more sophisticated methods.

N-Gram Word Model

A new model, where each word is dependent on previous words unlike the Bag-of-Words model. We can start by making a word dependent on all previous words in a sentence. This model is in a sense perfectly “correct” in that it captures all possible interactions between words, but it is not practical: with a vocabulary of 100,000 words and a sentence length of 40, this model would have (10**200) parameters to estimate. We can compromise with a Markov chain model that considers only the dependence between n adjacent words. This is known as an **n-gram model** and we implement this model in this section. In an n-gram model, the probability of each word is dependent only on the n-1 previous words.

To implement an n-gram model, we can use the ngram_range parameter of the CountVectorizer class in scikit-learn. The ngram_range parameter takes a tuple with two values, n1 and n2, where n1 is the smallest n-gram to consider, and n2 is the largest. For example, if we set ngram_range=(1,2), we will create a bag of words model that includes both unigrams (single words) and bigrams (pairs of adjacent words). In this model, we also drop fewer of the (common or rare) occurring words, and total features we use are 4123.

Naive Bayes

MultinomialNB is used to train a Naive Bayes model on the N-gram model features obtained from the preprocessed review text. After the model is trained, it is used to predict the sentiment of the test data, and the accuracy of the predictions is evaluated.
We calculate the accuracy of the classifier by comparing the predicted sentiments with the actual sentiments.

Accuracy = 0.86105

That's a considerable improvement from the previous bag-of-words model. The n-gram approach is capturing more of the contextual information present in the reviews, which can help improve the accuracy of the classifier.

Cross-validation: N-Gram Words

The average score of the cross-validation is 0.85825, which is similar to the model's accuracy score of 0.86105. This suggests that the model's performance is consistent across different subsets of the data and that it is a reliable estimator of the model's accuracy.

Limitations: N-Gram word Model

Here are some limitations of n-gram models:

1) Fixed-length context: N-gram models consider only a fixed number of preceding words to predict the next word, which limits their ability to capture long-range dependencies in language.
2) Sparsity: N-gram models suffer from sparsity, which means that they may not have enough data to accurately estimate the probability of some rare n-grams.
3) Out-of-vocabulary words: N-gram models cannot handle words that are not present in the training corpus, which can lead to problems when dealing with rare or domain-specific words.
4) Lack of semantic understanding: N-gram models do not have any explicit representation of the meaning of words or the relationships between them, which limits their ability to capture semantic nuances in language.
5) Limited context: N-gram models only consider the preceding words, but often the context needed to understand a sentence or document extends beyond a fixed length window.

Learning can be improved by representing words as points in a high-dimensional space, rather than as atomic values. The next section covers the use of recurrent neural networks to capture meaning and long-distance context as text is processed sequentially.

Deep Learning Models

Create Word Embeddings (Batch Processing)

We use the Universal Sentence Encoder to create sentence embeddings, which are then used as the input to the neural network. This approach removes the need for feature selection and allows the model to learn the most important features directly from the text data.

This is a big model and will take considerable time. One way to make this quicker is by using batch processing. Instead of passing all the reviews through the Universal Sentence Encoder at once, we can divide them into smaller batches and process each batch separately. This can significantly reduce the processing time and memory usage. We create sentence embeddings in batches of 1000 reviews at a time, rather than processing all reviews at once. This reduces memory usage and allows us to process large datasets efficiently.

Recurrent Neural Networks (Gated Recurrent Unit)

Accuracy = 0.8536 = 85%

Limitations: Recurrent Neural Networks (Gated Recurrent Unit) Model

Some of the limitations of using RNNs with GRU layers for sentiment analysis on Amazon reviews include:

1. Difficulty in capturing long-term dependencies: RNNs with GRU layers tend to perform poorly in capturing long-term dependencies due to the vanishing gradient problem. This can lead to the model ignoring important information that occurred earlier in the text, which can be crucial for sentiment analysis.
2. Limited context awareness: RNNs with GRU layers rely on a fixed-length context window to make predictions, which can limit their ability to capture the full context of a review. This can result in the model making inaccurate predictions based on incomplete or irrelevant information.
3. High computational complexity: RNNs with GRU layers require significant computational resources, particularly for large datasets. This can result in longer training times, which can be impractical for real-time applications.
4. Sensitivity to hyperparameters: RNNs with GRU layers are sensitive to hyperparameters such as learning rate, dropout rate, and the number of layers. If these hyperparameters are not chosen carefully, the model may not perform well on the given task.
5. Limited ability to handle noisy data: RNNs with GRU layers can struggle to handle noisy or inconsistent data, such as misspelled words or grammatical errors. This can lead to inaccurate predictions and a decrease in overall model performance.

Recurrent Neural Networks (LSTM Layers)

Recurrent Neural Networks (RNNs) are a type of neural network that can process sequential data, such as text. In traditional feedforward neural networks, the input and output layers are connected by a series of hidden layers, and information flows from input to output in one direction. However, RNNs can take the previous output as input for the current timestep, allowing them to capture temporal dependencies in sequential data.

We will also implement a RNN with LSTM layers, where the model architecture consists of two LSTM layers, each with a dropout layer to help prevent overfitting, and a final dense layer with a sigmoid activation function to make binary predictions.

This is a kind of RNN with gating units that don't suffer from the problem of imperfectly reproducing a message from one time step to the next. Rather, an LSTM can choose to remember some parts of the input, copying it over to the next timestep, and to forget other parts. An LSTM can learn to create a latent feature for the subject person and number and copy that feature forward without alteration until it is needed to make a choice like this. A regular RNN (or an n-gram model for that matter) often gets confused in long sentences with many intervening words between the subject and verb.

Accuracy = 0.7857 = 78%

Limitations: Recurrent Neural Networks (LSTM Layers) Model

Here are some limitations of RNN with LSTM layer model:

1. Computationally expensive: LSTM models require significant computational resources and can be slow to train, especially for large datasets.
2. Overfitting: RNNs, including LSTMs, can easily overfit the training data, leading to poor generalization performance on unseen data.
3. Gradient vanishing and exploding: RNNs, including LSTMs, can suffer from the vanishing or exploding gradient problem during training, which can lead to unstable or slow convergence.
4. Limited memory capacity: LSTMs have a fixed memory capacity, which can limit their ability to learn long-term dependencies in the input data.
5. Difficulty with variable-length inputs: RNNs, including LSTMs, require fixed-length inputs, which can be challenging to handle for datasets with variable-length inputs, such as text.
6. Difficulty with out-of-vocabulary words: LSTMs may struggle to handle words that are not present in the training data or that are rare, leading to poor performance on such words.
7. Difficulty with noisy or ambiguous input: RNNs, including LSTMs, may struggle with noisy or ambiguous input data, such as misspelled words or sarcasm, leading to inaccurate predictions.

Model Evaluation

Model evaluation is an important aspect of any machine learning project as it helps to determine the performance of a model on a given task. In this case, we evaluated the performance of four models - N-Gram model, Bag-of-Words model, Recurrent Neural Networks (Gated Recurrent Unit) model, and RNN with LSTM Layer - on a sentiment analysis task.

Accuracy is a commonly used metric to evaluate the performance of classification models. It measures the proportion of correct predictions made by the model.

Accuracy - It is defined as (true positives + true negatives) / (true positives + true negatives + false positives + false negatives).

Accuracy for each respective model is plotted (**Figure 4**). We will try and choose the model with the highest accuracy. As we can see (**Table 1**), the N-Gram model has the highest accuracy with around 86% and therefore, is considered the best model among these for this classification task. The Recurrent Neural Networks (Gated Recurrent Unit) model also performed almost equally well with an accuracy score of about 85%. The Bag-of-Words model didn't perform well at all, probably because of the uneven spread of star_ratings which we observed during our EDA (**Figure 3**).

It is important to note that accuracy is not always the best metric to use, depending on the problem at hand. For example, in a highly imbalanced dataset, accuracy may not be the best metric to use as the model could perform well on the majority class but poorly on the minority class. In such cases, other metrics such as precision, recall, and F1 score may be more appropriate.

Future Ideas

There are several ways to extend this project in the future, depending on the goals and resources available.

- Incorporate more features: In addition to the text data, there are other features that can be useful for sentiment analysis, such as metadata (e.g., product category, brand, price), user information (e.g., age, gender, location), and time-related information (e.g., date of review). By incorporating these features into the model, we can improve its performance and gain more insights into the factors that affect sentiment.

- Deal with imbalanced data: In this project, we observed that the distribution of star_ratings was not uniform, which could affect the performance of the models. To deal with this issue, we can use techniques such as oversampling (e.g., SMOTE) or undersampling (e.g., random undersampling) to balance the data. We can also use different evaluation metrics that are more suitable for imbalanced data, such as F1-score or AUC-ROC.

- Extend to other languages: In this project, we focused on sentiment analysis for English reviews. However, sentiment analysis can be applied to reviews in other languages as well. By using machine translation or training models on multilingual data, we can extend the project to support sentiment analysis for reviews in different languages.

- Real-time sentiment analysis: In this project, we focused on analyzing a static dataset of reviews. However, in practice, reviews are generated continuously and sentiment analysis needs to be done in real-time. By developing a real-time sentiment analysis system, we can provide instant feedback to businesses or individuals and enable them to take quick actions based on the feedback received.

The code is available on the link -
https://github.com/YashChat/Sentiment_Analysis-Amazon_Reviews

References

Klambauer, Guenter, et al. "Recurrent neural networks for sentiment analysis of short texts." Proceedings of the 4th Workshop on Representation Learning for NLP (ACL). 2019.

Koenig, R. (2019, July 28). NLP for Beginners: Cleaning & Preprocessing Text Data. towardsdatascience.com. Retrieved March 20, 2023, from https://towardsdatascience.com/nlp-for-beginners-cleaning-preprocessing-text-data-ae8 e306bef0f Loukas, S. (2020, October 12).

Maas, Andrew L., et al. "Bag of words meets bags of popcorn: Sentiment analysis with movie reviews." Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics. 2012.

Text Classification Using Naive Bayes: Theory & A Working Example. towardsdatascience.com. Retrieved March 20, 2023, from https://towardsdatascience.com/text-classification-using-naive-bayes-theory-a-workingexample-2ef4b7 eb7d5a

Text Classification with RNN. towardsai.net. (2020, November 21). Retrieved March 20, 2023, from https://towardsai.net/p/deep-learning/text-classification-with-rnn

Zaremba, W., Sutskever, I., & Vinyals, O. (2014). Recurrent neural network regularization. arXiv preprint arXiv:1409.2329.

Table 1: Classification Models and their Accuracies

Classification Model	Accuracy Score
Bag-of-Words (Naive Bayes)	4%
N-Gram Model	86%
Recurrent Neural Networks (Gated Recurrent Unit)	85%
Recurrent Neural Networks (LSTM Layers)	78%

Figure 1: Plot for the star ratings for verified purchase vs non verified purchases

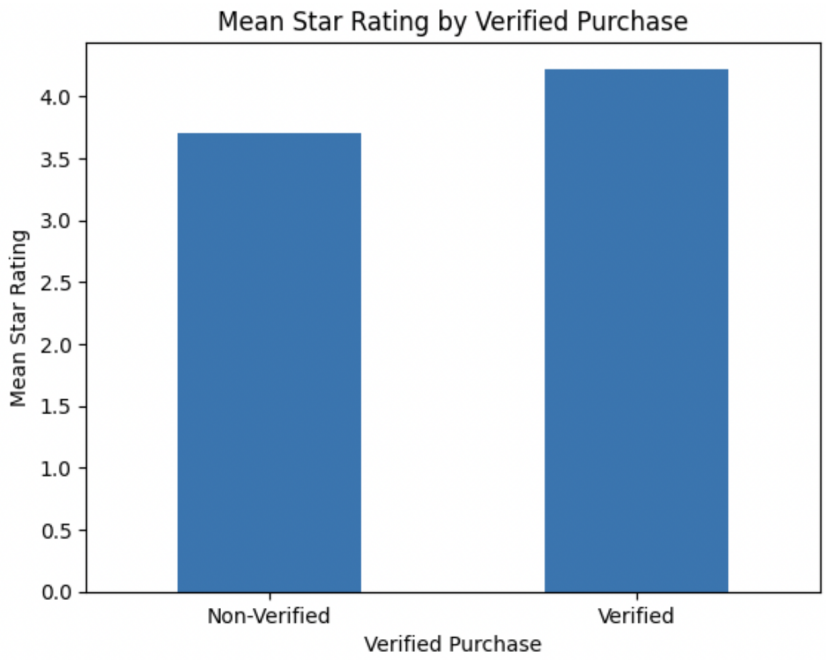

Figure 2: Box plot for the star ratings for verified purchase vs unverified purchases

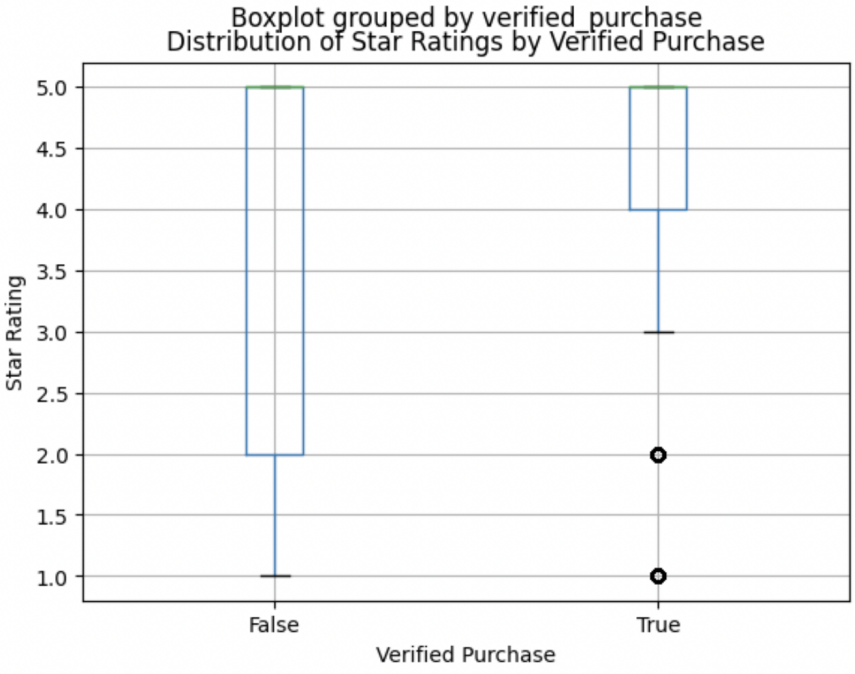

Figure 3: Frequency Distribution of Star Ratings

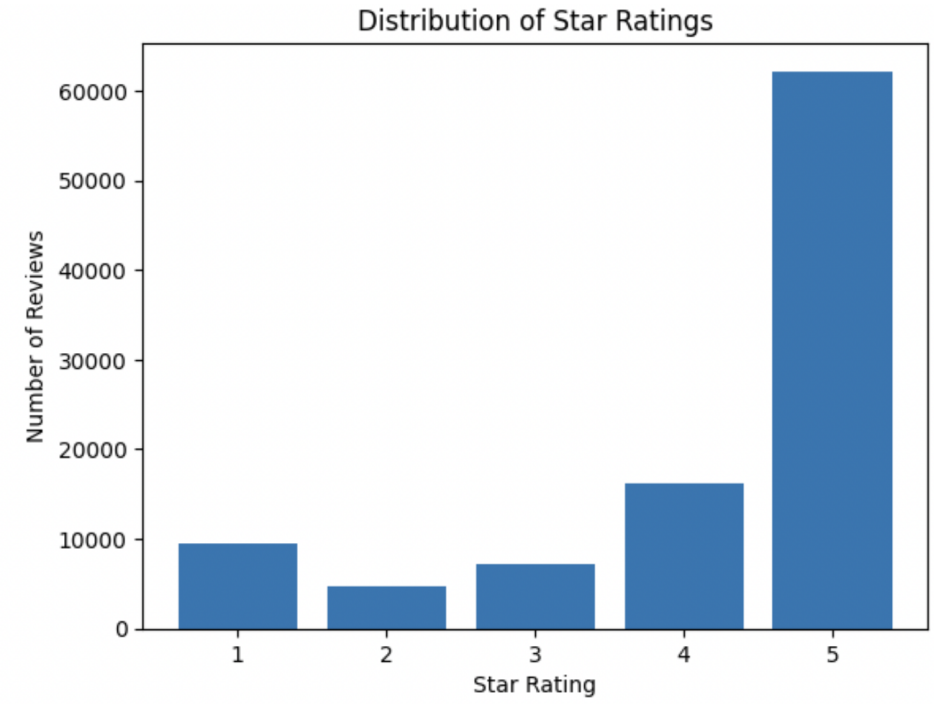

Figure 4: Accuracies of the different models performance