2023

Evaluation of Text Transformers for Classifying Sentiment of Reviews by Using TF-IDF, BERT (word embedding), SBERT (sentence embedding) with Support Vector Machine Evaluation

Mina Jamshidian
Technological University Dublin, Ireland

Evaluation of Text Transformers for Classifying Sentiment of Reviews by Using TF-IDF, BERT (word embedding), SBERT (sentence embedding) with Support Vector MachineEvaluation

Mina Jamshidian

A dissertation submitted in partial fulfilment of the requirements of
Technological University Dublin for the degree of
M.Sc. in Computing (Data Science)

January 2023

Declaration

I certify that this dissertation which I now submit for examination for the award of M.Sc. in Computing (Data Science), is entirely my own work and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.

This dissertation has been prepared in accordance with the regulations for postgraduate study at of the Technological University Dublin and has not been previously submitted in whole or part for an award at any other Institute or University.

The work reported in this dissertation adheres to the principles and requirements of the Institute's guidelines for ethical research.

Signed: Mina Jamshidian

Date: 05/01/2023

Abstract

As the online world evolves and new media emerge, consumers are sharing their reviews and opinions online. This has been studied in various academic fields, including marketing and computer science. Sentiment analysis, a technique used to identify the sentiment of a piece of text, has been researched in different domains such as movie reviews and mobile app ratings. However, the video game industry has received relatively little research on experiential products. The purpose of this study is to apply sentiment analysis to user reviews of games on Steam, a popular gaming platform, in order to produce actionable results. The video game industry is a major contributor to the entertainment industry's revenue and customer feedback is crucial for game developers. Sentiment analysis is widely used by companies to discover what customers are saying about their products. This paper proposes a process for evaluating video game acceptance using game user reviews through the application of sentiment analysis techniques.

The focus of this study is to examine the performance of different Text Transformer techniques in the context of text mining when applied to Steam game reviews, using an Support Vector Machine classifier. The goal is to compare the effectiveness of these methods for predicting sentiment, and to develop software that can accurately predict sentiment and explain the prediction through text highlighting. Specifically, the study aims to compare a sentiment analysis classifier based on the traditional TF-IDF text feature representation method to classifiers using the more recent BERT and SBERT techniques. The ultimate goal is to develop a more clear and accurate sentiment prediction tool.

Keywords: Sentiment Analysis, Support Vector Machine, Bert (Word Embedding), SBERT (Sentence Embedding), TF-IDF, NLP

Acknowledgements

I am deeply grateful to my supervisor, Dr. Bojan Božić, for his guidance and support throughout this process. Thank you for believing in my capabilities and providing direction during challenging times. Your encouragement and trust have been invaluable to me.

I am thankful to Dr. Luca Longo for their assistance in helping me clarify my area of interest.

I would like to extend my gratitude to my husband, Afshin Mehrabani, for his love and support throughout my master's course. His encouragement and understanding enabled me to successfully complete this program.

Table of contents

List of figures

List of tables

List of Acronyms

API	Application Programming Interface
BERT	Bidirectional Encoder Representations from Transformers
BOW	Bag of Word
CNN	Convolutional Neural Networks
CRISP – DM	Cross Industry Process for Data Mining
DT	Decision Trees
FN	False Negatives
FP	False Positives
GPU	Graphics Processing Unit
KNN	K-Nearest Neighbors
ML	Machine Learning
NB	Naive Bayes
NLP	Natural Language Processing
POS	Part Of Speech
RF	Random Forest
ROC	Receiver Operating Characteristic
SBERT	Sentence Bidirectional Encoder Representations from Transformers

SVM	Support Vector Machine
TF – IDF	Term Frequency Inverse Document Frequency
TN	True Negatives
TP	True Positives

Chapter 1

Introduction

1.1 Background

The video game industry has exploded in popularity and profitability in recent years, with billions of people around the world enjoying a wide variety of games on various platforms. In this competitive market, it's important for game developers and companies to create high-quality products that stand out from the competition. Strong storytelling, a stable multiplayer server, and fluid combat are all key elements that can contribute to a game's success and ensure it is well-received by players. Conducting a sentiment analysis can be useful in understanding how players feel about a game and how their emotions may be related to different aspects of the game (Fang & Zhan, 2015).

By analyzing the sentiment of customer reviews, companies can gain valuable insights into the opinions and experiences of their players, which can in turn lead to increased profits (Utz et al., 2012). Sentiment analysis can also help to uncover hidden sentiments within reviews that may not be immediately apparent (Lu & Wu, 2019). Overall, sentiment analysis can be a useful tool for understanding the acceptance of a video game among its players and can help companies make informed decisions about how to improve and market their games (Fang & Zhan, 2015; Vieira & Brandão, 2019).

Sentiment analysis is a subfield of natural language processing, which is the study of how computers can understand, interpret, and generate human language. Within the field of sentiment analysis, techniques have become increasingly advanced over the past decade, allowing for more accurate and nuanced analysis of text and language. In order to classify text using machine learning algorithms, it is often necessary to first transform the raw text using techniques such as text transformation(text Vectorization) , stemming, and lemmatization. These techniques help to pre-process the text and make it more suitable for analysis by breaking it down into smaller units and standardizing the form of words. Once the text has been transformed, it can be fed into machine learning algorithms for classification, allowing for the automated analysis of sentiment and other linguistic phenomena (Chouikhi et al., 2020).

In recent years, the BERT (Bidirectional Encoder Representations from Transformers) pre-trained transformer has become a widely used tool for natural language processing tasks, particularly within the industry. BERT is known for its flexibility and versatility, as it can be fine-tuned to work with any corpus or language, making it a popular choice for many applications. However, other approaches to natural language processing have also proven to be effective in the past. In this study, the performance of BERT and SBERT (Sentence-BERT) will be compared to that of traditional machine learning algorithms using a vocabulary generated using the TF-IDF (Term Frequency-Inverse Document Frequency) method. This comparison will help to evaluate the effectiveness of these different approaches and provide insight into which may be most suitable for specific tasks or applications.

1.2 Research Problem

One of the main challenges in natural language processing is converting text into a format that can be input to a machine learning algorithm. One common method for achieving this is using a text transformer algorithm such as TF-IDF (Term Frequency-Inverse Document Frequency) (Alzami et al., 2020; Balakrishnan et al., 2020; Cahyanti et al., 2020; Zuo, 2018). TF-IDF is a measure that reflects the frequency with which a word appears in a set of documents and the significance of the word within that set are both factors that contribute to the word's overall importance in the corpus. The bag-of-words model, on which TF-IDF is based, ignores the context, semantics, position, and cross-document occurrence of words. It only takes into account the frequency of words within a single document. Despite its widespread use, TF-IDF has certain limitations and may not be suitable for all natural language processing tasks. In this study, we aim to explore the effectiveness of different approaches to converting text into vectors, including traditional machine learning algorithms and pre-trained transformer models such as BERT and SBERT, and to provide insight into which may be most suitable for specific tasks or applications.

The use of BERT word embeddings in natural language processing has been shown to improve model performance due to their ability to capture subtle differences in word meaning and context. These embeddings are created using a dynamic process that takes into account the words surrounding a given word, allowing for more precise representation of features. BERT sentence embeddings, or SBERT, are an extension of BERT word embeddings that can be used to compare sentences using methods such as cosine similarity. SBERT shares many similarities with BERT word embeddings, but allows for the comparison of entire sentences rather than individual words. Overall, the use of BERT and SBERT embeddings has the potential to improve the accuracy and effectiveness of natural language processing models (Reimers & Gurevych, 2019).

Research Question:

The research question for this study is as follows:

"Is it possible for a Support Vector Machine classifier model that utilizes 'BERT' or 'SBERT' as pre-trained transformer techniques to achieve statistically significant higher accuracy compared to a Support Vector Machine model employing TF-IDF as the transformer technique for text classification of review sentiments?"

1.3 Research Objectives

The purpose of this study is to evaluate a research question through the use of a hypothesis. The hypothesis will be tested through the implementation of experiments, during which various metrics will be calculated and compared. The outcome of the hypothesis will be determined by the results of these experiments. To determine the validity of the hypothesis, statistical difference tests will be conducted between the two models being compared. If the difference between the models is statistically significant (p<0.05), the null hypothesis will be rejected and the alternative hypothesis will be accepted. On the other hand, if the difference is not statistically significant, the null hypothesis will be accepted and the alternative hypothesis will be rejected.

Evaluating a hypothesis through the scientific method involves designing and conducting experiments to test the hypothesis, and then analyzing the results using statistical techniques. This process is essential in order to determine the validity of the research question and reach a conclusion about the hypothesis. The process of experimentation and statistical analysis allows researchers to gather evidence and make informed decisions about the potential accuracy of the hypothesis. It is a key component of the scientific method, as it helps to ensure that the results of the research are reliable and can be replicated by other researchers. By carefully evaluating the hypothesis through experimentation and statistical analysis, researchers can better understand the underlying phenomena being studied and contribute to the body of knowledge in their field.

Research Hypothesis:

Null Hypothesis (H0) : If a SVM classifier is built on 'BERT' and 'SBERT' as pre-trained text transformer techniques then it will not achieve statistically significant higher accuracy for predicting sentiment of reviews, than a SVM classifier model built on TF-IDF text transformer technique.

Alternative Hypothesis (H1) : If a SVM classifier is built on 'BERT' and 'SBERT' as pre-trained text transformer techniques, it will achieve statistically significant higher accuracy for predicting sentiment of reviews, than a SVM classifier model built on TF-IDF text transformer technique.

1.4 Research Methodologies

The research conducted in this study is quantitative in nature, which means that it involves collecting and analyzing numerical data in order to answer a research question or test a hypothesis. To gather the data for this experiment, the researchers obtained a dataset from Kaggle [1], a website that provides access to a variety of pre-existing datasets that can be used for research purposes. This type of research is known as secondary research, as the data has already been collected and is being used for a new study. The specific dataset used in this experiment was the Steam Reviews 2021 dataset, which contains a collection of user reviews for video games on the Steam platform.

The research process in this study follows a deductive approach, where a research question or hypothesis is formulated and then tested through experiments. In this approach, researchers start with a general idea or theory and then use specific data and analysis to either support or refute that idea. In this study, statistical analysis was used to analyze the results of the experiments and determine whether the initial hypothesis could be accepted or rejected based on the data.

Overall, the study employed a systematic and data-driven approach to address a specific research question or test a hypothesis. This approach involves using both secondary data, which has already been collected for another purpose, and statistical analysis to draw conclusions and reach a final conclusion. This type of research is characterized by its rigorous and structured methodology, which helps to ensure that the results are reliable and accurate.

1.5 Research Scope and Limitations

BERT and SBERT are two types of embeddings that have been developed to transform text for various natural language processing tasks. In this research, the aim is to use BERT as a word embedding and SBERT as a sentence embedding for the text transformation of user reviews on Steam. The transformed reviews will then be used

[1] https://www.kaggle.com/najzeko/steam-reviews-2021

to classify the sentiment of the reviews using a Support Vector Machine. To the best of our knowledge, BERT and SBERT have not been previously used as the text transformer technique for classifying the sentiment of user reviews on Steam. The goal of this research is to explore the potential of BERT and SBERT as embeddings for text transformation and to evaluate their performance in conjunction with a Support Vector Machine for sentiment classification on user reviews from Steam.

This study has some limitations to consider. One limitation is that the BERT and SBERT models, which are typically used to train deep learning algorithms, were applied to train a machine learning algorithm (an SVM classifier) in this research. This means that the results of this study may not be directly comparable to other studies that used deep learning approaches to train their models. Another limitation is that the SVM classifier with the TF-IDF text representation technique was chosen as the baseline in this study. While this approach was used in a previous study (Alzami et al., 2020), it is possible that the accuracy achieved in this study could be different due to the use of a different dataset.

It is the purpose of this study to compare pre-trained embeddings to embeddings generated from scratch in terms of matching or outperforming them. This study examines only the BERT and SBERT pre-trained embeddings in great detail and uses TF-IDF as a reference. There are, however, a number of pre-trained embedding techniques available, such as Language Models' Text Transformer (Peters et al., 2018). It cannot be guaranteed that the research will produce the same results as BERT and SBERT did in other domains.

1.6 Document Outline

Chapter 2 - Literature review and related work:

The main objective of this chapter is to give a comprehensive overview of previous research on sentiment analysis of user reviews in different domains, as well as text

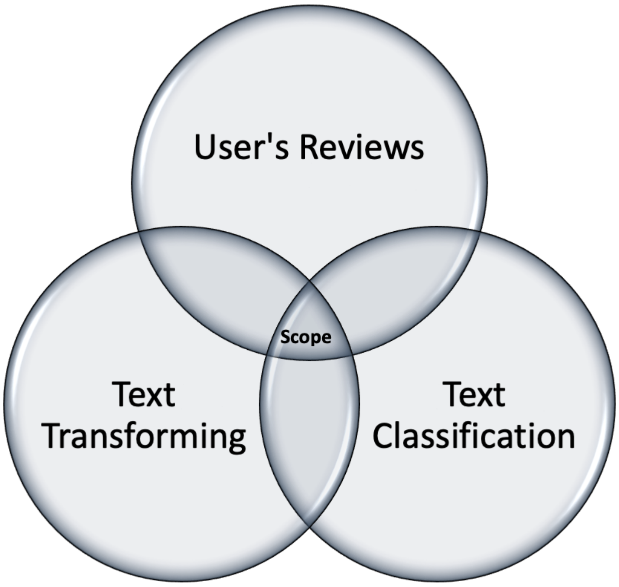


Fig. 1.1 Research Scope

transformation techniques like Term Frequency-Inverse Document Frequency (TF-IDF) and natural language processing methods such as word and sentence embeddings like BERT and SBERT. The chapter also aims to analyze different text classification approaches, including Support Vector Machines (SVM), in relation to the analysis of sentiment in user reviews. The goal of this review is to identify any areas that have not been thoroughly researched in the field and to create a research question that addresses these gaps. By examining previous research studies, a better understanding of the state of the art in sentiment analysis of user reviews can be gained and areas requiring further investigation can be determined.

Chapter 3 - Design and Methodology:

The aim of this chapter is to explain how the CRISP-DM methodology was utilized in the development of the dissertation and the execution of experiments related to the research question. The Steam dataset is initially introduced, and data understanding is given significant emphasis in this section. This involves understanding the motivations behind the selection of specific data preparation and data cleansing techniques for use on the dataset. The data preparation process is then thoroughly explained, including all steps taken. Following this, the text transformation techniques utilized in the experiment, such as TF-IDF, BERT, and SBERT, are described in detail. The modeling phase of the experiment is then discussed, which involves the use of SVM with various text transformation techniques such as the base model and a model that will be compared to the base model. Finally, the evaluation phase of the experiment is described, including a thorough explanation of the methods and steps used in this process.

Chapter 4 - Results, evaluation and discussion:

This chapter serves as a thorough examination of the data preparation process, as well as the results and analysis of various experiments conducted. Specifically, this chapter delves into the implementation and results of using TF-IDF, BERT, and SBERT with SVM, and includes a discussion of the cross-validation results and statistical tests performed to validate or refute the hypothesis. To begin, the data preparation process is thoroughly explained, including any necessary preprocessing or cleaning steps. Next, the results of each experiment are presented, along with a detailed discussion of the techniques used and their effectiveness. This includes a thorough analysis of the performance of TF-IDF, BERT, and SBERT with SVM, including any limitations or strengths of each method.

Following the presentation of the results, cross-validation results and statistical tests are described, highlighting the methods used to validate or refute the hypothesis being tested. Finally, an in-depth analysis of the results is presented, including an interpretation of the findings and their implications. Overall, the "Results, Evaluation and Discussion" chapter provides a comprehensive look at the data preparation process, the results of various experiments, and the analysis and interpretation of those results.

Chapter 5 - Conclusions:

This chapter presents a summary of the entire study, highlighting its main objectives and the results achieved. It reflects on the research conducted and presents the final conclusions drawn from the findings. To begin, the chapter summarizes the main objectives of the study and the methods used to achieve them. This includes a brief overview of the data preparation process and the experiments conducted, as well as a summary of the results obtained. Next, the chapter presents the final conclusions of the study, highlighting the main findings and their implications. This includes an in-depth analysis of the results, as well as an interpretation of their significance.

In conclusion, this chapter identifies potential directions for future research that could expand upon the findings of this study. This includes suggestions for further studies that could deepen our understanding of the topic, as well as ideas for how the results of this study could be applied in practical settings. Overall, this chapter provides a comprehensive summary of the study and its findings, along with ideas for future research that could further advance our understanding of the topic.

Chapter 2

Review of relevant literature and previous research

The purpose of this chapter is to review and summarize existing research on sentiment analysis and the analysis of consumer expressions, such as reviews. This research typically involves the use of encoding techniques, which are methods for representing text data in a format that can be processed by machine learning algorithms, as well as classification techniques, which are methods for assigning a label or category to a piece of text based on its content. In this review, some of the most commonly used technical techniques for sentiment analysis will be compared and their advantages and disadvantages will be discussed. Based on the findings of this overview, research gaps and questions will be identified and a research question will be formulated. The goal of this review is to provide a comprehensive summary of the current state of the field and to identify areas where further research is needed.

2.1 Associated Researches

2.1.1 Sentiment Analysis

In recent years, the use of digital platforms for collecting text-based consumer reviews has become increasingly common. These reviews are a valuable source of information for companies, as they provide insight into what customers think about a product or service. User reviews are typically the most common form of user feedback and can be an important source of information for companies looking to improve their products and services. Given the importance of customer reviews, it is crucial for companies to have an effective way to analyze the sentiment of the reviews they receive. One way to do this is to implement automated frameworks that use machine learning or natural language processing techniques to analyze the sentiment of a large number of reviews in an efficient and unbiased manner. By using these frameworks, companies can quickly and accurately assess the overall sentiment of their reviews and use this information to identify areas for improvement and make informed decisions. (Gallagher et al., 2019).

On the other hand in the modern digital age, many products, including games, can only be purchased online and are not available through traditional brick-and-mortar stores. This means that for many consumers, the only way to gather information about the user experience of a product before making a purchase is to rely on online reviews or ratings (Sobkowicz & Stokowiec, 2016). As a result, the sentiment of these online reviews can be an important factor for consumers when deciding whether or not to purchase a product. This is why sentiment analysis techniques are often used to evaluate the overall sentiment of a large number of online reviews, in order to provide a summary of the user experience of a product and help potential buyers make informed decisions.

By doing this, businesses can provide guidance to their clients, recommend appropriate products, and resolve negative feedback by implementing these frameworks. It is also possible to apply sentiment analysis to competitors in order to prevent repeating the mistakes they have made in the past. Therefore, for game products, sentiment analysis can be extremely useful and helpful. Text reviews have been analyzed using a variety of approaches by marketing researchers over the years. It has been hypothesized by the authors of Alantari et al. (2022) that diagnostic and predictive skills may be a trade off that is faced empirically.

According to research conducted by Iqbal et al. (2022), the use of machine learning techniques with neural networks and text preparation techniques resulted in the most precise predictions for sentiment analysis tasks. There are a wide range of analytical methods that can be used to evaluate sentiments, including machine learning and deep learning techniques. These methods involve the use of algorithms and statistical models to analyze and interpret the emotional tone of text data. By applying these techniques to large datasets, it is possible to identify patterns and trends in sentiments and use this information to make predictions or take informed decisions.

It is widely recognized that machine learning models cannot be applied directly to raw text reviews for sentiment analysis tasks without first preprocessing and transforming the data. This is why numerous studies have been conducted to demonstrate the importance of preprocessing and transforming text data before applying machine learning algorithms. There are a variety of methods that can be used to prepare text reviews for machine learning classification, including techniques such as stemming, lemmatization, and removal of punctuation and special characters. The choice of text transformer and machine learning classifier can also have an impact on the accuracy of the results. Different transformer and classifier combinations may be more or less effective for specific datasets or tasks, so it is important to carefully consider which methods to use in order to achieve the best possible results.

2.1.2 Classifier

In a study by Zuo (2018), the effectiveness of sentiment analysis was evaluated in terms of accuracy, precision, and recall by analyzing the sentiment of a large scale Steam Review dataset. The study compared the performance of two supervised machine learning algorithms, namely Decision Tree and Gaussian Naive Bayes, and found that the Decision Tree model achieved an accuracy of approximately 75% compared to the Gaussian Naive Bayes model. This result was specific to the Steam Review dataset used in the study. Overall, the findings of this study suggest that the Decision Tree algorithm may be a more effective method for sentiment analysis tasks when applied to the Steam Review dataset.

The study was done by Balakrishnan et al. (2020) investigated four supervised learning algorithms using Python for a Sentiment and Emotion Analysis. In particular, Support Vector Machine, Naive Bayes, Decision Trees and Random Forest were compared for Sentiment and Emotion Analysis. Accuracy and F1 scores indicate the Random Forest classifier with 75.62% accuracy and almost 1000 reviews achieved the highest classification accuracy. A larger dataset could have provided better results according to the study's authors, who believe the study would have been more accurate if a larger dataset had been used (Balakrishnan et al., 2020) .

According to the study obtained by Normah (2019), the purpose of the study was to examine customer sentiment toward Windows Phone Store applications. This was done based on automated categorization of reviews in order to identify positive and negative sentiments. Because of its simplicity and level of performance, Nave Bayes has been proven to be a reliable classification model for a wide range of textual domains. This is also true for a wide range of different types of textual data. For the validation of the model, we used tenfold cross validation. For the measurements, a Confusion Matrix and ROC curve were used. This study showed an accuracy rate of 84.50%, which indicates that Naive Bayes is a suitable model for text classification especially in the case of sentiment analysis (Normah, 2019).

According to this study, two classifier types have been applied namely Naive Bayes and Support Vector Machines along with different feature selection methods in order to perform sentiment analysis on movie reviews. Different methods of feature selection and how they affect sentiment analysis were discussed. As a result, it is evident from the classification results that the Linear SVM classifier provides a higher level of accuracy compared to the Naive Bayes classifier. SVM has also been identified as a more effective method for sentiment analysis in many previous studies but the results obtained from linear SVM are also superior. Based on the model described in this paper, hybrid techniques can be beneficial for sentiment analysis. Incorporating the corpus and selecting features in an effective manner can lead to significant improvements (Tripathi & S, 2015).

This study was conducted by Jeffrey et al. (2020) using Steam Review Datasets consisting of one million reviews of four video games. The Support Vector Machine algorithm and the Nive Bayes algorithm are both able to achieve approximately 85%

accuracy, but they differ greatly in many ways. In the SVMS, certain interactions can occur between features. However, in the NB, features are considered to be independent features that do not interact with each other and are therefore not taken into consideration in the calculation. This study, according to the findings of the authors, indicates that pretrained text transformers, such as BERT, can be used to increase performance by leveraging the learning process (Jeffrey et al., 2020).

2.1.3 TF-IDF Text Transformer Technique

Srivastava et al. (2021) conducted this study in order to focus on feature generation by using a bag-of-words based method as well as the TF-IDF to generate the sentiment analysis features as well as the use of machine learning to build up the sentiment analysis of Customer reviews. An experiment was conducted using a dataset of 20k reviews which were cleaned and pre-processed, and then TF-IDF and Bow were applied to extract features. The training and evaluation of the classifiers was carried out after the implementation of the classifiers. Classifiers are evaluated based on accuracy metrics. Among the three classifiers used to determine accuracy, MultinomialNB achieved the highest accuracy for Bag of Word features, while Random Forest performed better for TF-IDF. In Bag of Word, MultinomialNB had an accuracy rate of 82% and in TF-IDF Random Forest, it had an accuracy rate of 78% (Srivastava et al., 2021).

This study was carried out by Arief and Deris (2021) to observe the impact of text preprocessing on the processing of a set of unstructured product reviews, using sentiment classifiers like Decision Tree, Naïve Bayes, and Support Vector Machine. In terms of performance, SVM performed superiorly, with an accuracy of 88,13%, however the Nave Bayes classifier is faster, as it takes less time to execute. Furthermore, the experimental results using TF-IDF for feature extraction may result in improved classification accuracy. In light of the results obtained by this approach, it can be concluded that a good text preprocessing sequence is critical to the classifier's ability to predict the outcome of data that is unstructured (Arief & Deris, 2021).

In a research study conducted by Alzami et al. (2020), the aim was to identify a combination of feature extraction and machine learning methods that could improve the accuracy of polarity sentiment analysis. To achieve this, the authors used a variety of feature extraction techniques such as Word Bags, TF-IDF (term frequency-inverse document frequency), and Word2Vector, and applied machine learning algorithms including Random Forest, Support Vector Machines (SVM), K-Nearest Neighbors (KNN), and Nave Bayes. The results of this study may provide insights into which combinations of feature extraction and machine learning methods are most effective for polarity sentiment analysis tasks (Saifullah et al., 2021).

According to this study, the use of Support Vector Machines (SVM) with TF-IDF (term frequency-inverse document frequency) feature extraction resulted in an 87.3% performance for classifying the polarity of customer reviews in unstructured sentiment analysis tasks. This method involved the preprocessing of documents by removing punctuation and special characters and applying stemming techniques to standardize the words. The study also found that it is possible to achieve even better results in sentiment analysis by using transformer methods such as BERT (a deep learning transformer). Transformer models are a type of neural network architecture that are particularly well-suited for natural language processing tasks and can be used to effectively analyze the sentiment of text (Alzami et al., 2020).

2.1.4 Hybrid Model

In the study by Cahyanti et al. (2020), the authors explored the use of support vector machine (SVM) classification for analyzing movie review data. They found that using term frequency-inverse document frequency (TF-IDF) as a method of weighting words was effective in improving the accuracy of the SVM model. Additionally, they discovered that combining the extraction of latent features with TF-IDF using latent features Dirichlet allocation (LDA) could further improve performance by modeling topics in the review data. The combination of TF-IDF and LDA resulted in the highest performance, with an accuracy of 82.16%. This suggests that by combining these two techniques, it is possible to overcome the limitations of SVM when applied to movie review data.

According to a study by Dang et al. (2020), the combination of deep learning architectures with word embeddings (such as Word2Vec) can be more effective than traditional term frequency-inverse document frequency (TF-IDF) models for sentiment analysis tasks. In a separate study by Mohamed Ali et al. (2019), Word2Vec was used to create numerical representations of words and was tested alongside various deep learning and hybrid models. The hybrid model was found to be the most effective, achieving an accuracy of 89.2%. Additionally, Beseiso and Alzahrani (2020) found that using BERT word embeddings as features in their model led to better performance compared to other feature combinations.

2.1.5 BERT (Word Embedding)

In a study conducted by Dong et al. (2020), the authors used BERT to analyze reviews of online commodities. The BERT model was first trained, and then the review texts were encoded using a representation layer. Next, CNN and BERT were used to extract local features from the review text vectors, with a semantic connection layer being applied to merge the information from these two complementary models. Finally, a sentiment classification layer was used to classify the reviews based on their sentiment. According to the experimental results, the F1 value of the BERTCNN model (i.e., the combination of BERT and CNN) was 14.4% higher than the F1 values of BERT and CNN separately. The combination of BERT and CNN may improve the accuracy of sentiment analysis in a specific context, according to the findings of a study.

A study found that using BERT in combination with another classifier can increase the accuracy of the classifier (Dong et al., 2020). However, the combination of BERT with a convolutional neural network (CNN) did not result in improved accuracy compared to using a support vector machine (SVM) with term frequency-inverse document frequency (TF-IDF) (Huang et al., 2023). This raises the question of whether using SVM with BERT or SBERT (Sentence-BERT) transformers could be more accurate than using SVM with TF-IDF. It would be interesting to see if the increased efficiency and representation learning capabilities of BERT and SBERT can improve the performance of SVM in classification tasks.

2.1.6 SBERT (Sentence Embedding)

SBERT (Sentence-BERT) is a variant of the BERT (Bidirectional Encoder Representations from Transformers) language model that was specifically designed to process sentence-level tasks more efficiently.A study found that SBERT is 9% faster than InferSent, a rival sentence-level model, and 55% faster than Universal Sentence Encoder, a widely used sentence-level model (Reimers & Gurevych, 2019). This increased efficiency allows for tasks to be modeled using SBERT that would be computationally infeasible with the original BERT model. As an example, the authors of the study mention the use of SBERT for hierarchical clustering, a process that involves analyzing a large number of sentence combinations in order to group similar sentences together. With the original BERT model, this process would take approximately 65 hours when working with 10,000 sentences and 50 million sentence combinations. However, using SBERT, the same process can be completed in just five seconds.

2.2 Research Gaps

For the extraction of text features, almost all studies with a good accuracy for classifying reviews' sentiment. A TF-IDF value equals the relative importance of a term in a document in relation to the number of times it appears in the document, inverted by the number of documents that incorporate the word in a larger corpus and using a pretrained transformer such as BERT is advised by all researchers in the game area (Steam Platform). Existing literature papers used the TF-IDF method with a machine learning classifier.

It was recognized during a thorough literature review that there was a significant research gap since the BERT (Word Embedding) and SBERT (Sentence Embedding) Text transformers had never been used for sentiment analysis in the review domain, particularly in the gaming domain (Steam Platform). This is the primary reason this was identified as the major research gap. In addition, pretrained text transformer techniques can be used to effectively analyze the contextual meaning of words and sentences within a corpus of text, thereby providing a strong foundation for performing in-depth text analysis

This study will be conducted by utilizing BERT and SBERT embeddings as pretrained text transformer techniques, along with a Support Vector Machine Classifier, in order to compare and contrast the results with an SVM model based on the TF-IDF text transformer technique developed by Alzami et al. (2020) in the Steam Game domain Jeffrey et al. (2020)in the same way that has been used in this study previously.

Chapter 3

Experiment Design and Methodology

In this chapter, the steps that will be taken in the project will be outlined and explained. The project aims to develop a framework that can perform end-to-end conversions of text into numerical form, which can then be used to train a text classifier for sentiment analysis of user reviews. The research methods included in this framework are those that are relevant to the project's goals and have been thoroughly reviewed in the literature. The details of the experimental process that will be used to collect and analyze data will be described. This includes the research design, which will be explained in terms of the type of study, the research question being addressed, and the hypotheses being tested. The sample selection process will also be described, including the criteria used to select participants and the size of the sample. The data collection methods that will be used will be described, including information about the instruments or tools used to gather data and the procedures followed to ensure the accuracy and reliability of the data. The data analysis techniques that will be used to analyze the data will be described, including any statistical techniques or machine learning algorithms that will be used, as well as any other methods used to interpret the results of the study.

Overall, this chapter will provide a clear and detailed explanation of the methods that will be used to collect and analyze data, ensuring transparency and replicability in the research process. The main purpose of this framework is to efficiently process text data, transforming it into a format suitable for use in machine learning and text classification for sentiment analysis of user reviews. Its ultimate goal is to use machine learning techniques to analyze user reviews for sentiment. This framework is designed to facilitate the efficient and effective performance of necessary computations, making it a valuable tool for sentiment analysis of user reviews.

3.1 Methodology

Conducting sentiment analysis can be a complex and challenging process, as it involves analyzing the emotions or opinions expressed in text data. One way to approach this task is by using the CRISP-DM (Cross-Industry Standard Process for Data Mining), a recent study by Nabiha et al. (2021) showed this to be the case. CRISP-DM is a widely used methodology for data mining that provides a structured approach to understanding and addressing a research question.

By following the steps of CRISP-DM as it is obvious in Figure 3.1 and using the Python programming language, it is possible to simplify and clarify the sentiment analysis process, making it more understandable and easier to implement. This can be especially helpful when deploying the project in a real-world setting. Figure 3.2 illustrates the methodology and steps that will be followed in this project using the CRISP-DM method.

3.2 Data Understanding

Data understanding is the process of gaining a deep understanding of a dataset, including its characteristics, properties, and any potential issues or challenges. This involves gathering detailed information about the dataset, such as how and where it was collected, the sample size, the variables included, and any biases or limitations. By thoroughly understanding the dataset, it is possible to determine its suitability for a research or project and to identify and address any potential issues that may arise during the data analysis process. The ultimate goal of data understanding is to ensure the reliability and validity of findings.

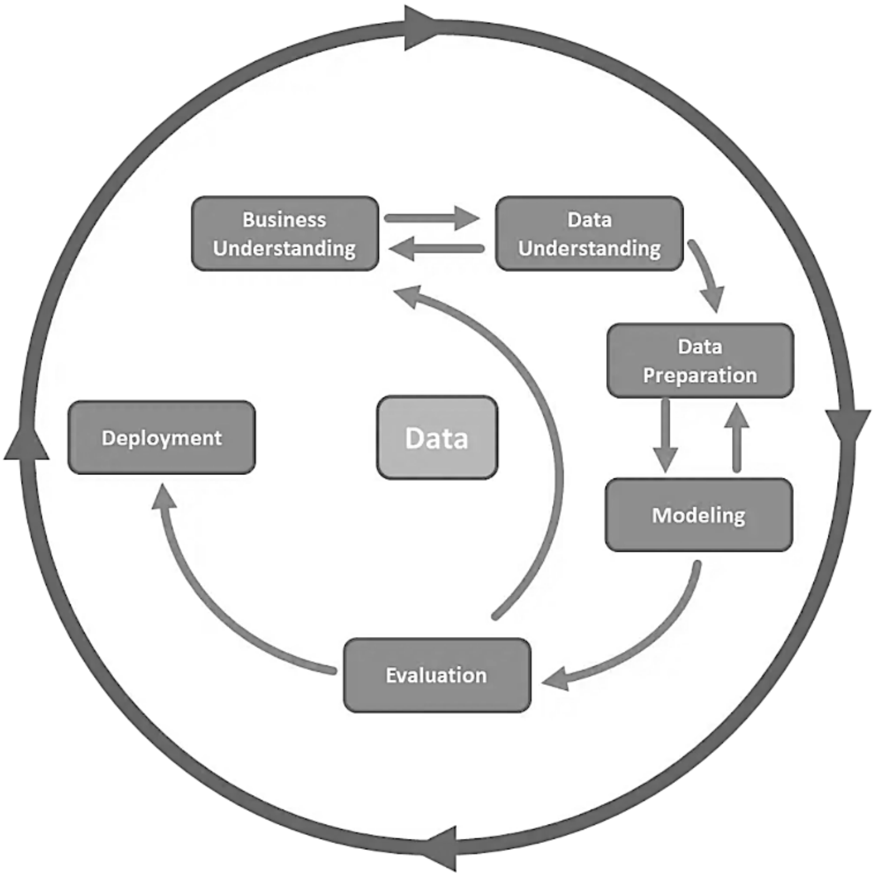


Fig. 3.1 CRISP-DM Diagram

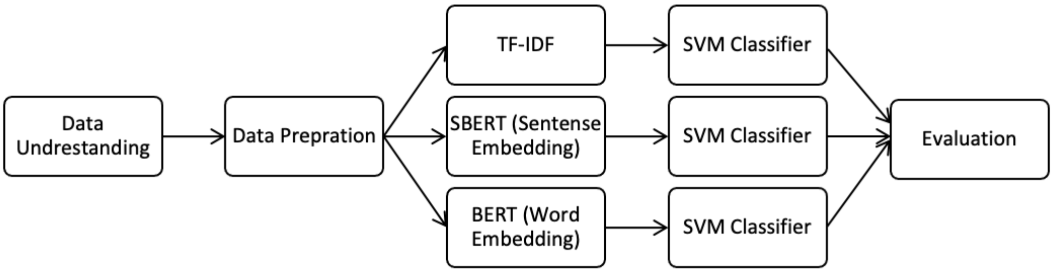


Fig. 3.2 Diagram of the experimental implementation with the associated phases of CRISP-DM

3.2.1 The Dataset

This dataset is based on Kaggle[1] data which contains over 21 million user reviews of approximately 300 different games on Steam. Based on the documentation provided by Steam, the reviews were obtained using the Steamworks API[2]. In the data set, there are 23 features, but some of them are collected specifically for this study, which will be detailed in the next chapter.

3.3 Data Preparation

It is widely understood that in every data science lifecycle, the data preparation step usually takes up to half of the overall project's time. It is therefore one of the most critical steps, since clean data will create a more consistent and reliable dataset, on top of being more accurate and consistent. During this step, the data is cleaned to remove inaccurate information and prepared for training and evaluation by converting the text into vectors and splitting the data. For the purpose of this experiment, two data preparation phases were undertaken. One of which was applied to all text transforms and the other of which was done specifically for each text transformer. Following is a description of the steps taken to prepare the data:

3.3.1 Removing of Irrelevant Data

Data cleaning begins with the removal of unwanted features and observations from the dataset. In this experiment, it was conducted by analyzing the most popular games among users and removing irrelevant observations, a target dataset was selected based on specific features. Irrelevant observations and features pertain to observations and features that do not address the specific problem the researcher is trying to solve.

3.3.2 Handling Missing Data

Conducted this step in two parts, first before deep cleaning of reviews for each specific text transformer. Secondly, the cleaning of reviews for each text transformer was completed, and the new features were added to the dataset with their specific cleaning.

[1] https://www.kaggle.com/najzeko/steam-reviews-2021

[2] https://partner.steamgames.com/doc/store/getreviews

For both parts all missing values were discarded completely from the reviews column. As a result, text data cannot be treated in the same manner as numeric data in the event that a value is missing.

3.3.3 Data Type Conversion

Conversion of type refers to changing the type of data in each column to the appropriate one. As the target variable of this experiment in dataset is categorical boolean, It need to be changed to numeric boolean and all reviews need to be changed to string format.

3.3.4 Text Preprocessing

3.3.4.1 Lowercasing

Lowercaseing method is a popular text preprocessing method. Input text is converted into the same casing format, such that 'text', 'text', and 'TEXT' are all treated equally. lowercase is applied to all text, because it has been proven that machine learning models may treat lowercase differently from uppercase. Using this approach is particularly helpful for text feature extraction techniques such as TFIDF. This is because it facilitates the combination of the same words, which reduces duplication and allows correct counts and TFIDF values to be obtained.

3.3.4.2 Fix structural errors

In the text, multiple spaces have been replaced with a single space in order to make the texts more readable .After that the contraction verbs in the text were reformed to their full forms, such as "n't" to "not". For preparing the text for TF-IDF, compound nouns that contained hyphens were divided into parts and the numbers were eliminated.

3.3.4.3 Removing of Punctuation

The removal of punctuation from text data is another common technique for preprocessing text data in order to improve its readability. The punctuation can be added or removed according to the needs of the experiments. To prepare the text for TF-IDF, punctuation was replaced with spaces from the text. In the other hand, for preparing the text for BERT and SBERT, because these two embedding were trained on Wikipedia data, it may result in a more accurate embedding when including the numbers and some of the punctuation and the hyphenated compound nouns in the text. These punctuation marks (?, !,& .) remain to allow SBERT to detect the end of a sentence.

3.3.4.4 Removing Low Frequency words

It is possible that a domain-specific corpus will contain some flow-recurring words that are of less importance to experiment with specifically for TF-IDF which are less of an issue. In order to prepare a corpus for the TF-IDF, low frequency words were removed from the text.

3.3.4.5 Stemming

Stemming is a method of normalizing words in natural language processing tasks by reducing them to their root form. The purpose of stemming is to reduce the amount of time needed to search for a particular word or phrase by grouping together words that have similar meanings but may vary in their inflections or endings. It is typically accomplished using a rule-based approach that removes suffixes (such as "s," "ly," "es," and "ing") from a word.

While stemming can be a useful tool for natural language processing tasks, it does have some limitations. Because it is based on a set of rules, it may not always produce the correct stem for a given word. Additionally, this method does not consider the context or parts of speech of the words, potentially leading to less accurate results in some situations. For this reason, lemmatization, which takes into account the context and parts of speech of the words, is often preferred over stemming in more complex natural language processing tasks.

3.3.4.6 Lemmatization

Lemmatization is a process that involves reducing words to their base form, known as lemmas. This is similar to stemming, which involves reducing words to their root form. However, lemmatization takes into account the parts of speech and the context in which the words are used, while stemming does not. It can be useful for natural language processing tasks, such as information retrieval and text classification, because it allows words with similar meanings to be treated as the same word, even if they have different inflections or endings. This can help improve the accuracy of the model and the relevance of the search results.

Lemmatization can be performed using POS tags, which provide additional context about the word's role in the sentence. This can help to determine the most appropriate lemma for the word. For example, the word "jumps" might be tagged as a verb, while the word "jump" might be tagged as a noun. The lemmatizer would then use this information to determine the correct lemma for each word. Overall, it can be a useful tool for natural language processing tasks, as it allows for the reduction of words to their base form while taking into account their parts of speech and context. This can help improve the accuracy and relevance of the results.

3.4 Text Feature Extraction Techniques (Vectorization):

In order to train a machine learning model, numerical data must be used because machines are only able to process numerical inputs. While text data is frequently utilized with machine learning techniques, it must be transformed into a recognizable form for the algorithm before it can be used. This process of converting text into numerical vectors is called vectorization and is necessary because the machine learning algorithm cannot directly process text data. By vectorizing the text, the algorithm is able to understand and work with the data in a way that it is capable of processing. For the purpose of this study, three text transformers will be used: tf-idf, BERT, and SBERT. These transformers will be explained in detail in following section.

3.4.1 TF–IDF

TF-IDF (Term Frequency - Inverse Document Frequency) is a statistical measure used to evaluate the importance of words in a document or collection of documents. It is calculated by multiplying the term frequency (TF) of a word in a document by the inverse document frequency (IDF) of the same word in a collection of documents. Term frequency is simply the word's frequency of appearance in the document. To find the frequency of a word in a document, divide the number of times it appears by the total number of words in the document. Inverse document frequency is a measure of how common or rare a word is across a collection of documents. To calculate the inverse document frequency (IDF), the logarithm of the total number of documents is taken in a collection, divided by the number of documents containing the word. The resulting value is then multiplied by the term frequency of the word in the document in question.

The combination of term frequency and inverse document frequency allows for the importance of words to be evaluated within the context of a specific document, as well as relative to the rest of the collection of documents. Words that are more common across the entire collection of documents will have a lower inverse document frequency, and therefore a lower overall weight in the calculation. On the other hand, words that are rare or specific to a particular document will have a higher inverse document frequency, and therefore a higher overall weight in the calculation. TF-IDF is often used in information retrieval and machine learning tasks, such as building search engines, summarizing documents, and other natural language processing and text analysis tasks. It is a simple and intuitive approach that can provide useful insights into the importance and relevance of words in a given context.

$$TFIDF\ score\ for\ term\ i\ in\ document\ j = TF(i,j) * IDF(i)$$

$where$

$IDF = Inverse\ Document\ Frequency$

$TF = Term\ Frequency$

$$TF(i,j) = \frac{\text{Term i frequency in document j}}{\text{Total words in document j}}$$

$$IDF(i) = \log_2\left(\frac{\text{Total documents}}{\text{documents with term i}}\right)$$

and

$t = Term$

$j = Document$

Fig. 3.3 TF-IDF

3.4.2 BERT

BERT, or Bidirectional Encoder Representations from Transformers, is a natural language processing (NLP) model developed by Google in 2018. It is based on the concept of word embeddings, which are numerical representations of words that capture the semantic meaning of the words in a given context. One of the key features of BERT is its ability to process text in a bi-directional manner. This means that it takes into account the context of words not only from the words that come before them in a sentence, but also from the words that come after them. This allows BERT to better understand the meaning of words and the relationships between them, as well as the overall context of the sentence or document.

BERT is used in a wide range of applications, including language translation, question answering, and text classification. It has been shown to be highly effective at improving the performance of natural language processing tasks, and has become one of the most widely-used models in the field. BERT is trained using large amounts of labeled and unlabeled text data, and can be fine-tuned for specific tasks by adding additional layers to the model.

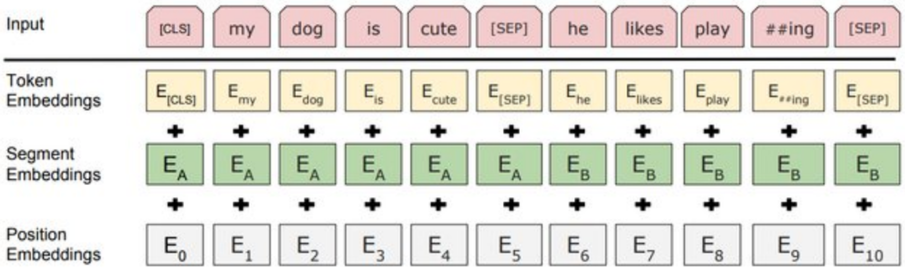


Fig. 3.4 BERT (Devlin et al., 2019)

3.4.3 SBERT

SBERT is a variant of BERT, a popular language model developed by Google, that is specifically designed to generate sentence embeddings. In the SBERT model, two sentences are input into the network as part of a Siamese architecture, which means that they are processed by the same set of layers in the network. The two input sentences are first passed through a BERT model to generate token embeddings, which are then pooled together to create a fixed-size embedding for each sentence. The obtained sentence embeddings can be applied to various natural language processing tasks, such as natural language inference, in which the aim is to determine the relationship between two sentences (e.g., whether one sentence contradicts or supports the other). By fine-tuning SBERT on natural language inference data, it becomes capable of encoding the semantics of sentences in a way that is useful for this task.

The fact that SBERT is computationally efficient makes it suitable for real-time search applications, where it is important to be able to process queries and return results quickly. This is because SBERT requires fewer computational resources to run compared to some other natural language processing models, such as BERT, which is a very large model. In addition to being computationally efficient, SBERT is also relatively easy to use. It can be fine-tuned on a variety of natural language processing tasks with minimal task-specific modifications, which makes it a convenient choice for researchers and practitioners who want to apply it to different types of problems. This is because SBERT is designed to generate sentence embeddings, which can be used as input to a wide range of downstream tasks. As a result, it is often possible to fine-tune SBERT on a new task with minimal task-specific modifications, which makes it a convenient choice for many natural language processing applications.

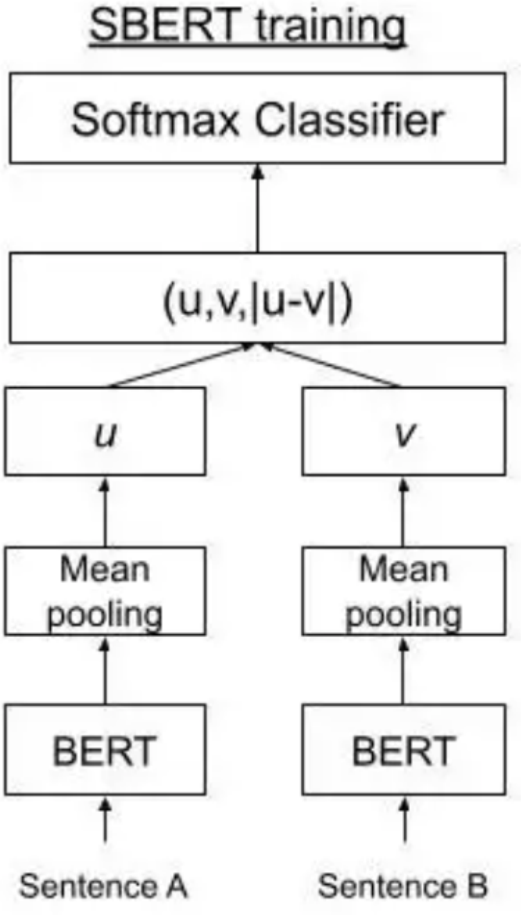


Fig. 3.5 SBERT

3.5 Modelling

Based on a review of previous literature, it was determined that the Support Vector Machine (SVM) Classifier was the most appropriate model for use in this experiment due to its high accuracy rating. The decision to use the SVM Classifier was based on the results of previous research in this area, which demonstrated that this model consistently performed well on a variety of tasks. As a result, it was considered the most suitable model for use in the current experiment.

In a study by Alzami et al. (2020), the highest accuracy results in the field of sentiment analysis of reviews were obtained by using the combination of the Term Frequency-Inverse Document Frequency (TF-IDF) text vectorization technique and the Support Vector Machine (SVM) classification technique. The accuracy rate of this approach was 88.6%, which was the highest accuracy rate reported in the literature for this specific area of research. These results demonstrate the effectiveness of using the TF-IDF technique in combination with the SVM classifier for sentiment analysis of reviews.

The Support Vector Machine (SVM) classifier has been shown to be capable of achieving high performance in sentiment analysis of reviews in previous research (Arief & Deris, 2021; Jeffrey et al., 2020) . As a result, this current research builds upon the findings of these previous studies by using the combination of the Term Frequency-Inverse Document Frequency (TF-IDF) text vectorization technique and the SVM classifier as the baseline model. This approach was chosen due to the demonstrated effectiveness of the SVM classifier in sentiment analysis tasks, as well as the versatility and effectiveness of the TF-IDF technique for text representation.

Based on a review of previous literature, it has been observed that transformer methods, such as BERT (a deep learning transformer), may be utilized to improve results in sentiment analysis. Therefore, this study proposes the construction of models using BERT (a word embedding technique) for extracting text features in conjunction with the Support Vector Machine (SVM) classifier, as well as SBERT (a sentence embedding technique) for extracting text features in combination with the SVM classifier. The accuracy results of all three models will be compared in the following section. This approach was selected due to the demonstrated effectiveness of transformer methods, such as BERT and SBERT, in various natural language processing tasks, including sentiment analysis.

3.6 Model Evaluation method

In this research study, the performance of the different models will be evaluated using a range of metrics, including accuracy measures, F1 scores, recall scores, precision scores, and confusion matrix performance metrics. To obtain a comprehensive assessment of model performance, the accuracy of the models will be calculated using ten-fold cross validation. This involves dividing the dataset into ten equal parts, training the model on nine of the parts and evaluating it on the remaining part, and repeating this process ten times so that each part is used as the test set once. The mean accuracy across all ten iterations will be calculated and used as an overall measure of model performance.

In order to determine whether the differences in performance between the different models are statistically significant, normality and statistical tests will be conducted on the accuracy scores derived from the ten-fold cross validation. The Shapiro-Wilk test will be used to assess whether the scores are normally distributed, and if the data is normally distributed, the Student t-test will be used to determine whether the differences in performance are statistically significant. If the data does not follow a normal distribution, the Mann-Whitney U test will be used to determine whether the differences in performance are statistically significant. By conducting these tests, it will be possible to determine whether the results of the cross validation are statistically significant and whether the hypothesis can be accepted or rejected.

3.6.1 Accuracy

A classification model’s accuracy can be defined in terms of its capacity to predict the true class labels of a dataset. It is calculated by dividing the number of correct predictions generated by the model by the total number of predictions, and expressing the result as a percentage. To calculate the accuracy of a model, the predictions made by the model are compared to the true class labels of the examples in the dataset. If the prediction is correct, it is considered a true positive or true negative (depending on the class); if the prediction is incorrect, it is considered a false positive or false negative.

To determine the accuracy of a classification model, the total number of true positives and true negatives is divided by the total number of predictions. Accuracy is a valuable measurement for comparing a model's predictions to the true labels and is often used in conjunction with other evaluation metrics, such as precision, recall, and F1 score, to gain a more thorough understanding of the model's performance.

$$Accuracy = \frac{TrueNegatives + TruePositive}{TruePositive + FalsePositive + TrueNegative + FalseNegative}$$

Fig. 3.6 Accuracy

3.6.2 Precision

Precision is a measure of the accuracy of positive predictions made by a classification model and is calculated by dividing the number of true positive predictions made by the model by the total number of positive predictions made. It is useful for evaluating a model's ability to correctly identify positive examples and is particularly important in situations where false positive predictions should be minimized, such as in medical diagnosis or fraud detection. While precision is a valuable metric for evaluating a classification model's performance, it should be considered alongside other metrics, such as recall, to provide a comprehensive understanding of the model's capabilities.

$$Precision = \frac{TruePositive}{TruePositive + FalsePositive}$$

Fig. 3.7 Precision

3.6.3 Recall

Recall is a performance evaluation metric commonly used in the field of machine learning, specifically for classification models. In some terminology, this is known as the sensitivity of the model, which is expressed as the percentage of correct predictions from the model for actual positive examples. A recall calculation is obtained through dividing the number of true positive predictions generated by the model In accordance with the number of examples in the dataset that were positive. This metric is important because it allows us to assess the model's ability to identify all of the positive cases are present in the data. The importance of recall should be considered in the evaluation of the performance of a classification model, and it is often combined with other metrics like precision in order to provide a comprehensive assessment of the model's capabilities.

$$Recall = \frac{TruePositive}{TruePositive + FalseNegative}$$

Fig. 3.8 Recall

3.6.4 F1 Score

In terms of classification models, the F1 score is an evaluation metric based on the combination of precision and recall of a classification model. It is determined by taking precision and recall's harmonic meaning , with a higher score representing better performance. Researchers benefit from the F1 score since it includes both precision and recall. This can be helpful for comparing different models or for determining the relative importance of precision and recall in a specific context. Overall, the F1 score is useful in evaluating classification models' performance.

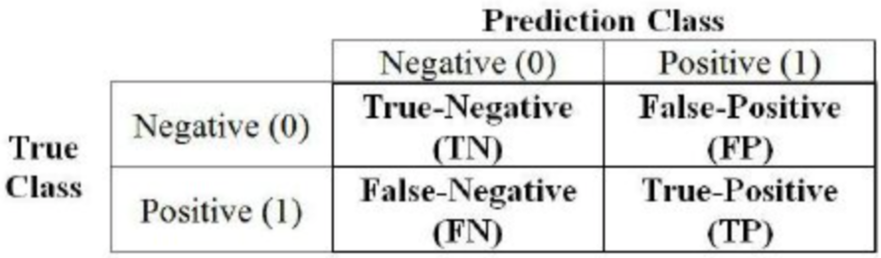
$$F1\ score = 2 * \frac{Precision\ * Recall}{Precision + Recall}$$

Fig. 3.9 F1 Score

3.6.5 Confusion Matrix

A comparison table, known as a confusion matrix, is created by taking the predicted values of a classification model and comparing them to the actual values from the data. The number of correct and incorrect predictions made by the model are shown, and the matrix is based on four possible combinations of predicted and actual values: true positives (TP), false positives (FP), false negatives (FN), and true negatives (TN). An accurate prediction of a positive outcome is considered a true positive. A false positive is an incorrect prediction of a positive outcome. False negatives are instances in which the model incorrectly predicts a negative result, and true negatives are instances in which the model correctly predicts a negative result.

Perception of the model's efficiency can be gained through the use of a confusion matrix, which permits visualization of the model's performance. This matrix can help to identify where the model may be committing errors. In addition to providing insights into the strengths and weaknesses of a classification model, it is a useful method for assessing its performance.



		Prediction Class	
		Negative (0)	Positive (1)
True Class	Negative (0)	**True-Negative (TN)**	**False-Positive (FP)**
	Positive (1)	**False-Negative (FN)**	**True-Positive (TP)**

Fig. 3.10 Confusion Matrix

3.6.6 K-Fold Cross-validation

The procedure of cross-validation is often utilized in the realm of applied machine learning in order to assess the model's performance on data that it has not previously seen. This resampling method helps to ensure that the model's results are consistent and not overly influenced by the specific subset of data it was trained on. It involves dividing the data into a specified number of folds, training and evaluating the model on each fold in turn, and averaging the resulting accuracy scores to give an estimate of the model's performance.

One type of cross-validation is 10-fold cross-validation, which involves dividing the data into 10 folds and training and evaluating the model 10 times, using each fold as a test set in turn. This process is repeated for each fold, and the resulting accuracy scores are averaged to give an estimate of the model's performance on unseen data. 10-fold cross-validation is a useful tool for evaluating the accuracy of a machine learning model and estimating its ability to generalize to unseen data. It is computationally efficient, widely used and well-understood, and can be adapted to different types of data.

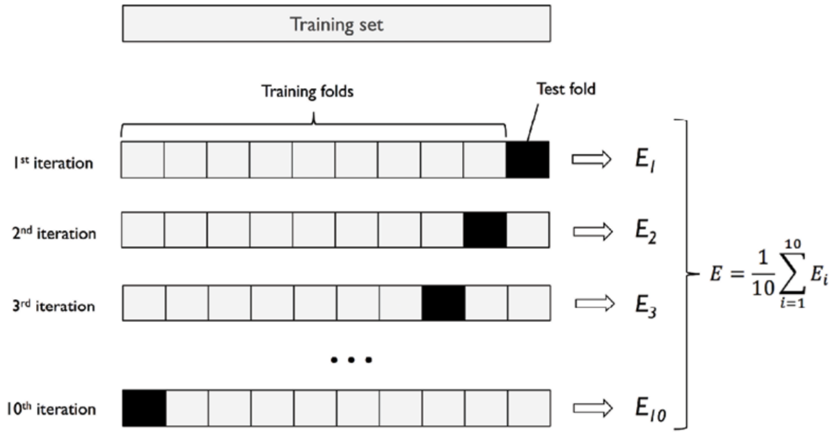


Fig. 3.11 Calculate the mean score of the model performance obtained by using the K-fold method

3.7 Statistical Analysis

In this study, statistical analysis is needed to accept or reject hypotheses. Two approaches that can be used for this purpose are the Mann-Whitney U test and the student t-test. To determine which of these tests to use or whether to use both, it is necessary to first assess the normality of the data. The Shapiro-Wilk test can be conducted to determine the normality of the data as the sample data is big enough to use this approach (Hanusz et al., 2016). If the data is found to be normally distributed, the student t-test should be applied (Mishra et al., 2019). On the other hand, if the data is not normally distributed, the Mann-Whitney U test should be employed (Kasuya, 2001). By carefully considering the normality of the data and selecting the appropriate statistical test, the reliability and validity of the results can be ensured.

3.7.1 Shapiro-Wilk Test

The Shapiro-Wilk test is a statistical test used to assess the normality of a data. This is important because the normality of a data can affect the choice of statistical tests and the reliability of the results. To conduct the Shapiro-Wilk test in pandas, the scipy.stats.shapiro() [3]function can be used. This function takes in a data as an input and returns two values: the test statistic and the p-value. The p-value is a measure of the probability that the data follows a normal distribution. If the p-value is below a certain threshold (usually 0.05), it can be concluded that the data is not normally distributed. On the other hand, if the p-value is above the threshold, it can be assumed that the data is normally distributed.

3.7.2 Student t-test

The Student's t-test is a parametric test that is used when the data is normally distributed and the variances of the two groups are equal. It compares the means of the two groups and determines the probability that the observed difference between the means could have occurred by chance. The test is named after William Sealy Gosset, who developed it while working as a statistician at the Guinness Brewery in Dublin, Ireland. Gosset published the test under the pseudonym "Student" in 1908 due to the company's policy

[3] https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.shapiro.html

at the time of not allowing publication of work done by its employees. There are two main types of Student's t-test: the one-sample t-test and the two-sample t-test. If the data in this study were normally distributed, the two-sample t-test would be the appropriate statistical test to use.

3.7.3 Mann-Whitney U test

The Mann-Whitney U test, also known as the Mann-Whitney-Wilcoxon (MWW) rank sum test or the Wilcoxon rank sum test, is a nonparametric statistical hypothesis test used to compare the medians of two independent samples. It is often used as an alternative to the t-test when the data are not normally distributed or when the variances of the two samples are not equal. The test is named after Henry B. Mann and Donald R. Whitney, who developed it in the 1940s. It is based on the concept of ranking the values in the two samples and comparing the ranks of the observations in the two samples.

To perform the Mann-Whitney U test, the observations from both samples are first ranked together. The test statistic, U, is then calculated based on the ranks of the observations in the two samples. If U is large enough, it can be concluded that the medians of the two samples are significantly different. The Mann-Whitney U test is a two-tailed test, meaning that it can detect differences in either direction (i.e., one sample having a median that is either higher or lower than the median of the other sample). It is also known as a non-directional test because it does not assume that one sample has a higher median than the other.

3.8 Delimitation and scope

This experiment aimed to evaluate the effectiveness of different text transformation techniques for analyzing the sentiment of reviews using support vector machine (SVM) classifiers. The techniques that were compared included term frequency-inverse document frequency (TF-IDF), BERT (Bidirectional Encoder Representations from Transformers), and SBERT (Sentence-BERT). The experiment was conducted using the Steam Review dataset from Kaggle as the source of review data. The entire experiment was developed using the Python programming language.

The initial phase of the study was conducted using Jupyter Notebook, a widely-used tool for interactive computing and data visualization. Jupyter Notebook was selected for its convenience in local coding and its ability to facilitate easy sharing and collaboration. However, the majority of the project was ultimately completed using Google Collaboratory due to issues that arose when attempting to run transformer-based language model embeddings (BERT and SBERT) in Jupyter Notebook. Google Collaboratory is a cloud-based platform that provides an interactive computing environment and enables the execution of code and the creation of documents that contain live code, equations, and visualizations. It was chosen as an alternative to Jupyter Notebook due to the issues that arose during the initial phase of the study.

To run the scripts in this project, several libraries and modules were required. These included NumPy, a library for scientific computing with Python; Scikit-Learn, a library for machine learning; Pandas, a library for data manipulation and analysis; MXNET, a library for deep learning; NLTK, a library for natural language processing; matplotlib, a library for data visualization; bert_embedding, a library for BERT embeddings; and sentence_transformers, a library for SBERT embeddings. These libraries and modules were used to transform the text data, apply SVM classifiers, and visualize the results of the experiment.

3.9 Strength and Limitation of Approach

In this research, there are some limitations to the use of BERT and SBERT, as these models are typically used to train deep learning algorithms, but they have been applied to train a machine learning algorithm using the Support Vector Machine (SVM) classifier. The decision to use the SVM classifier with a TF-IDF text representation technique as the baseline was based on a previous study (Alzami et al., 2020). However, it is important to note that the accuracy achieved in this research may not be directly comparable to the accuracy obtained in the previous study due to the use of a different dataset. This means that the results of this research may not be directly comparable to those of the previous study, as the different datasets and approaches used could potentially impact the accuracy of the results.

The process of understanding the datasets was a crucial step in the data preparation process, as it allowed for the identification of a specific target data and tailoring the process to its specific characteristics. By taking this approach, a customized data preparation process was created that was tailored specifically to the target data, rather than relying on a generic approach that may not be effective for all datasets. This allowed for more effective preparation of the data and ensured that the process was optimized for the specific characteristics of the target data. Overall, the time spent on understanding the datasets and customizing the data preparation process was well worth the effort, as it allowed for better results and more accurate analysis of the data.

Prior to this study, the application of BERT as a word embedding and SBERT as a sentence embedding in the sentiment analysis of game reviews had not been previously investigated. This gap in the research was identified through a comprehensive review of the literature. The utilization of embedding techniques, which capture the contextual meaning of words within a body of text, has the potential to offer valuable insights for conducting effective text analysis. As such, exploring the use of these text representation techniques in the sentiment analysis of game reviews represents a promising avenue for future research.

Chapter 4

Results, evaluation and discussion

This chapter aims to provide a thorough understanding of the methods used to conduct the research outlined in Chapter 3. This includes detailing the steps taken to design and execute the experiments, as well as reporting the findings. To determine whether the Null and Alternate Hypotheses can be accepted or rejected, a statistical difference test will be applied to the results of the various models. This will allow for the determination of whether there is a significant difference between the results of the models and the Null Hypothesis, or if the results support the Alternate Hypothesis. By analyzing the results in this way, conclusions can be drawn about the effectiveness of the different models and their ability to accurately predict the outcomes of the experiments.

4.1 Data Preparation

4.1.1 Removing of Irrelevant Data

The Steam data set contains 23 features, but some of them were collected for the purpose of this study, as explained in the following section which details come from steam documentation[1]:

1. **Game**

   language: language of the review

[1] https://partner.steamgames.com/doc/store/getreviews

review : text of written review

app_name : Name of game

2. **Author**

   steamid : the user's SteamID

   num_games_owned : number of games owned by the user

   num_reviews : number of reviews written by the user

   playtime_forever : lifetime playtime tracked in this app

   playtime_at_review : playtime when the review was written

3. **Game Recommendation**

   recommended : True and False

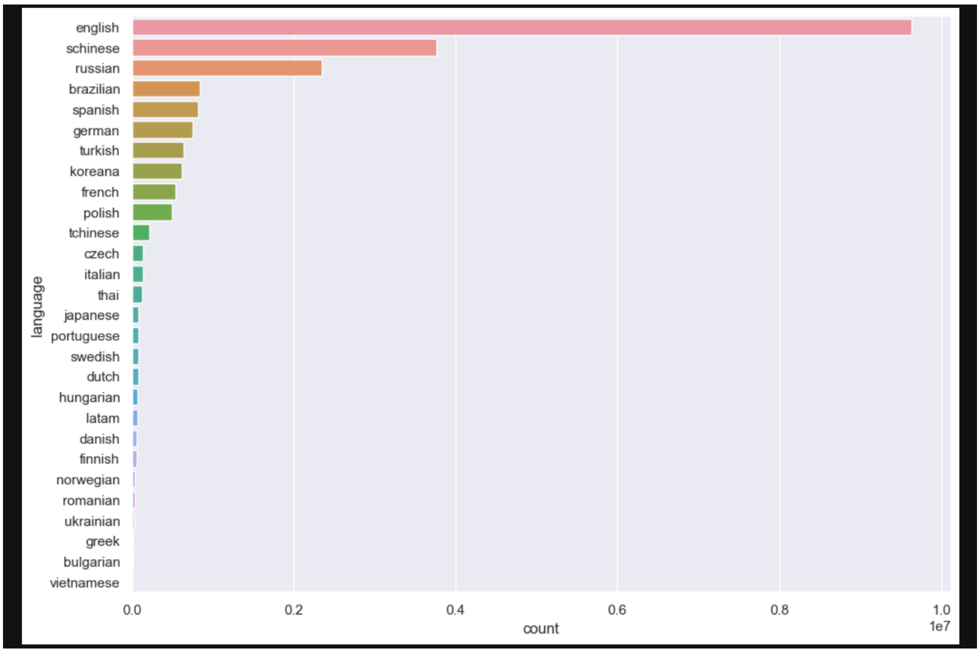


Fig. 4.1 Different Languages of Reviews

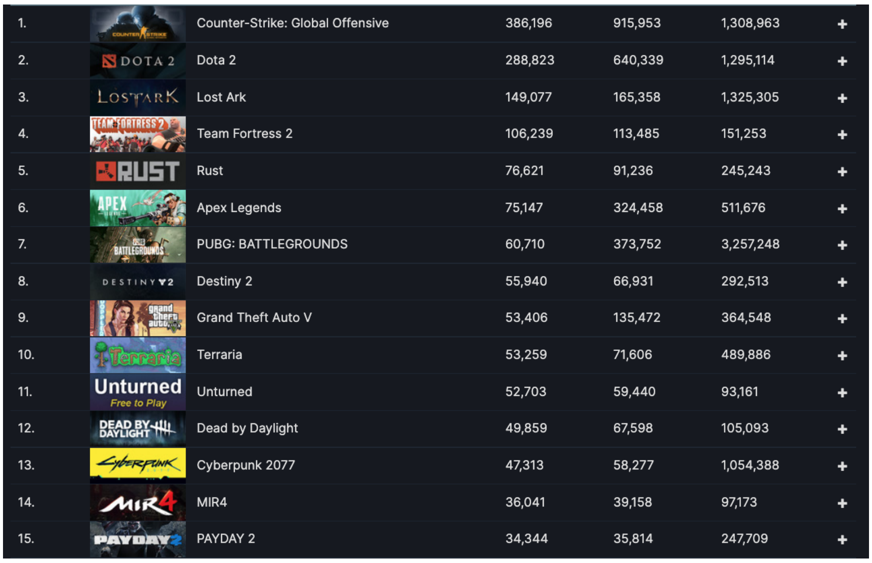
Four games with a high number of active users and a significant number of English reviews were selected from the dataset of all existing games, specifically from the most 15 popular games according to Steam Info[2] as it is obvious in Figure 4.2 . The target data that was chosen had around 25,000 reviews in total. The selection of active

[2] https://steamdb.info/graph/

users was based on the following criteria:

1. **English reviews.** As depicted in Figure 4.1, English reviews make up the majority of the reviews, therefore only English reviews were selected.
2. **The number of games owned by the user on Steam.** After sorting and calculating the number of games owned by each user, it was found that three users had the most games. In order to calculate the mean number of games owned by each user, these three users were removed from the calculation. The resulting mean was 119.66 games. The data for users who owned more than 119.66 games, including the three users with the highest number of games, were then collected.
3. **Amount of reviews that have been written by users on Steam.** After finding the mean number of games owned on Steam for each user, the mean number of reviews written by these users was calculated. The mean number of reviews was found to be 8.33. The data for users who had written more than 8.33 reviews were then collected.
4. **The amount of time a user has spent playing on Steam over their lifetime on Steam.**The mean amount of time spent playing on Steam over a lifetime was calculated for each user. The mean time spent was found to be 11107.61 minutes. The data for users who had played for more than 11107.61 minutes were then collected.
5. **Time spent playing when the review was written by the user on Steam.** The mean playtime when writing a review was calculated for each user. The mean time spent was found to be 21638.03 minutes. The data for users who had played for more than 21638.03 minutes were then collected.

After considering all options, the final dataset chosen consisted of data from four games, as shown in Figure 4.3: "Grand Theft Auto V", "Rust", "Terraria", and



1.	Counter-Strike: Global Offensive	386,196	915,953	1,308,963	+
2.	Dota 2	288,823	640,339	1,295,114	+
3.	Lost Ark	149,077	165,358	1,325,305	+
4.	Team Fortress 2	106,239	113,485	151,253	+
5.	Rust	76,621	91,236	245,243	+
6.	Apex Legends	75,147	324,458	511,676	+
7.	PUBG: BATTLEGROUNDS	60,710	373,752	3,257,248	+
8.	Destiny 2	55,940	66,931	292,513	+
9.	Grand Theft Auto V	53,406	135,472	364,548	+
10.	Terraria	53,259	71,606	489,886	+
11.	Unturned	52,703	59,440	93,161	+
12.	Dead by Daylight	49,859	67,598	105,093	+
13.	Cyberpunk 2077	47,313	58,277	1,054,388	+
14.	MIR4	36,041	39,158	97,173	+
15.	PAYDAY 2	34,344	35,814	247,709	+

Fig. 4.2 The 15 most popular games on the Steam platform (November 2022)

"PAYDAY 2". These games were chosen for their relevance and popularity within the gaming community and the selection was made based on factors such as relevance, quality, and potential usefulness for the research. On the x-axis of Figure 4.3, the column labeled "app_name" displays the name of each game. On the y-axis, the column labeled "count" shows the number of reviews for each game.

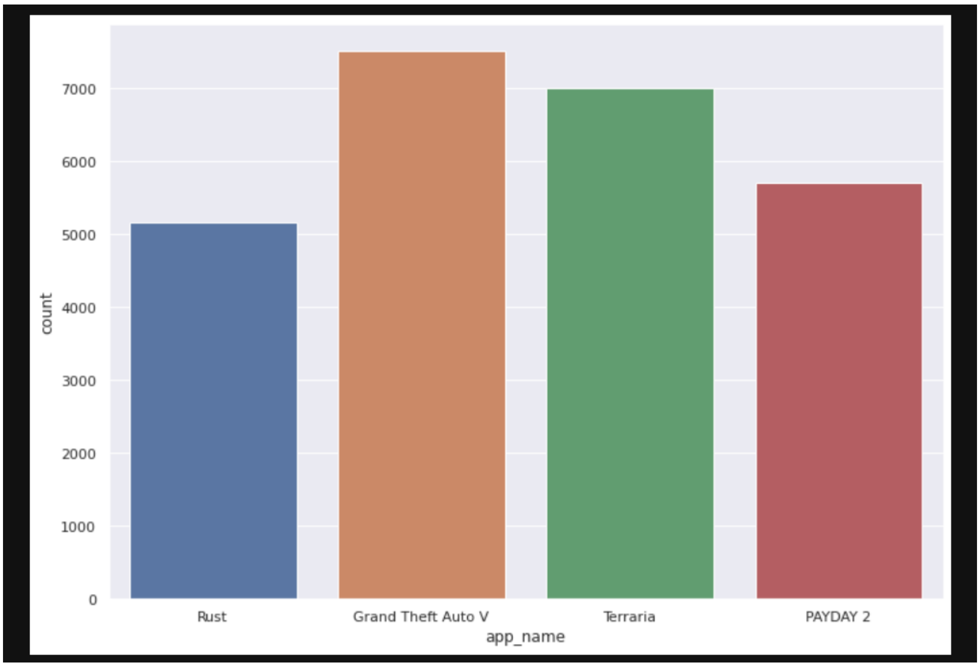


Fig. 4.3 Four Games Selected

4.1.2 Handling Missing Data

First Part:

During the initial data preparation process, it was identified that there were several instances of the string ’NA’ present in the text column. To maintain the accuracy and reliability of the data, the drop.na() method was used to eliminate all ’NA’ values from the dataset. This method effectively removes any rows in the dataset that contain missing or null values, which is crucial for accurately analyzing and interpreting the data.

Second Part:

Upon completion of the data preparation process prior to the text transformer phase, it was discovered that reviews with a minimum length of 3 caused errors during the text transformation process. As a result, all reviews with a minimum length of 3 were removed.

4.1.3 Data Splitting

The target data was split into train and test sets using Sklearn[3]’s train_test_split function, with a ratio of 80/20. This means that 80% of the dataset was used for the train set and 20% was used for the test set. Finally, the following data cleaning steps were applied to both the train and test sets.

4.1.4 Data Type Conversion

Conversion of type refers to changing the type of data in each column to the appropriate one. As the target variable of this experiment in dataset is categorical boolean, It need to be changed to numeric boolean and all reviews need to be changed to string format.

[3] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

4.1.5 Text Preprocessing

In the next steps, all reviews will be cleaned, reformatted, and added as three new columns to the train and test datasets and the result as an example is obvious in Figure 4.4. These columns are:

1. "prep_review" for use with the TF-IDF, BERT and SBERT text transformer.
2. "prep_review_tfidf" for use with the TF-IDF text transformer, based on the "prep_review" column.
3. "prep_review_bert_sbert" for use with the BERT and SBERT transformers, based on the "prep_review" column.

These columns will be used to prepare the reviews for the various text transformers mentioned: the TF-IDF transformer, the BERT transformer, and the SBERT transformer.

4.1.5.1 Lowercasing, Removing of Punctuation & Fixing structural errors

All reviews were converted to lowercase because it has been shown that machine learning models may treat lowercase and uppercase letters differently. This is especially important when performing text transformation techniques such as TF-IDF, BERT, and SBERT, as words with different casing styles may be treated as distinct entities. Then, to improve readability, multiple spaces in the reviews were replaced with a single space. After that, five contraction verbs in the review were reformed to their full forms, including "n't" to "not", "can't" to "can not", "'ve" to "have", "'re" to "are", "won't" to "will not", and "'ll" to "will.

Afterwards, to prepare the review for TF-IDF text transformation, the numbers were eliminated and compound nouns containing hyphens were divided into parts. In the following steps, in order to prepare the review for TF-IDF, all punctuation was replaced by spaces except for the following marks: (?, !, &, .). These marks were retained in the review for use with BERT and SBERT. Finally, all multiple spaces in the review for TF-IDF, BERT, and SBERT were replaced by a single space.

4.1.5.2 Stemming, Lemmatization and Removing Low Frequency words

The "word_tokenize" function from the tokenize [4] package in the nltk.stem library was utilized to divide each review in the dataset into a list of individual words, or tokens. This is a crucial step in the text preprocessing process as it allows the analysis to consider each word separately and analyze it in the context of the entire document. After the review text was tokenized into a list of words, the next step was to perform lemmatization and stemming.

Lemmatization refers to the reduction of words to their simplest form, also known as their lemma. This is useful because it allows the analysis to group together related words that may have different inflections. The WordNetLemmatizer() function from nltk.stem.wordnet[5] was used to perform lemmatization in this case.

The stem of a word is its base form, and stemming is the act of reducing a word to its stem [6]. This is useful because it allows the analysis to group together related words that may have different inflections or suffixes. The PorterStemmer() function from nltk.stem.porter [7] was used to perform stemming in this case. The resulting preprocessed review can then be used for further TF-IDF analysis.

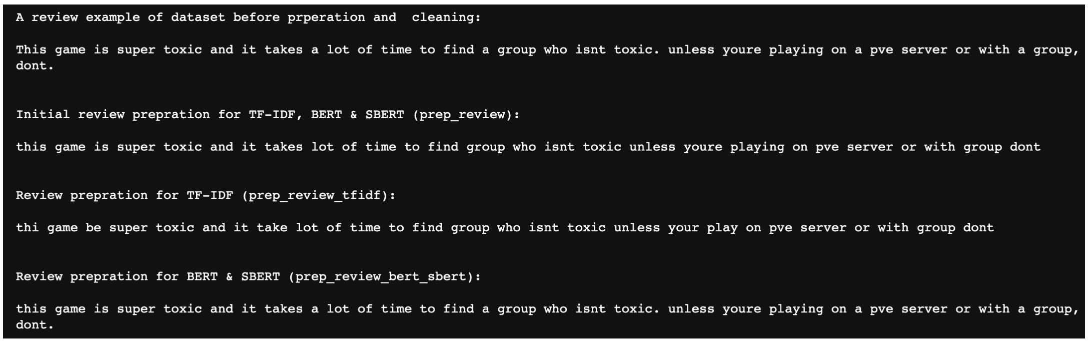
To prepare a corpus for a TF-IDF analysis, it is beneficial to remove low frequency words from the review. These are words that have been repeated fewer than 5 times in the entire dataset when considering the "prep_review_tfidf" column, which was created for the purpose of normalization in the TF-IDF process. By eliminating these words, the focus can be placed on more significant terms, leading to a clearer understanding of the main topics and themes in the corpus.

[4] https://www.nltk.org/api/nltk.tokenize.html

[5] https://www.nltk.org/_modules/nltk/stem/wordnet.html

[6] https://michael-fuchs-python.netlify.app/2021/05/31/nlp-text-pre-processing-iii-pos-ner-and-normalization/

[7] https://www.nltk.org/api/nltk.stem.porter.html



```
A review example of dataset before prperation and  cleaning:

This game is super toxic and it takes a lot of time to find a group who isnt toxic. unless youre playing on a pve server or with a group,
dont.


Initial review prepration for TF-IDF, BERT & SBERT (prep_review):

this game is super toxic and it takes lot of time to find group who isnt toxic unless youre playing on pve server or with group dont


Review prepration for TF-IDF (prep_review_tfidf):

thi game be super toxic and it take lot of time to find group who isnt toxic unless your play on pve server or with group dont


Review prepration for BERT & SBERT (prep_review_bert_sbert):

this game is super toxic and it takes a lot of time to find a group who isnt toxic. unless youre playing on a pve server or with a group,
dont.
```

Fig. 4.4 Result of Review Preparation

4.1.6 Text Feature Extraction Techniques (Vectorization):

After preparing all of the reviews and collecting them in specific columns with the names "prep_review_tfidf" and "prep_review_bert_sbert" the next step is to perform vectorization. Vectorization refers to the process of converting the text into numerical representations, or vectors, that can be used as input for machine learning models. In this case, vectorization will be performed on the "prep_review_tfidf" for TF-IDF and "prep_review_bert_sbert" for BERT and SBERT. These techniques each have their own unique ways of representing the text as numerical vectors, and they can be used to extract different types of features from the text that may be useful for classification.

4.1.6.1 TF–IDF

The TfidfVectorizer[8] from the scikit-learn package will be used to convert each statement into its vector representation. This vectorization mechanism has several defined parameters that can be specified. The following parameters will be applied, and all other parameters defined by the implementation will use the default values specified by the package. The n-gram range will be set to '1,2' to use unigrams and bigrams. Unigrams represent single words, and bigrams represent word pairs. By including word pairs, the data is encoded as specific columns in the data, forcing the pairs to be considered regarding significance and modeling. The max_features parameter is equal to 16000, which builds a vocabulary that only considers the top max_features ordered by term frequency across the corpus.

[8] https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

4.1.6.2 BERT (Word Embedding)

For the BERT (word embedding) method, the BertEmbedding from the bert-embedding[9] package will be used to convert each statement into its vector representation(to a 768-dimensional dense vector space) that is implemented with MXNet [10]. The BertEmbedding class converts a sequence of words or tokens into their corresponding BERT embeddings, which are dense, numerical vectors that capture the semantic and syntactic characteristics of the input text. These embeddings then be used as input to text classification for sentiment analysis, and machine translation. The bert-embedding package provides a convenient and efficient way to compute these embeddings in Python.

4.1.6.3 SBERT (Sentence Embedding)

For the SBERT (sentence embedding) method, the SentenceTransformer()[11] from the sentence-transformers package will be used to map sentences and paragraphs to a 768-dimensional dense vector space. The SentenceTransformer class, with the "bert-base-nli-mean-tokens"[12](SentenceTransformer(bert-base-nli-mean-tokens)), is a pre-trained model designed for generating sentence embeddings, which are numerical representations of the meaning and content of a sentence. this embedding was used as input to text classification. The sentence-transformers package provides a convenient and efficient way to compute SBERT embeddings in Python.

4.2 Modeling

The purpose of this research was to compare the performance of using BERT and SBERT as text representation techniques for sentiment analysis, versus using the traditional TF-IDF method. The study involved vectorizing all of the reviews using three different text transformation techniques: TF-IDF, BERT, and SBERT. The resulting features were split into training and testing sets, with "tfidf_features_train" and "tfidf_features_test" representing the TF-IDF features, "bert_features_train" and

[9] https://pypi.org/project/bert-embedding/

[10] https://mxnet.apache.org/versions/1.7/api/python/docs/api/mxnet/context/index.html

[11] https://www.sbert.net/

[12] https://huggingface.co/sentence-transformers/bert-base-nli-mean-tokens

"bert_features_test" representing the BERT features, and "sbert_features_train" and "sbert_features_test" representing the SBERT features. These features were then used to train and evaluate a support vector machine (SVM) classifier, with the goal of determining which text representation technique performed the best.

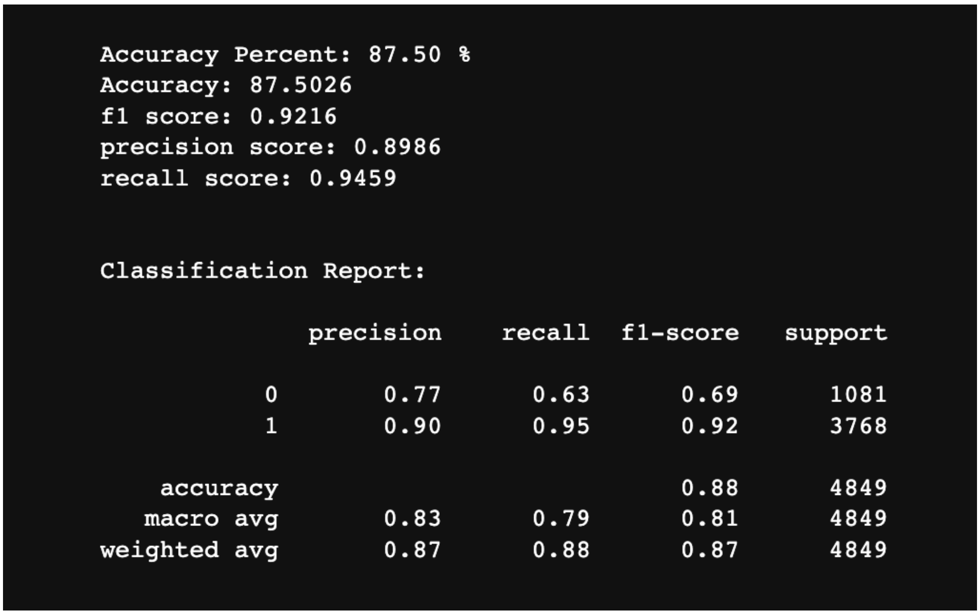
For this study, a support vector machine (SVM) classifier with a linear kernel and a C value of 1 was used to perform sentiment analysis. The SVM classifier was chosen because it has been found to produce the best results for sentiment analysis in previous literature. The specific SVM implementation used in this study was the C-Support Vector Classification provided by the scikit-learn library[13] (sklearn.svm.SVC). This implementation allows for the use of different kernel functions, and the linear kernel was chosen for this study. The C parameter determines the strength of the regularization, and a value of 1 was used in this study.

4.2.1 TF-IDF with SVM

In this study, a baseline model using the term frequency-inverse document frequency (TF-IDF) text representation technique and a support vector machine (SVM) classifier was implemented in order to reproduce the results of previous research (Alzami et al., 2020). That research found that using a SVM with a TF-IDF technique and selecting the top 16,000 features resulted in an accuracy of 87.30% when applied to an SVM model for sentiment analysis. In this current study, a similar level of accuracy was achieved with the baseline model, resulting in accuracy of 87.50% when applied to the Steam dataset using an SVM and TF-IDF.

The classification report for the steam dataset when using a support vector machine (SVM) model based on the term frequency-inverse document frequency (TF-IDF) representation can be found in Figure 4.5. This report provides information on the performance of the SVM model in classifying the data using the TF-IDF representation. It includes metrics such as precision, recall, and f1-score for each class, as well as an overall accuracy score.

[13] https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC



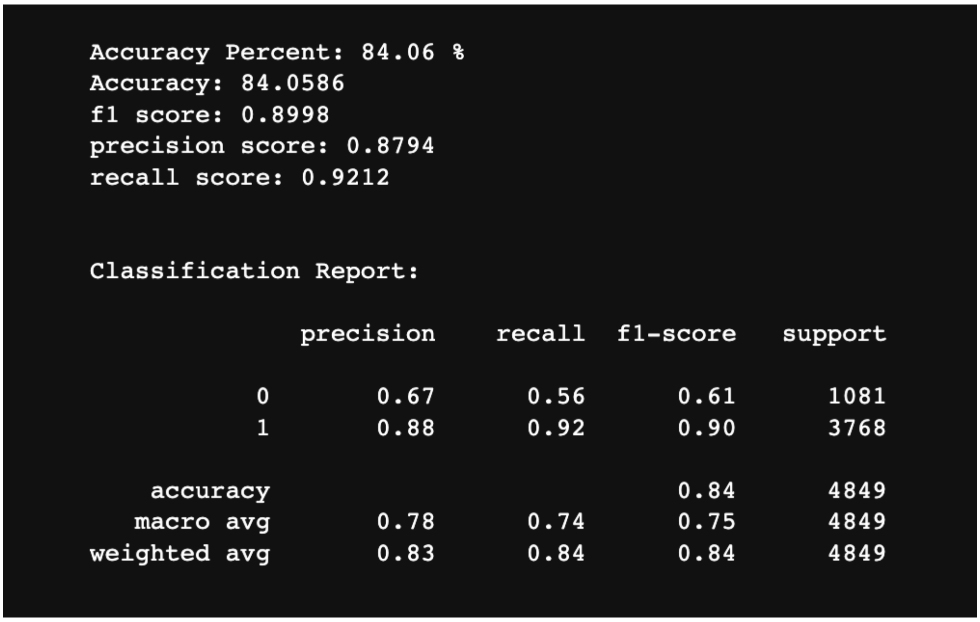
```
Accuracy Percent: 87.50 %
Accuracy: 87.5026
f1 score: 0.9216
precision score: 0.8986
recall score: 0.9459


Classification Report:

              precision    recall  f1-score   support

           0       0.77      0.63      0.69      1081
           1       0.90      0.95      0.92      3768

    accuracy                           0.88      4849
   macro avg       0.83      0.79      0.81      4849
weighted avg       0.87      0.88      0.87      4849
```

Fig. 4.5 TF-IDF with SVM Classification Report

4.2.2 BERT with SVM

The aim of this research was to evaluate the effectiveness of using BERT as a text representation technique compared to using TF-IDF. The BERT model was trained on data from the 'prep_review_bert_sbert' column. BERT transforms a sequence of words into a fixed-length vector representation, which captures the meaning and context of the words. This representation can be used as a feature in machine learning models for various NLP tasks. In the current study, it was found that using BERT-based features with a support vector machine (SVM) model resulted in an accuracy of 84.06% for sentiment analysis.

The classification report in Figure 4.7 shows the performance of an SVM model using SBERT as the word embedding for the steam dataset. The report includes metrics of evaluation such as f1-score, accuracy, precision and recall, which provide a summary of the model's performance.



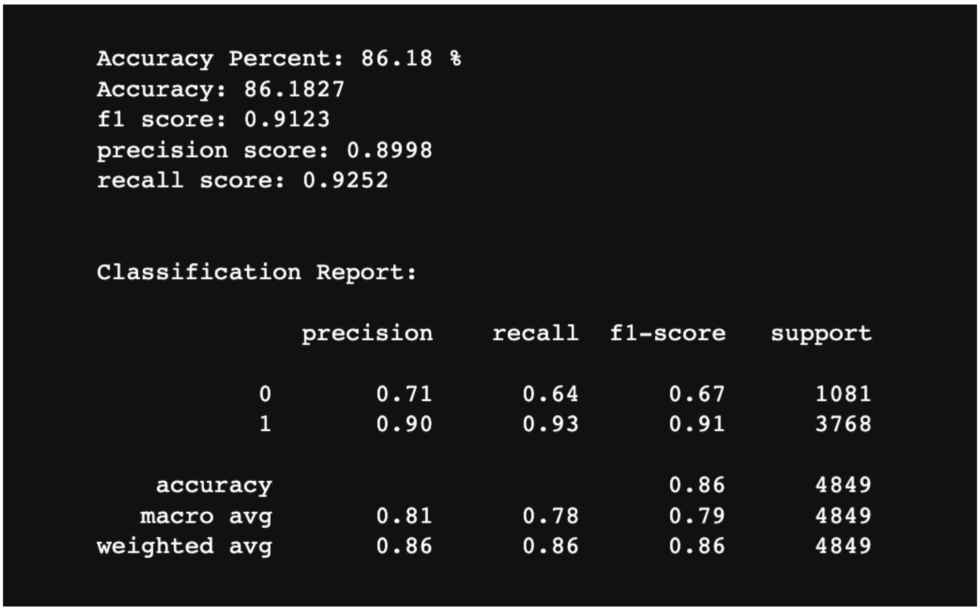
```
Accuracy Percent: 84.06 %
Accuracy: 84.0586
f1 score: 0.8998
precision score: 0.8794
recall score: 0.9212


Classification Report:

              precision    recall  f1-score   support

           0       0.67      0.56      0.61      1081
           1       0.88      0.92      0.90      3768

    accuracy                           0.84      4849
   macro avg       0.78      0.74      0.75      4849
weighted avg       0.83      0.84      0.84      4849
```

Fig. 4.6 BERT with SVM Classification Report

4.2.3 SBERT with SVM

The purpose of this study was to examine the effectiveness of using SBERT as a sentence embedding in comparison to using TF-IDF. The SBERT model was trained using data from the ’prep_review_bert_sbert’ column. As a sentence embedding, SBERT converts a sequence of words into a single numerical vector, known as an embedding, that encapsulates the meaning and context of the sentence as a whole. It converts a sequence of words as input into a fixed-length vector representation of the text as output This representation captures the meaning and context of the input text at the sentence level and can be utilized as a feature in machine learning models.

The classification report in Figure 4.7 shows the performance of an SVM model using SBERT as the sentence embedding for the steam dataset. The report includes metrics of evaluation such as f1-score, accuracy, precision, and recall, which provide a summary of the model’s performance.



```
Accuracy Percent: 86.18 %
Accuracy: 86.1827
f1 score: 0.9123
precision score: 0.8998
recall score: 0.9252


Classification Report:

              precision    recall  f1-score   support

           0       0.71      0.64      0.67      1081
           1       0.90      0.93      0.91      3768

    accuracy                           0.86      4849
   macro avg       0.81      0.78      0.79      4849
weighted avg       0.86      0.86      0.86      4849
```

Fig. 4.7 SBERT with SVM Classification Report

4.3 Evaluation

In this section, the results of all evaluation metrics for the three models (TF-IDF with SVM, BERT with SVM, and SBERT with SVM) will be presented, including the confusion matrix. The accuracy score was chosen as the primary metric for comparison and final evaluation when applying 10-fold cross validation. This is because accuracy is not affected by class imbalances in the data and it is important to minimize both false positives and false negatives. The results of the experiments were reported in the following sections, and a statistical test was performed to compare the results of the different models in order to determine whether to accept or reject the null and alternate hypotheses. The outcome of this test allowed for the conclusion of which model was the most effective and could be used with confidence in future projects.

4.3.1 Accuracy, F1-score, Precision and Recall

Table 4.1 presents the results of all evaluation metrics - including accuracy, F1 score, precision, and recall - for the three models: TF-IDF with SVM, BERT with SVM, and SBERT with SVM. As can be seen, the accuracy of the TF-IDF with SVM model is higher than that of the two BERT and SBERT models with SVM. However, the precision for the TF-IDF with SVM model is similar to that of the SBERT with SVM model, and the recall of the TF-IDF with SVM model is similar to that of the BERT with SVM model. The F1 scores of all three models are also almost similar for three models. It is not clear which of the three models (TF-IDF with SVM, BERT with SVM, and SBERT with SVM) has the best performance. Therefore, a confusion matrix was also generated for the three models, and further evaluation using 10-fold cross validation with accuracy is necessary to determine which model is the most effective.

Models	Accuracy	F1 Score	Precision	Recall
TF-IDF + SVM	0.87	0.92	0.89	0.94
BERT + SVM	0.84	0.89	0.87	0.92
SBERT + SVM	0.86	0.91	0.89	0.92

Table 4.1 The Result of Evaluation Metrics

For the rest of the study evaluation, the three models were evaluated and compared using only the accuracy score for both the 10-fold cross validation and the final step. This is because, it is important to minimize both false positives and false negatives. Moreover, accuracy is the most commonly used metric in literature on this topic for final evaluation. Beside of that, as shown in the Table 4.1 and 4.8, the precision for the TF-IDF with SVM classifier is similar to that of the SBERT with SVM classifier, and the recall for the BERT with SVM classifier is similar to that of the SBERT with SVM classifier, the accuracy was chosen.

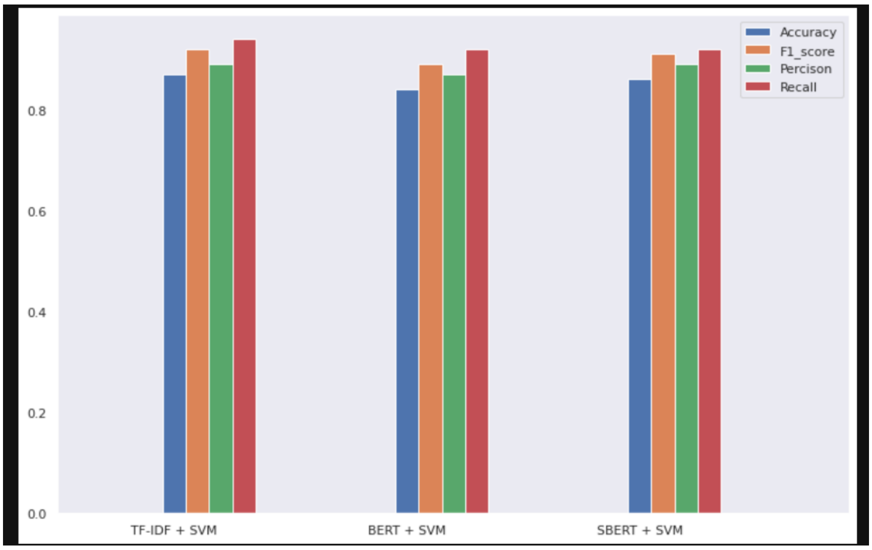


Fig. 4.8 The Result of Evaluation Metrics

4.3.2 Confusion Matrix

The results of the confusion matrix analysis, as presented in Figure 4.9, 4.10, 4.11, and Table 4.2, show that there are differences in the performance of the three models being compared in terms of their ability to accurately classify samples as either positive or negative.

Metric	TF-IDF+SVM	BERT+SVM	SBERT+SVM
False Negative	204 (4.21%)	297 (6.12%)	282 (5.82%)
False Positive	402 (8.29%)	476 (9.82%)	388 (8.00%)
True Negative	679 (14.00%)	605 (12.48%)	693 (14.29%)
True Positive	3564 (73.50%)	3471 (71.58%)	3486 (71.89%)

Table 4.2 Confusion Matrix Result

The TF-IDF model has the highest True Positive rate, meaning it is the most effective at correctly identifying positive samples. On the other hand, the BERT model has the lowest True Positive rate, indicating it is the least successful at accurately identifying positive samples. In terms of True Negative rate, the SBERT model performs the best, correctly identifying the highest number of negative samples, while the BERT model has the lowest True Negative rate, accurately identifying the fewest number of negative samples. The SBERT model also has the lowest False Positive rate, meaning it is the least likely to incorrectly classify a negative sample as positive. The BERT model has the highest False Positive rate, indicating it is the most likely to incorrectly classify a negative sample as positive. The BERT model also has the highest False Negative rate, meaning it is the most likely to incorrectly classify a positive sample as negative, while the TF-IDF model has the lowest False Negative rate, indicating it is the least likely to incorrectly classify a positive sample as negative.

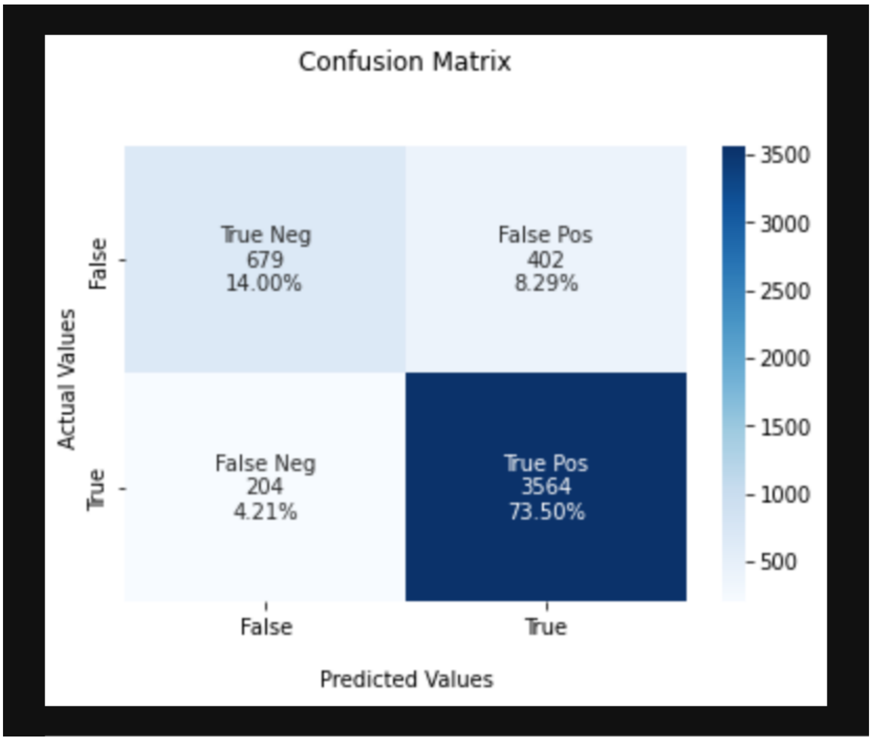


Fig. 4.9 Confusion Matrix of SBERT + SVM

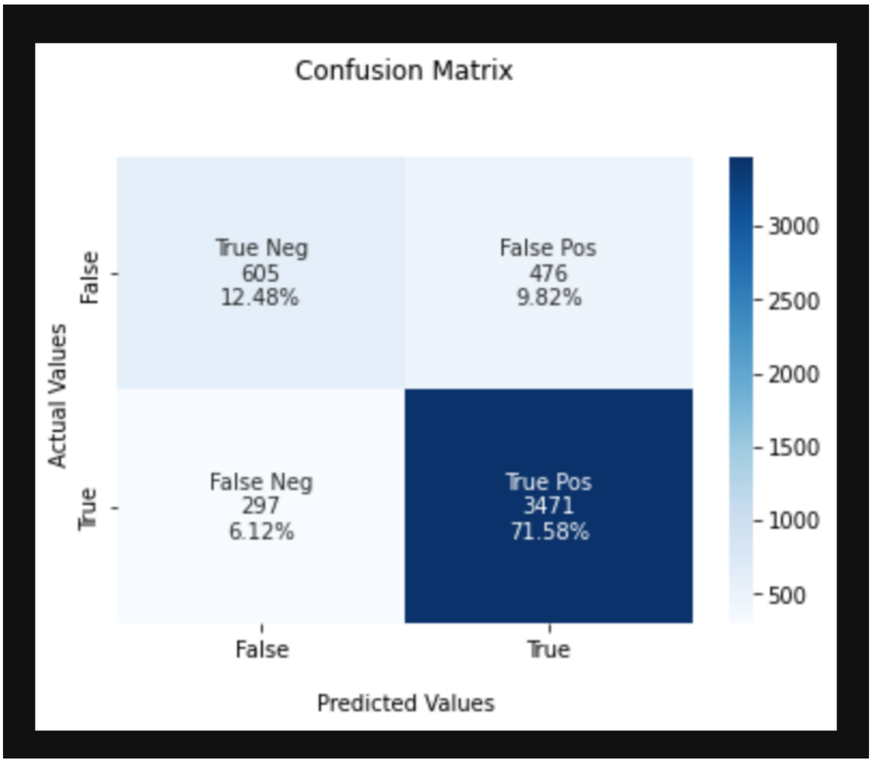


Fig. 4.10 Confusion Matrix of BERT + SVM

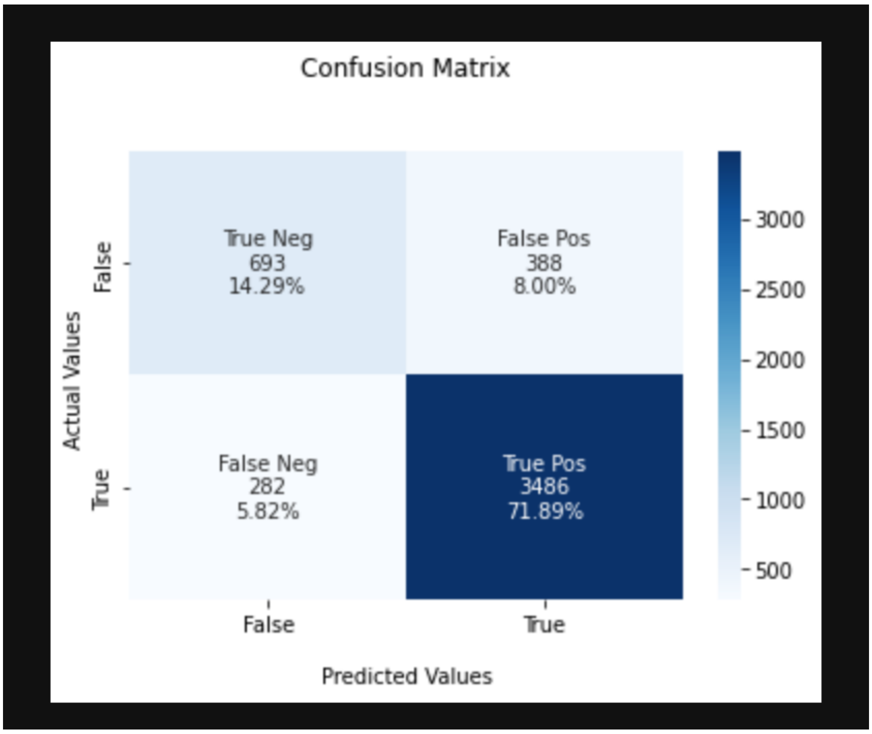


Fig. 4.11 Confusion Matrix of SBERT + SVM

4.3.3 10-fold Cross-Validation Mean Accuracy Scores

This section presents the mean test accuracy scores obtained by applying the 10-fold cross-validation method to the steam datasets using RepeatedStratifiedKFold[14]from sklearn.model_selection. This method, which is effective for classification problems with severe class imbalances, allows for the estimated performance of a machine learning model to be improved by repeating the cross-validation procedure multiple times (as specified by the n_repeats[15] parameter) and reporting the mean result across all folds from all runs. In this study, n_repeats is set to 3, resulting in a total of 30 folds being used as it is obvius in Figure 4.12, 4.13, 4.14. The mean result is expected to be a more accurate estimate of the model's performance. The cross_val_score[16] method from the sklearn.model_selection library in Python was also used.

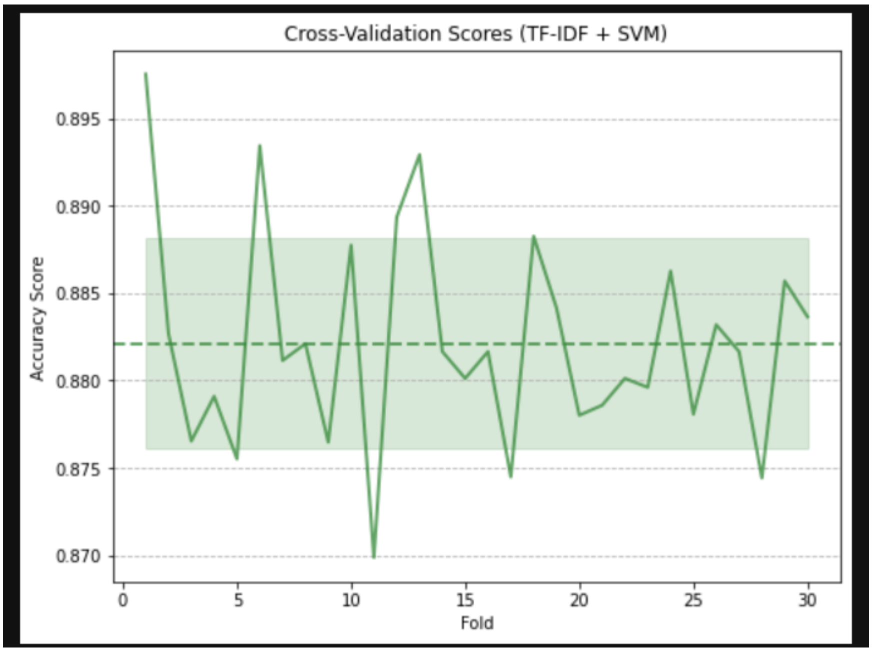


Fig. 4.12 Cross-Validation Scores (TF-IDF + SVM))

[14] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RepeatedStratifiedKFold.html

[15] https://machinelearningmastery.com/repeated-k-fold-cross-validation-with-python/

[16] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html

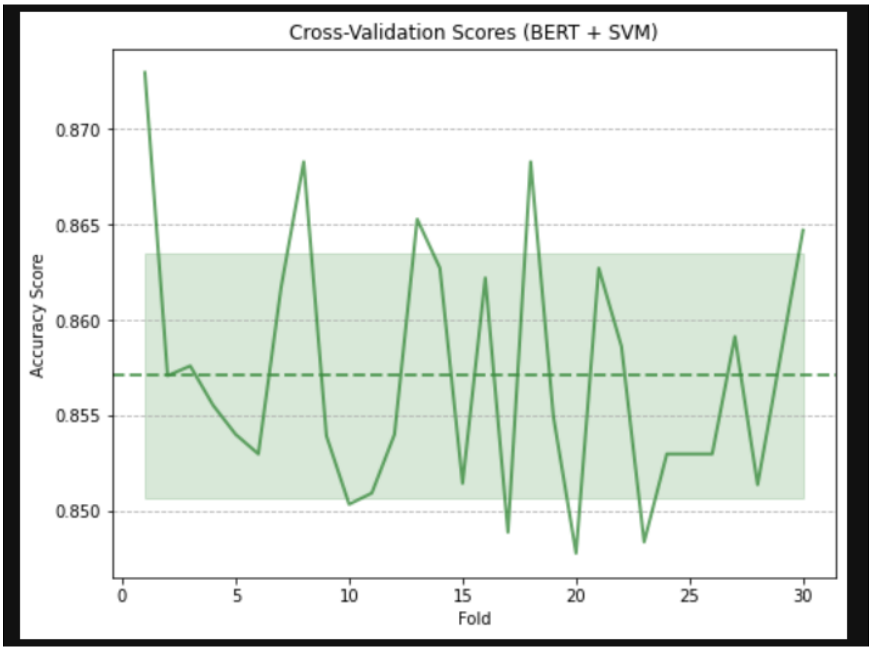


Fig. 4.13 Cross-Validation Scores (BERT + SVM)

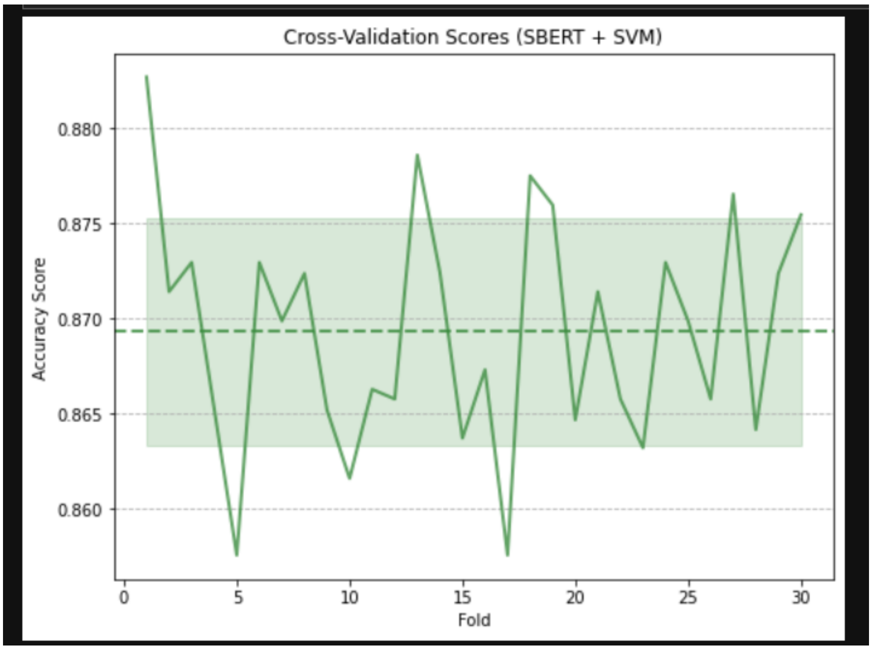


Fig. 4.14 Cross-Validation Scores (SBERT + SVM)

According to the results presented in Table 4.3 and Figure 4.15, the SVM model based on TF-IDF appears to have the highest mean accuracy when using 10-fold cross validation.

Text transformation + Classifier	**Mean Accuracy %**
TF-IDF + SVM	88.21 %
BERT + SVM	85.71 %
SBERT + SVM	86.93 %

Table 4.3 Mean Accuracy of Three Models
(10-fold Cross-Validation)

The SBERT-based SVM model, which uses a transformer-based language model called SBERT as features and a Support Vector Machine (SVM) classifier, is the second most accurate model among the three models evaluated in this study. The BERT-based SVM model, which uses a similar transformer-based language model called BERT and an SVM classifier, the accuracy was the lowest among the three models. Overall, the model that uses the term frequency-inverse document frequency (TF-IDF) feature representation and a classifier appears to perform the best among the three models.

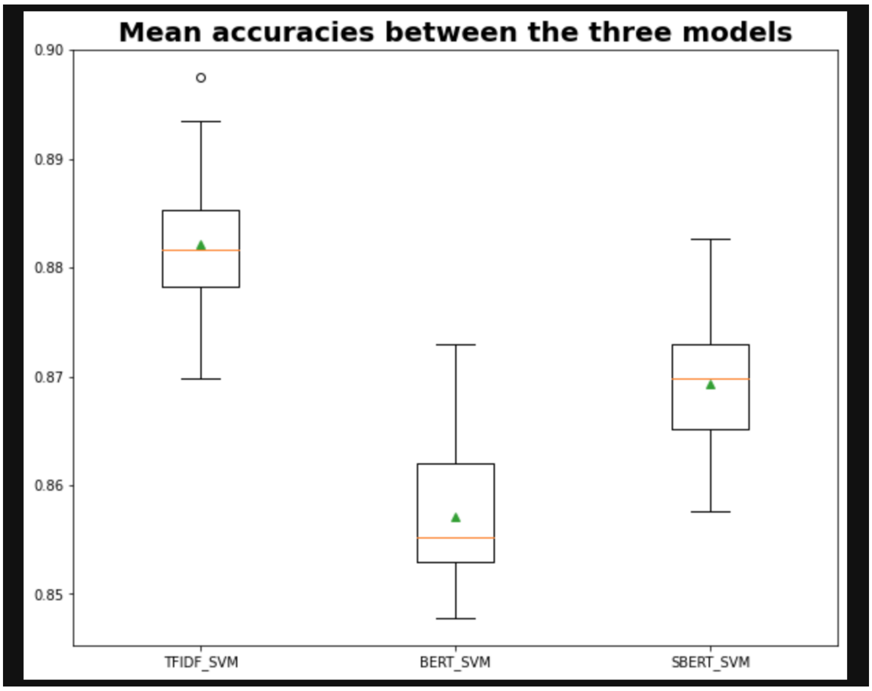


Fig. 4.15 Mean Accuracy of Three Models
(10-fold Cross-Validation)

4.3.4 Normality analysis

The purpose of normality analysis is to determine whether a given dataset follows a normal distribution or not. This is important because it determines which statistical test can be used to accept or reject hypotheses. In this study, it was determined that if the data is not normally distributed, the Mann-Whitney U test should be used. On the other hand, f the data demonstrates a normal distribution, the student t-test should be utilized. Normality analysis is crucial in statistical analysis because it helps to ensure that the chosen statistical test is appropriate for the data being analyzed, which can affect the reliability and validity of the results.

To determine whether it is appropriate to use parametric or non-parametric difference tests on a steam dataset, it is necessary to determine whether the data is normally distributed. The Shapiro-Wilk test can be used to evaluate the normality of data samples. This test is commonly used to assess whether the data follows a normal distribution. The Shapiro-Wilk test was conducted on models, and the results are presented in Table 4.4. It is important to determine the normality of the data in order to choose the appropriate type of difference test to use. Parametric tests require the data to follow a normal distribution, while non-parametric tests do not have this requirement.

Model	Static	p-value	Normality
TF-IDF + SVM	0.97	0.57	Normal
BERT + SVM	0.94	0.12	Normal
SBERT + SVM	0.97	0.68	Normal

Table 4.4 Shapiro-Wilk Test Result

For the accuracy scores of the Steam dataset in three models, the Shapiro-Wilk test was applied. The null hypothesis of this test is that the data follows a normal distribution, while the alternative hypothesis is that the data does not follow a normal distribution. If the p-value obtained from the test is lower than 0.05, it indicates that the data does not follow a normal distribution. However, in the case of the Steam dataset, all three models had a p-value higher than 0.05, which means that there is sufficient statistical evidence to support the null hypothesis and conclude that the data follows a normal distribution. Therefore, parametric tests, including the Student's t-test, can be used for statistical analysis and does not need to follow the Mann-Whitney U tests.

4.3.5 Statistical Analysis

The Student's t-test was selected as the statistical analysis method for this study because the data in all of the models conforms to a normal distribution. This meant that the t-test, which is designed for normally distributed data, was an appropriate choice. In contrast, the Mann-Whitney U test is used when the data is not normally distributed. Therefore, the decision was made to use the Student's t-test in this case because the data was found to be normally distributed and did not need to be analyzed using the Mann-Whitney U test.

The Student's t-test is a statistical procedure that is used to determine whether there is a significant difference between the means of two groups or samples. In the context of machine learning, it can be used to compare the performance of different models or techniques. In this study, the t-test is used to compare the performance of two models that use BERT and SBERT text transformers with SVM classifiers, to a baseline model that uses a TF-IDF base with an SVM classifier. To perform the t-test, it is assumed that the data is normally distributed. This assumption is important because it allows for certain statistical calculations to be made, such as the calculation of the mean and standard deviation. The t-test involves calculating the means and standard deviations of the two samples and using this information to calculate a t-statistic. The t-statistic is then used to determine a p-value, which is a measure of the probability that the difference between the means is due to random chance. If the p-value is below a predetermined level of significance (usually 0.05), it can be concluded that there is a statistically significant difference between the means of the two samples.

Comparing Model	T-Static	p-value
(BERT + SVM) with (TF-IDF + SVM)	15.36	0.00
(SBERT + SVM) with (TF-IDF + SVM)	8.12	0.00

Table 4.5 Student t-test Result

The data in Table 4.5 suggests that the mean accuracy of the baseline TF-IDF based SVM model is significantly higher than the mean accuracy of both the BERT and SBERT based SVM models. This is indicated by the p-values for the comparisons between these models, which are both less than 0.05, and the positive t-statistics for both comparisons. These results suggest that the baseline model performs better in terms of mean accuracy compared to the models that incorporate BERT and SBERT.

4.4 Hypothesis Acceptance and Rejection

The results of the 10-fold mean cross-validation, as presented in Table 4.3, demonstrate that the baseline model using term frequency-inverse document frequency (TF-IDF) and support vector machine (SVM) had higher accuracy compared to the SVM models based on BERT (Bidirectional Encoder Representations from Transformers) and SBERT (Sentiment-aware BERT) for the data in this study. In order to determine the statistical significance of this difference, Shapiro-Wilk tests were conducted to assess the normality of the data. Upon finding that the data was normal, a student's t-test was conducted to formally accept or reject the hypothesis. Both Table 4.3 and Table 4.5 present statistical evidence that leads to the acceptance of the null hypothesis and the rejection of the alternate hypothesis and providing evidence of that the SVM classifiers built on BERT and SBERT as pre-trained text transformer techniques did not significantly improve the accuracy of sentiment prediction in reviews compared to an SVM classifier model built on the TF-IDF text transformer technique.

4.5 Discussion

In this study, a TF-IDF based SVM Classifier model was trained on the Steam dataset for sentiment analysis classification of users' reviews on the Steam game platform. The model achieved an accuracy of 88.21%, which was the highest among the three models tested, including two pre-trained text transformers (BERT and SBERT). It is possible that the lower effectiveness of the BERT and SBERT models in this case was due to their design for use with Deep Learning Neural Network models rather than traditional Machine Learning models such as SVM.

The results of this research showed that the TF-IDF based SVM model was more effective at predicting the sentiment of users' reviews on the Steam platform compared to the BERT and SBERT based SVM models. However, it would be worthwhile to investigate in future studies whether the accuracy of BERT and SBERT based Machine Learning models could be improved by using them with Deep Learning models, as this would provide a more comprehensive comparison of their effectiveness. The objective of this study was to leverage the ability of BERT and SBERT, as pre-trained text transformers, to capture contextual meaning and accurately classify text articles according to sentiment.

Chapter 5

Conclusion

5.1 Research Overview

In recent years, there has been a growing demand for natural language processing approaches and text analytics to analyze online reviews. The proliferation of digital devices and social media has made it increasingly important to identify and process user feedback. This has made research in the field of sentiment analysis especially crucial. In the case of the Steam gaming platform, analyzing the sentiment of user reviews can help the platform understand how players feel about specific games. This information can be used to improve the platform by recommending well-liked games and highlighting those with negative reviews. Additionally, sentiment analysis can assist developers in understanding the reception of their games and help them make improvements. A number of embedding techniques are employed in this study, like BERT as a word embedder and SBERT as a sentence embedder, along with traditional techniques like TF-IDF, to classify the sentiment of customer reviews.

5.2 Problem Definition

This research aimed to address the gaps in the literature identified during the literature review, specifically the lack of exploration of word and sentence embedding techniques for sentiment analysis of user reviews on the Steam gaming platform. Traditional techniques such as TF-IDF are commonly used for this task, but do not capture the meaning of words based on their context. In contrast, word embedding techniques such as BERT and sentence embedding techniques like SBERT are designed to capture the meaning of words based on the words surrounding them.

To evaluate the effectiveness of these techniques, experiments were conducted using a Support Vector Machine (SVM) classifier model with BERT and SBERT as the text representation method, and comparing the results to a Support Vector Machine (SVM) classifier model using TF-IDF. The goal of these experiments was to determine if the use of embedding-based techniques would result in higher accuracy compared to the traditional TF-IDF method.

5.3 Design, Experimentation, Evaluation & Results

The CRISP-DM methodology, a well-established framework for data mining projects, was used in this study to evaluate the performance of various text representation techniques for sentiment analysis on the Steam platform. To gather the data for the study, the researchers obtained the Steam dataset from the Kaggle website and conducted a thorough analysis to understand its characteristics and features. The findings from this analysis were then used to prepare the dataset for modeling, which involved selecting and transforming the data in a format that could be used by machine learning algorithms. Overall, the CRISP-DM methodology provided a structured approach for guiding the analysis and modeling of the Steam dataset for sentiment analysis.

The main focus of the research was to compare the In order to evaluate the performance of different techniques of text representation for sentiment analysis on the Steam platform, different techniques of text representation were compared during the research. A baseline model based on the TF-IDF technique was selected from existing research (Alzami et al., 2020), and this model was compared against SVM models using BERT and SBERT for word and sentence embedding. These techniques were chosen based on the research gap identified during the literature review. According to the results of the experiments, the most effective technique for representing text for sentiment analysis on Steam platform has been determined based on the results of the experiments.

After all methods were applied, classification reports were generated based on the steam dataset. To validate the results, a 10-fold cross-validation method was used. Then Shapiro-Wilk normality tests were conducted to determine whether the 10-fold cross-validation scores were distributed normally. In order to determine which statistical difference test would be most appropriate to compare the results of the different models, this test was performed. If the distribution of the scores was found to be normal, a Student's t-test was performed. However, as the normality test revealed that the scores for all three models were normal, a Mann-Whitney U test was not used instead. These tests were used to determine whether there was a statistically significant difference in accuracy between the models, thereby determining whether the null hypothesis was accepted or rejected.

Finally, according to the results of this study, the traditional TF-IDF text representation based model of SVM classifiers performed significantly better than that of the BERT and SBERT based SVM classifiers, which are based on embeddings of words and sentences respectively. The null hypothesis was rejected, and the alternate hypothesis was accepted. These results suggest that the traditional TF-IDF technique may be more effective for sentiment analysis on the Steam platform than modern embedding-based methods. Further research may be needed to confirm these findings and explore other potential applications of these techniques.

5.4 Contributions & Impact

As part of this study, an evaluation of the performance of the classifier that incorporates BERT as word embeddings and SBERT as sentence embeddings was conducted compared to the performance of the TF-IDF-based support vector machine classifier. The purpose of using BERT and SBERT as embeddings is to capture the contextual information and relationships between words and sentences, respectively, in the text data. It is hypothesized that by leveraging this contextual information, the SVM model with BERT and SBERT embeddings will be able to achieve higher accuracy in various Processes related to natural language processing, including sentiment analysis and text classification, compared to the TF-IDF model, which only considers the frequency of words that exist in the text.

Although the support vector machine (SVM) model that utilized BERT as word embeddings and SBERT as sentence embeddings did not outperform the term frequency-inverse document frequency (TF-IDF) model with 89% accuracy, both the BERT-based and SBERT-based SVM models still demonstrated reliable accuracy with scores of 86% and 87% on the steam Kaggle dataset, respectively. These results indicate that although the use of BERT and SBERT embeddings in a support vector machine (SVM) model may not have significantly enhanced the overall performance compared to a term frequency-inverse document frequency (TF-IDF) model, they were still able to offer valuable insights and information about the contextual relationships within the text data.

The BERT and SBERT embeddings capture the contextual information and relationships between words and sentences, respectively, in the text data, and this information can be useful for tasks such as text classification and sentiment analysis. While the improvement in performance may not have been significant in this particular study, the BERT and SBERT embeddings could potentially be more beneficial in other Natural Language Processing(NLP) tasks or when used in combination with other techniques. Additionally, it is worth noting that the use of these embeddings may still be beneficial for certain specific applications or scenarios, even if they do not significantly improve the overall performance of the model.

5.5 Future Work & Recommendations

Term frequency-inverse document frequency (TF-IDF) has been shown to be effective when used with machine learning algorithms in previous research. However, the use of transformer-based language models such as BERT and SBERT as word and sentence embedding techniques in conjunction with deep learning algorithms may potentially yield improved results in the sentiment analysis of user reviews. Deep learning algorithms are known to be particularly effective at processing text representation techniques based on embeddings, and may be able to achieve higher accuracy compared to the current method of using TF-IDF and Support Vector Machines (SVMs). Therefore, it may be worth considering further research into this area in the future to determine the potential benefits of using transformer-based embeddings and deep learning algorithms for sentiment analysis tasks.

It would be worthwhile for future research to explore the use of fake review detection techniques to identify and collect both fake and genuine user reviews. Once these reviews have been identified and separated, sentiment analysis could be performed on both fake and real reviews. This approach would provide a more comprehensive understanding of the sentiment present in user reviews, as it would allow for the separate analysis of fake and real reviews. Additionally, this approach may result in more accurate fake review detection, as it may enable the identification of patterns or characteristics that distinguish fake reviews from real reviews. By analyzing both fake and real reviews, it may be possible to identify features or indicators that are specific to fake reviews and use this information to inform future fake review detection efforts.

A potential direction for future research could be to conduct a comprehensive and reliable sentiment analysis on fake and genuine user reviews using clustering algorithms. Clustering algorithms are a type of unsupervised machine learning technique that could used to group data points into clusters based on their similarity. By applying clustering algorithms to fake and genuine user reviews, it may be possible to identify patterns or trends in the sentiment of the reviews. This approach may provide more accurate results and a deeper understanding of the sentiment present in the reviews, compared to the current methods used. As such, exploring the use of clustering algorithms in this context may yield valuable insights and could be a promising area of study.

As a recommendation for future research, it may be useful to consider comparing the performance of various machine learning models for the task of sentiment analysis of user reviews. By doing so, the best performing model can be identified and chosen based on the chosen metrics. This approach could help to improve the accuracy and effectiveness of sentiment analysis, as well as potentially leading to the discovery of new and innovative methods for addressing this task. Additionally, it may be valuable to consider incorporating additional data sources or techniques, such as natural language processing or data visualization, to enhance the performance of the selected model. Overall, ongoing exploration and experimentation with different approaches to sentiment analysis can help to advance the field and provide more effective solutions for understanding and analyzing the sentiments of users.

References

Alantari, H. J., Currim, I. S., Deng, Y., & Singh, S. (2022). An empirical comparison of machine learning methods for text-based sentiment analysis of online consumer reviews. *International Journal of Research in Marketing*, *39*(1), 1–19. https://doi.org/10.1016/j.ijresmar.2021.10.011

Alzami, F., Udayanti, E. D., Prabowo, D. P., & Megantara, R. A. (2020). Document preprocessing with TF-IDF to improve the polarity classification performance of unstructured sentiment analysis. *Kinetik: Game Technology, Information System, Computer Network, Computing, Electronics, and Control*, 235–242. https://doi.org/10.22219/kinetik.v5i3.1066

Arief, M., & Deris, M. B. M. (2021). Text preprocessing impact for sentiment classification in product review. *2021 Sixth International Conference on Informatics and Computing (ICIC)*, 1–7. https://doi.org/10.1109/ICIC54025.2021.9632884

Balakrishnan, V., Selvanayagam, P. K., & Yin, L. P. (2020). Sentiment and emotion analyses for malaysian mobile digital payment applications. *Proceedings of the 2020 the 4th International Conference on Compute and Data Analysis*, 67–71. https://doi.org/10.1145/3388142.3388144

Beseiso, M., & Alzahrani, S. (2020). An empirical analysis of BERT embedding for automated essay scoring. *International Journal of Advanced Computer Science and Applications*, *11*(10). https://doi.org/10.14569/IJACSA.2020.0111027

Cahyanti, F. E., Adiwijaya, & Faraby, S. A. (2020). On the feature extraction for sentiment analysis of movie reviews based on SVM. *2020 8th International Conference on Information and Communication Technology (ICoICT)*, 1–5. https://doi.org/10.1109/ICoICT49345.2020.9166397

Chouikhi, H., Chniter, H., & Jarray, F. (2020). On the feature extraction for sentiment analysis of movie reviews based on SVM. *2020 8th International Conference on Information and Communication Technology (ICoICT)*.

Dang, N. C., Moreno-García, M. N., & De la Prieta, F. (2020). Sentiment analysis based on deep learning: A comparative study. *Electronics*, *9*(3), 483. https://doi.org/10.3390/electronics9030483

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. *Proceedings of the 2019 Conference of the North*, 4171–4186. https://doi.org/10.18653/v1/N19-1423

Dong, J., He, F., Guo, Y., & Zhang, H. (2020). A commodity review sentiment analysis based on BERT-CNN model. *2020 5th International Conference on Computer and Communication Systems (ICCCS)*, 143–147. https://doi.org/10.1109/ICCCS49078.2020.9118434

Fang, X., & Zhan, J. (2015). Sentiment analysis using product review data. *Journal of Big Data*, *2*(1), 5. https://doi.org/10.1186/s40537-015-0015-2

Gallagher, C., Furey, E., & Curran, K. (2019). The application of sentiment analysis and text analytics to customer experience reviews to understand what customers are really saying: *International Journal of Data Warehousing and Mining*, *15*(4), 21–47. https://doi.org/10.4018/IJDWM.2019100102

Hanusz, Z., Tarasinska, J., & Zielinski, W. (2016). Shapiro–wilk test with known mean [Artwork Size: 89–100 Pages Publisher: REVSTAT-Statistical Journal]. *REVSTAT-Statistical Journal*, 89–100 Pages. https://doi.org/10.57805/REVSTAT.V14I1.180

Huang, B., Zhang, J., Ju, J., Guo, R., Fujita, H., & Liu, J. (2023). CRF-GCN: An effective syntactic dependency model for aspect-level sentiment analysis. *Knowledge-Based Systems*, *260*, 110125. https://doi.org/10.1016/j.knosys.2022.110125

Iqbal, A., Amin, R., Iqbal, J., Alroobaea, R., Binmahfoudh, A., & Hussain, M. (2022). Sentiment analysis of consumer reviews using deep learning. *Sustainability*, *14*(17), 10844. https://doi.org/10.3390/su141710844

Jeffrey, R., Bian, P., Ji, F., & Sweetser, P. (2020). The wisdom of the gaming crowd. *Extended Abstracts of the 2020 Annual Symposium on Computer-Human Interaction in Play*, 272–276. https://doi.org/10.1145/3383668.3419915

Kasuya, E. (2001). Mann–whitney u test when variances are unequal. *Animal Behaviour*, *61*(6), 1247–1249. https://doi.org/10.1006/anbe.2001.1691

Lu, K., & Wu, J. (2019). Sentiment analysis of film review texts based on sentiment dictionary and SVM. *Proceedings of the 2019 3rd International Conference on Innovation in Artificial Intelligence - ICIAI 2019*, 73–77. https://doi.org/10.1145/3319921.3319966

Mishra, P., Singh, U., Pandey, C., Mishra, P., & Pandey, G. (2019). Application of student's t-test, analysis of variance, and covariance. *Annals of Cardiac Anaesthesia*, *22*(4), 407. https://doi.org/10.4103/aca.ACA_94_19

Mohamed Ali, N., El Hamid, M. M. A., & Youssif, A. (2019). SENTIMENT ANALYSIS FOR MOVIES REVIEWS DATASET USING DEEP LEARNING MODELS. *International Journal of Data Mining & Knowledge Management Process*, *09*(3), 19–27. https://doi.org/10.5121/ijdkp.2019.9302

Nabiha, A., Mutalib, S., & Malik, A. M. A. (2021). Sentiment analysis for informal malay text in social commerce. *2021 2nd International Conference on Artificial Intelligence and Data Sciences (AiDAS)*, 1–6. https://doi.org/10.1109/AiDAS53897.2021.9574436

Normah, N. (2019). Naïve bayes algorithm for sentiment analysis windows phone store application reviews. *SinkrOn*, *3*(2), 13. https://doi.org/10.33395/sinkron.v3i2.242

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., & Zettlemoyer, L. (2018, March 22). Deep contextualized word representations. Retrieved December 7, 2022, from http://arxiv.org/abs/1802.05365

Reimers, N., & Gurevych, I. (2019, August 27). Sentence-BERT: Sentence embeddings using siamese BERT-networks. Retrieved December 26, 2022, from http://arxiv.org/abs/1908.10084

Saifullah, S., Fauziah, Y., & Aribowo, A. S. (2021). Comparison of machine learning for sentiment analysis in detecting anxiety based on social media data [Publisher: arXiv Version Number: 1]. https://doi.org/10.48550/ARXIV.2101.06353

Sobkowicz, A., & Stokowiec, W. (2016, May 23). *Steam review dataset - new, large scale sentiment dataset*.

Srivastava, R., Bharti, P., & Verma, P. (2021). Sentiment analysis using feature generation and machine learning approach. *2021 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS)*, 86–91. https://doi.org/10.1109/ICCCIS51004.2021.9397135

Tripathi, G., & S, N. (2015). Feature selection and classification approach for sentiment analysis. *Machine Learning and Applications: An International Journal*, *2*(2), 01–16. https://doi.org/10.5121/mlaij.2015.2201

Utz, S., Kerkhof, P., & van den Bos, J. (2012). Consumers rule: How consumer reviews influence perceived trustworthiness of online stores. *Electronic Commerce Research and Applications*, *11*(1), 49–58. https://doi.org/10.1016/j.elerap.2011.07.010

Vieira, A., & Brandão, W. (2019). Evaluating acceptance of video games using convolutional neural networks for sentiment analysis of user reviews. *Proceedings of the 30th ACM Conference on Hypertext and Social Media*, 273–274. https://doi.org/10.1145/3342220.3344924

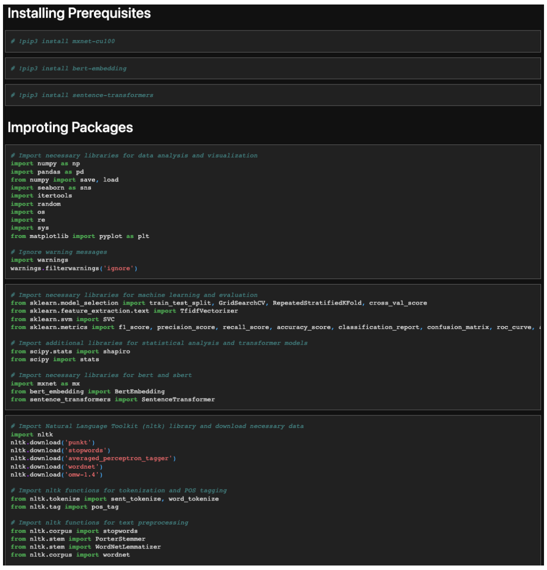
Zuo, Z. (2018). Sentiment analysis of steam review datasets using naive bayes and decision tree classifier, 7.

Appendix A

Research Coding Part After Collection of Target Data

The code for this study can be found on the Github under the repository "Msc_Diss_Cod [1].

The code for appendix A can be found on the Github under the repository "Msc_Diss_Cod" with the filename "D02124995_Mina_Jmashidian_Disseratation_Main_Coding.ipynb [2].



Installing Prerequisites

```
# !pip3 install mxnet-cu100
```

```
# !pip3 install bert-embedding
```

```
# !pip3 install sentence-transformers
```

Improting Packages

```
# Import necessary libraries for data analysis and visualization
import numpy as np
import pandas as pd
from numpy import save, load
import seaborn as sns
import itertools
import random
import os
import re
import sys
from matplotlib import pyplot as plt

# Ignore warning messages
import warnings
warnings.filterwarnings('ignore')
```

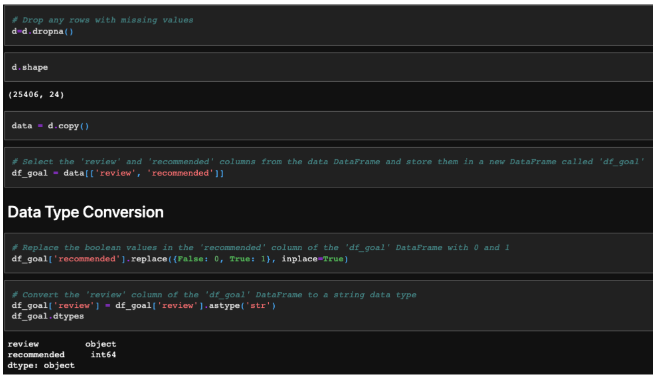
```
# Import necessary libraries for machine learning and evaluation
from sklearn.model_selection import train_test_split, GridSearchCV, RepeatedStratifiedKFold, cross_val_score
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import SVC
from sklearn.metrics import f1_score, precision_score, recall_score, accuracy_score, classification_report, confusion_matrix, roc_curve,

# Import additional libraries for statistical analysis and transformer models
from scipy.stats import shapiro
from scipy import stats

# Import necessary libraries for bert and sbert
import mxnet as mx
from bert_embedding import BertEmbedding
from sentence_transformers import SentenceTransformer
```

```
# Import Natural Language Toolkit (nltk) library and download necessary data
import nltk
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('averaged_perceptron_tagger')
nltk.download('wordnet')
nltk.download('omw-1.4')

# Import nltk functions for tokenization and POS tagging
from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.tag import pos_tag

# Import nltk functions for text preprocessing
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet
```

Fig. A.1 Installing and Importing package

[1] https://github.com/minajm/Msc_Diss_Cod

[2] https://github.com/minajm/Msc_Diss_Cod/blob/main/D02124995_Mina_Jmashidian_Disseratation_Main_Coding.ipynb



```
# Drop any rows with missing values
d=d.dropna()
```

```
d.shape
```

```
(25406, 24)
```

```
data = d.copy()
```

```
# Select the 'review' and 'recommended' columns from the data DataFrame and store them in a new DataFrame called 'df_goal'
df_goal = data[['review', 'recommended']]
```

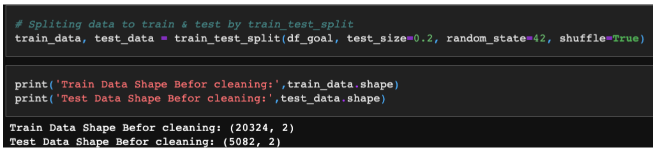
Data Type Conversion

```
# Replace the boolean values in the 'recommended' column of the 'df_goal' DataFrame with 0 and 1
df_goal['recommended'].replace({False: 0, True: 1}, inplace=True)
```

```
# Convert the 'review' column of the 'df_goal' DataFrame to a string data type
df_goal['review'] = df_goal['review'].astype('str')
df_goal.dtypes
```

```
review          object
recommended      int64
dtype: object
```

Fig. A.2 Data Cleaning 1

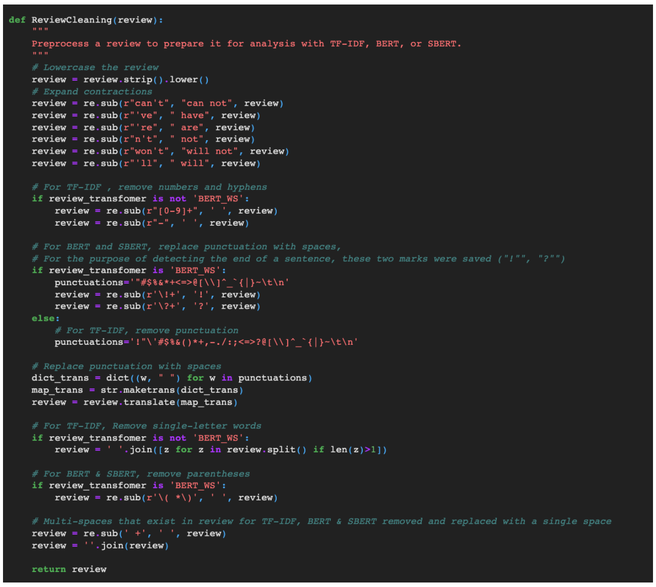


```
# Spliting data to train & test by train_test_split
train_data, test_data = train_test_split(df_goal, test_size=0.2, random_state=42, shuffle=True)
```

```
print('Train Data Shape Befor cleaning:',train_data.shape)
print('Test Data Shape Befor cleaning:',test_data.shape)
```

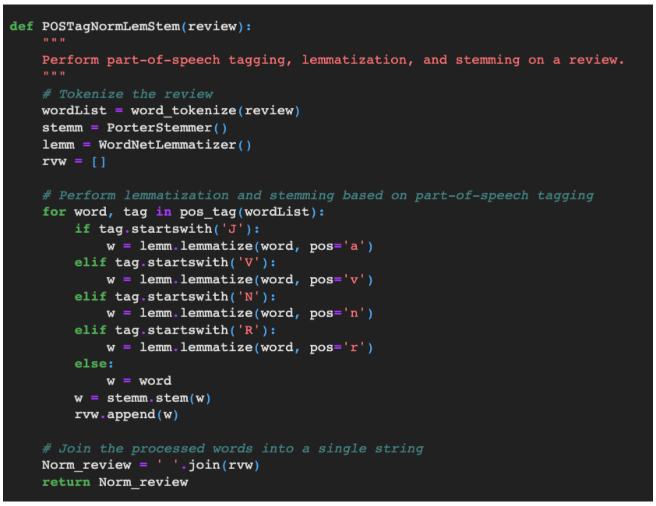
```
Train Data Shape Befor cleaning: (20324, 2)
Test Data Shape Befor cleaning: (5082, 2)
```

Fig. A.3 Data Splitting to Train and Test Data



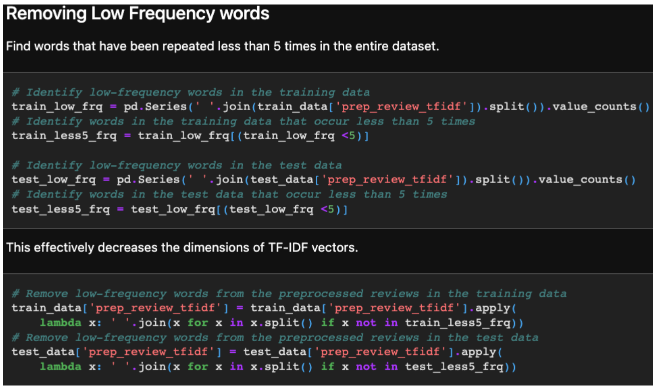
```
def ReviewCleaning(review):
    """
    Preprocess a review to prepare it for analysis with TF-IDF, BERT, or SBERT.
    """
    # Lowercase the review
    review = review.strip().lower()
    # Expand contractions
    review = re.sub(r"can't", "can not", review)
    review = re.sub(r"'ve", " have", review)
    review = re.sub(r"'re", " are", review)
    review = re.sub(r"n't", " not", review)
    review = re.sub(r"won't", "will not", review)
    review = re.sub(r"'ll", " will", review)

    # For TF-IDF , remove numbers and hyphens
    if review_transfomer is not 'BERT_WS':
        review = re.sub(r"[0-9]+", ' ', review)
        review = re.sub(r"-", ' ', review)

    # For BERT and SBERT, replace punctuation with spaces,
    # For the purpose of detecting the end of a sentence, these two marks were saved ("!"", "?"")
    if review_transfomer is 'BERT_WS':
        punctuations='"#$%&*+<=>@[\\]^_`{|}~\t\n'
        review = re.sub(r'\!+', '!', review)
        review = re.sub(r'\?+', '?', review)
    else:
        # For TF-IDF, remove punctuation
        punctuations='!"\'#$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n'

    # Replace punctuation with spaces
    dict_trans = dict((w, " ") for w in punctuations)
    map_trans = str.maketrans(dict_trans)
    review = review.translate(map_trans)

    # For TF-IDF, Remove single-letter words
    if review_transfomer is not 'BERT_WS':
        review = ' '.join([z for z in review.split() if len(z)>1])

    # For BERT & SBERT, remove parentheses
    if review_transfomer is 'BERT_WS':
        review = re.sub(r'\( *\)', ' ', review)

    # Multi-spaces that exist in review for TF-IDF, BERT & SBERT removed and replaced with a single space
    review = re.sub(' +', ' ', review)
    review = ''.join(review)

    return review
```

Fig. A.4 Text Preprocessing 1



```
def POSTagNormLemStem(review):
    """
    Perform part-of-speech tagging, lemmatization, and stemming on a review.
    """
    # Tokenize the review
    wordList = word_tokenize(review)
    stemm = PorterStemmer()
    lemm = WordNetLemmatizer()
    rvw = []

    # Perform lemmatization and stemming based on part-of-speech tagging
    for word, tag in pos_tag(wordList):
        if tag.startswith('J'):
            w = lemm.lemmatize(word, pos='a')
        elif tag.startswith('V'):
            w = lemm.lemmatize(word, pos='v')
        elif tag.startswith('N'):
            w = lemm.lemmatize(word, pos='n')
        elif tag.startswith('R'):
            w = lemm.lemmatize(word, pos='r')
        else:
            w = word
        w = stemm.stem(w)
        rvw.append(w)

    # Join the processed words into a single string
    Norm_review = ' '.join(rvw)
    return Norm_review
```

Fig. A.5 Text Preprocessing 2



Removing Low Frequency words

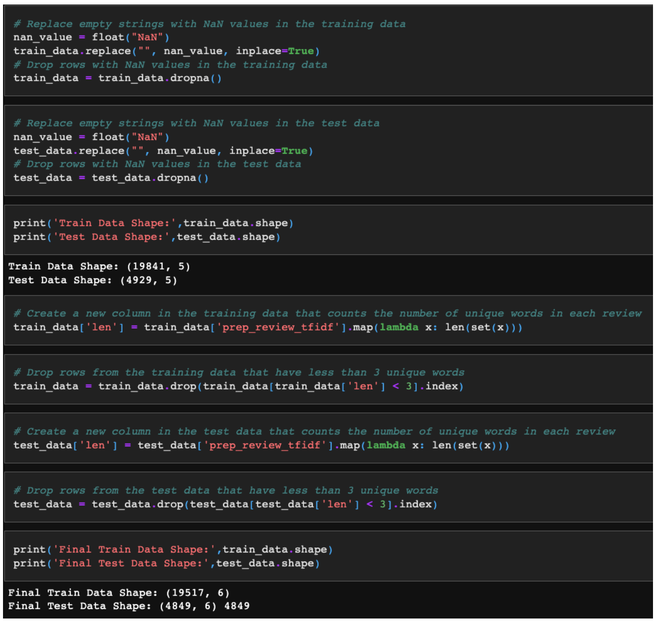
Find words that have been repeated less than 5 times in the entire dataset.

```
# Identify low-frequency words in the training data
train_low_frq = pd.Series(' '.join(train_data['prep_review_tfidf']).split()).value_counts()
# Identify words in the training data that occur less than 5 times
train_less5_frq = train_low_frq[(train_low_frq <5)]

# Identify low-frequency words in the test data
test_low_frq = pd.Series(' '.join(test_data['prep_review_tfidf']).split()).value_counts()
# Identify words in the test data that occur less than 5 times
test_less5_frq = test_low_frq[(test_low_frq <5)]
```

This effectively decreases the dimensions of TF-IDF vectors.

```
# Remove low-frequency words from the preprocessed reviews in the training data
train_data['prep_review_tfidf'] = train_data['prep_review_tfidf'].apply(
    lambda x: ' '.join(x for x in x.split() if x not in train_less5_frq))
# Remove low-frequency words from the preprocessed reviews in the test data
test_data['prep_review_tfidf'] = test_data['prep_review_tfidf'].apply(
    lambda x: ' '.join(x for x in x.split() if x not in test_less5_frq))
```

Fig. A.6 Text Preprocessing 3



```
# Replace empty strings with NaN values in the training data
nan_value = float("NaN")
train_data.replace("", nan_value, inplace=True)
# Drop rows with NaN values in the training data
train_data = train_data.dropna()
```

```
# Replace empty strings with NaN values in the test data
nan_value = float("NaN")
test_data.replace("", nan_value, inplace=True)
# Drop rows with NaN values in the test data
test_data = test_data.dropna()
```

```
print('Train Data Shape:',train_data.shape)
print('Test Data Shape:',test_data.shape)
```

```
Train Data Shape: (19841, 5)
Test Data Shape: (4929, 5)
```

```
# Create a new column in the training data that counts the number of unique words in each review
train_data['len'] = train_data['prep_review_tfidf'].map(lambda x: len(set(x)))
```

```
# Drop rows from the training data that have less than 3 unique words
train_data = train_data.drop(train_data[train_data['len'] < 3].index)
```

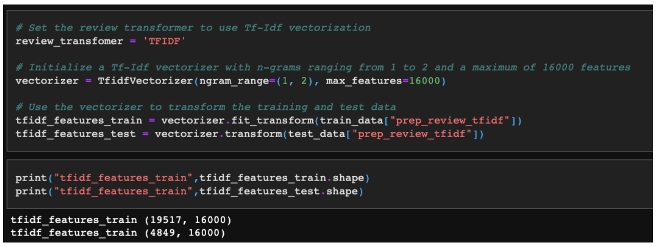
```
# Create a new column in the test data that counts the number of unique words in each review
test_data['len'] = test_data['prep_review_tfidf'].map(lambda x: len(set(x)))
```

```
# Drop rows from the test data that have less than 3 unique words
test_data = test_data.drop(test_data[test_data['len'] < 3].index)
```

```
print('Final Train Data Shape:',train_data.shape)
print('Final Test Data Shape:',test_data.shape)
```

```
Final Train Data Shape: (19517, 6)
Final Test Data Shape: (4849, 6) 4849
```

Fig. A.7 Data Cleaning 2

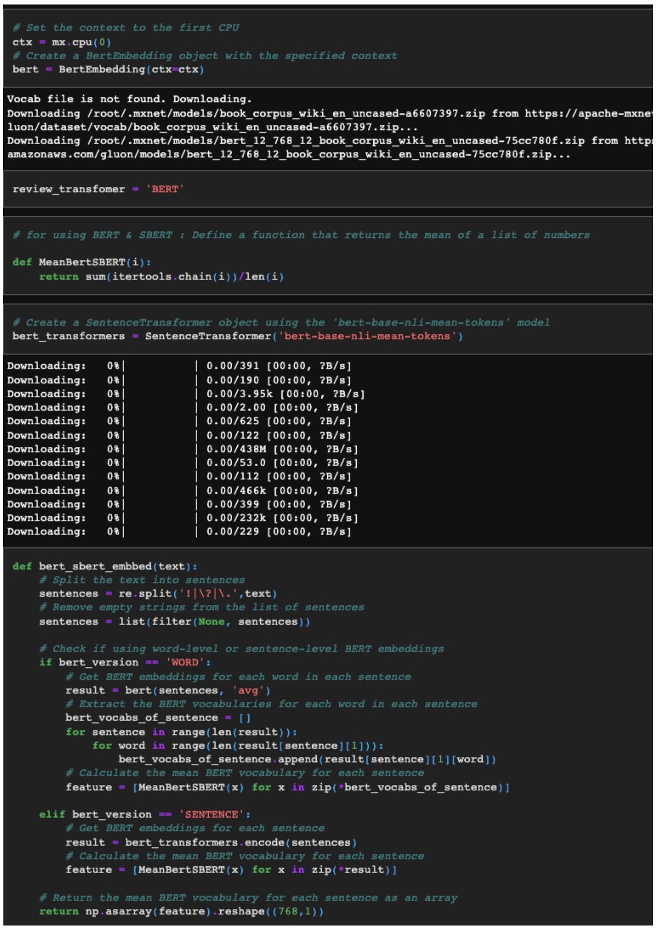


```
# Set the review transformer to use Tf-Idf vectorization
review_transfomer = 'TFIDF'

# Initialize a Tf-Idf vectorizer with n-grams ranging from 1 to 2 and a maximum of 16000 features
vectorizer = TfidfVectorizer(ngram_range=(1, 2), max_features=16000)

# Use the vectorizer to transform the training and test data
tfidf_features_train = vectorizer.fit_transform(train_data["prep_review_tfidf"])
tfidf_features_test = vectorizer.transform(test_data["prep_review_tfidf"])
```

```
print("tfidf_features_train",tfidf_features_train.shape)
print("tfidf_features_train",tfidf_features_test.shape)
```

```
tfidf_features_train (19517, 16000)
tfidf_features_train (4849, 16000)
```

Fig. A.8 Text Transformer: TF-IDF



```
# Set the context to the first CPU
ctx = mx.cpu(0)
# Create a BertEmbedding object with the specified context
bert = BertEmbedding(ctx=ctx)
```

```
Vocab file is not found. Downloading.
Downloading /root/.mxnet/models/book_corpus_wiki_en_uncased-a6607397.zip from https://apache-mxne
luon/dataset/vocab/book_corpus_wiki_en_uncased-a6607397.zip...
Downloading /root/.mxnet/models/bert_12_768_12_book_corpus_wiki_en_uncased-75cc780f.zip from http
amazonaws.com/gluon/models/bert_12_768_12_book_corpus_wiki_en_uncased-75cc780f.zip...
```

```
review_transfomer = 'BERT'
```

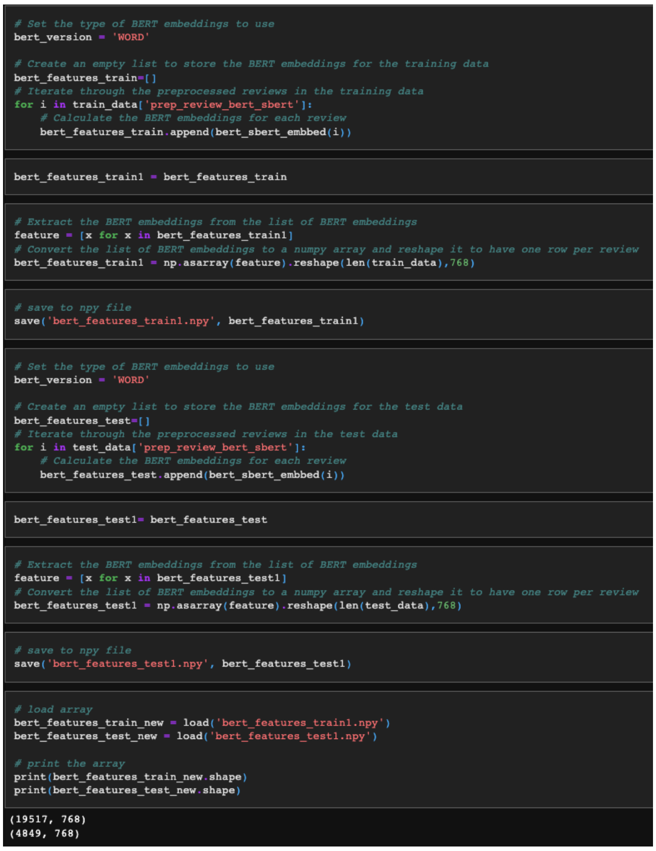
```
# for using BERT & SBERT : Define a function that returns the mean of a list of numbers

def MeanBertSBERT(i):
    return sum(itertools.chain(i))/len(i)
```

```
# Create a SentenceTransformer object using the 'bert-base-nli-mean-tokens' model
bert_transformers = SentenceTransformer('bert-base-nli-mean-tokens')
```

```
Downloading:   0%|          | 0.00/391 [00:00, ?B/s]
Downloading:   0%|          | 0.00/190 [00:00, ?B/s]
Downloading:   0%|          | 0.00/3.95k [00:00, ?B/s]
Downloading:   0%|          | 0.00/2.00 [00:00, ?B/s]
Downloading:   0%|          | 0.00/625 [00:00, ?B/s]
Downloading:   0%|          | 0.00/122 [00:00, ?B/s]
Downloading:   0%|          | 0.00/438M [00:00, ?B/s]
Downloading:   0%|          | 0.00/53.0 [00:00, ?B/s]
Downloading:   0%|          | 0.00/112 [00:00, ?B/s]
Downloading:   0%|          | 0.00/466k [00:00, ?B/s]
Downloading:   0%|          | 0.00/399 [00:00, ?B/s]
Downloading:   0%|          | 0.00/232k [00:00, ?B/s]
Downloading:   0%|          | 0.00/229 [00:00, ?B/s]
```

```
def bert_sbert_embbed(text):
    # Split the text into sentences
    sentences = re.split('!|\?|\.',text)
    # Remove empty strings from the list of sentences
    sentences = list(filter(None, sentences))

    # Check if using word-level or sentence-level BERT embeddings
    if bert_version == 'WORD':
        # Get BERT embeddings for each word in each sentence
        result = bert(sentences, 'avg')
        # Extract the BERT vocabularies for each word in each sentence
        bert_vocabs_of_sentence = []
        for sentence in range(len(result)):
            for word in range(len(result[sentence][1])):
                bert_vocabs_of_sentence.append(result[sentence][1][word])
        # Calculate the mean BERT vocabulary for each sentence
        feature = [MeanBertSBERT(x) for x in zip(*bert_vocabs_of_sentence)]

    elif bert_version == 'SENTENCE':
        # Get BERT embeddings for each sentence
        result = bert_transformers.encode(sentences)
        # Calculate the mean BERT vocabulary for each sentence
        feature = [MeanBertSBERT(x) for x in zip(*result)]

    # Return the mean BERT vocabulary for each sentence as an array
    return np.asarray(feature).reshape((768,1))
```

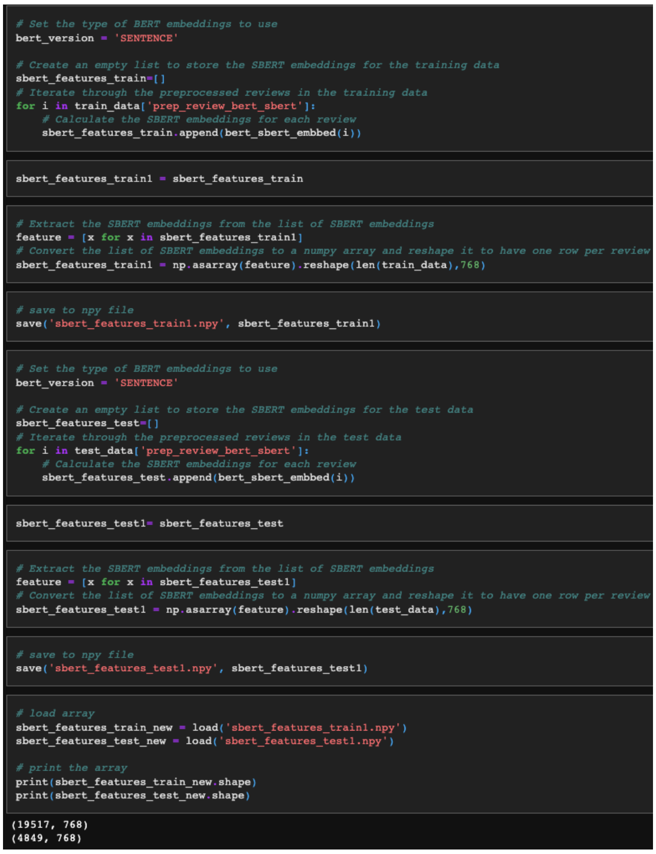
Fig. A.9 Text Transformer: Prepration for BERT and SBERT



```
# Set the type of BERT embeddings to use
bert_version = 'WORD'

# Create an empty list to store the BERT embeddings for the training data
bert_features_train=[]
# Iterate through the preprocessed reviews in the training data
for i in train_data['prep_review_bert_sbert']:
    # Calculate the BERT embeddings for each review
    bert_features_train.append(bert_sbert_embbed(i))

bert_features_train1 = bert_features_train

# Extract the BERT embeddings from the list of BERT embeddings
feature = [x for x in bert_features_train1]
# Convert the list of BERT embeddings to a numpy array and reshape it to have one row per review
bert_features_train1 = np.asarray(feature).reshape(len(train_data),768)

# save to npy file
save('bert_features_train1.npy', bert_features_train1)

# Set the type of BERT embeddings to use
bert_version = 'WORD'

# Create an empty list to store the BERT embeddings for the test data
bert_features_test=[]
# Iterate through the preprocessed reviews in the test data
for i in test_data['prep_review_bert_sbert']:
    # Calculate the BERT embeddings for each review
    bert_features_test.append(bert_sbert_embbed(i))

bert_features_test1= bert_features_test

# Extract the BERT embeddings from the list of BERT embeddings
feature = [x for x in bert_features_test1]
# Convert the list of BERT embeddings to a numpy array and reshape it to have one row per review
bert_features_test1 = np.asarray(feature).reshape(len(test_data),768)

# save to npy file
save('bert_features_test1.npy', bert_features_test1)

# load array
bert_features_train_new = load('bert_features_train1.npy')
bert_features_test_new = load('bert_features_test1.npy')

# print the array
print(bert_features_train_new.shape)
print(bert_features_test_new.shape)
```

```
(19517, 768)
(4849, 768)
```

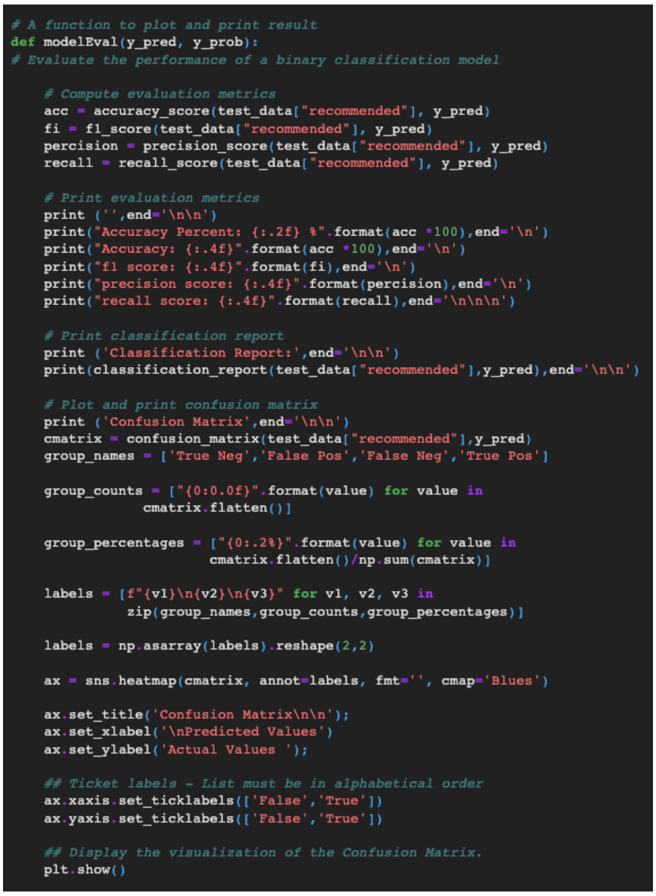
Fig. A.10 Text Transformer: BERT



```
# Set the type of BERT embeddings to use
bert_version = 'SENTENCE'

# Create an empty list to store the SBERT embeddings for the training data
sbert_features_train=[]
# Iterate through the preprocessed reviews in the training data
for i in train_data['prep_review_bert_sbert']:
    # Calculate the SBERT embeddings for each review
    sbert_features_train.append(bert_sbert_embbed(i))

sbert_features_train1 = sbert_features_train

# Extract the SBERT embeddings from the list of SBERT embeddings
feature = [x for x in sbert_features_train1]
# Convert the list of SBERT embeddings to a numpy array and reshape it to have one row per review
sbert_features_train1 = np.asarray(feature).reshape(len(train_data),768)

# save to npy file
save('sbert_features_train1.npy', sbert_features_train1)

# Set the type of BERT embeddings to use
bert_version = 'SENTENCE'

# Create an empty list to store the SBERT embeddings for the test data
sbert_features_test=[]
# Iterate through the preprocessed reviews in the test data
for i in test_data['prep_review_bert_sbert']:
    # Calculate the SBERT embeddings for each review
    sbert_features_test.append(bert_sbert_embbed(i))

sbert_features_test1= sbert_features_test

# Extract the SBERT embeddings from the list of SBERT embeddings
feature = [x for x in sbert_features_test1]
# Convert the list of SBERT embeddings to a numpy array and reshape it to have one row per review
sbert_features_test1 = np.asarray(feature).reshape(len(test_data),768)

# save to npy file
save('sbert_features_test1.npy', sbert_features_test1)

# load array
sbert_features_train_new = load('sbert_features_train1.npy')
sbert_features_test_new = load('sbert_features_test1.npy')

# print the array
print(sbert_features_train_new.shape)
print(sbert_features_test_new.shape)
```

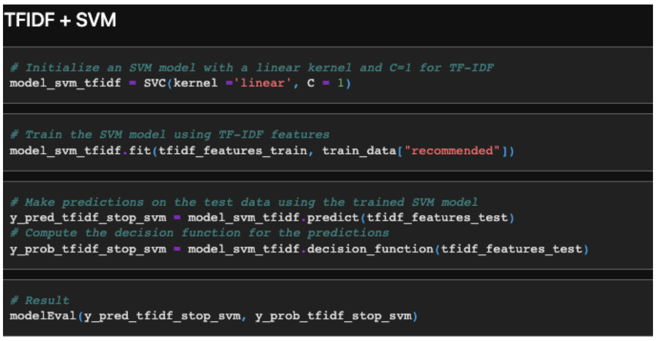
```
(19517, 768)
(4849, 768)
```

Fig. A.11 Text Transformer: SBERT



```
# A function to plot and print result
def modelEval(y_pred, y_prob):
# Evaluate the performance of a binary classification model

    # Compute evaluation metrics
    acc = accuracy_score(test_data["recommended"], y_pred)
    fi = f1_score(test_data["recommended"], y_pred)
    percision = precision_score(test_data["recommended"], y_pred)
    recall = recall_score(test_data["recommended"], y_pred)

    # Print evaluation metrics
    print ('',end='\n\n')
    print("Accuracy Percent: {:.2f} %".format(acc *100),end='\n')
    print("Accuracy: {:.4f}".format(acc *100),end='\n')
    print("f1 score: {:.4f}".format(fi),end='\n')
    print("precision score: {:.4f}".format(percision),end='\n')
    print("recall score: {:.4f}".format(recall),end='\n\n\n')

    # Print classification report
    print ('Classification Report:',end='\n\n')
    print(classification_report(test_data["recommended"],y_pred),end='\n\n')

    # Plot and print confusion matrix
    print ('Confusion Matrix',end='\n\n')
    cmatrix = confusion_matrix(test_data["recommended"],y_pred)
    group_names = ['True Neg','False Pos','False Neg','True Pos']

    group_counts = ["{0:0.0f}".format(value) for value in
                cmatrix.flatten()]

    group_percentages = ["{0:.2%}".format(value) for value in
                        cmatrix.flatten()/np.sum(cmatrix)]

    labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
              zip(group_names,group_counts,group_percentages)]

    labels = np.asarray(labels).reshape(2,2)

    ax = sns.heatmap(cmatrix, annot=labels, fmt='', cmap='Blues')

    ax.set_title('Confusion Matrix\n\n');
    ax.set_xlabel('\nPredicted Values')
    ax.set_ylabel('Actual Values ');

    ## Ticket labels - List must be in alphabetical order
    ax.xaxis.set_ticklabels(['False','True'])
    ax.yaxis.set_ticklabels(['False','True'])

    ## Display the visualization of the Confusion Matrix.
    plt.show()
```

Fig. A.12 SVM Classifier



TFIDF + SVM

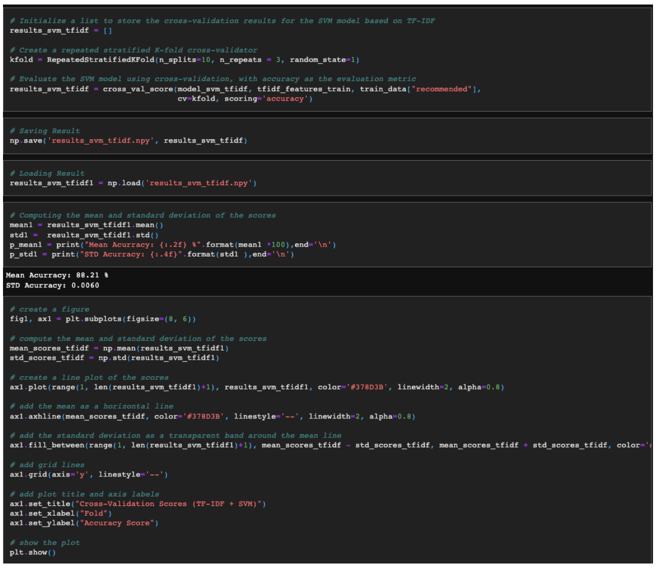
```
# Initialize an SVM model with a linear kernel and C=1 for TF-IDF
model_svm_tfidf = SVC(kernel ='linear', C = 1)
```

```
# Train the SVM model using TF-IDF features
model_svm_tfidf.fit(tfidf_features_train, train_data["recommended"])
```

```
# Make predictions on the test data using the trained SVM model
y_pred_tfidf_stop_svm = model_svm_tfidf.predict(tfidf_features_test)
# Compute the decision function for the predictions
y_prob_tfidf_stop_svm = model_svm_tfidf.decision_function(tfidf_features_test)
```

```
# Result
modelEval(y_pred_tfidf_stop_svm, y_prob_tfidf_stop_svm)
```

Fig. A.13 TF-IDF with SVM Classifier 1



```
# Initialize a list to store the cross-validation results for the SVM model based on TF-IDF
results_svm_tfidf = []

# Create a repeated stratified K-fold cross-validator
kfold = RepeatedStratifiedKFold(n_splits=10, n_repeats = 3, random_state=1)

# Evaluate the SVM model using cross-validation, with accuracy as the evaluation metric
results_svm_tfidf = cross_val_score(model_svm_tfidf, tfidf_features_train, train_data["recommended"],
                                    cv=kfold, scoring='accuracy')
```

```
# Saving Result
np.save('results_svm_tfidf.npy', results_svm_tfidf)
```

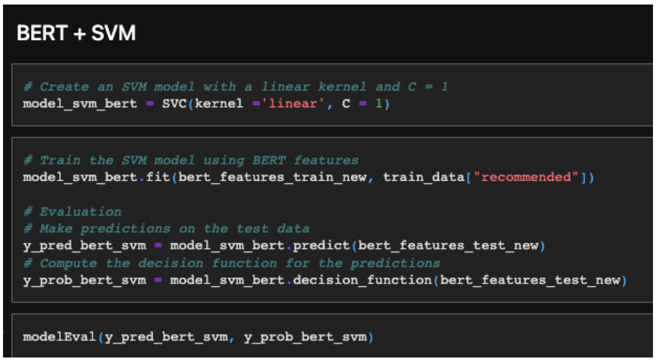
```
# Loading Result
results_svm_tfidf1 = np.load('results_svm_tfidf.npy')
```

```
# Computing the mean and standard deviation of the scores
mean1 = results_svm_tfidf1.mean()
std1 =  results_svm_tfidf1.std()
p_mean1 = print("Mean Acurracy: {:.2f} %".format(mean1 *100),end='\n')
p_std1 = print("STD Acurracy: {:.4f}".format(std1 ),end='\n')
```

```
Mean Acurracy: 88.21 %
STD Acurracy: 0.0060
```

```
# create a figure
fig1, ax1 = plt.subplots(figsize=(8, 6))

# compute the mean and standard deviation of the scores
mean_scores_tfidf = np.mean(results_svm_tfidf1)
std_scores_tfidf = np.std(results_svm_tfidf1)

# create a line plot of the scores
ax1.plot(range(1, len(results_svm_tfidf1)+1), results_svm_tfidf1, color='#378D3B', linewidth=2, alpha=0.8)

# add the mean as a horizontal line
ax1.axhline(mean_scores_tfidf, color='#378D3B', linestyle='--', linewidth=2, alpha=0.8)

# add the standard deviation as a transparent band around the mean line
ax1.fill_between(range(1, len(results_svm_tfidf1)+1), mean_scores_tfidf - std_scores_tfidf, mean_scores_tfidf + std_scores_tfidf, color='

# add grid lines
ax1.grid(axis='y', linestyle='--')

# add plot title and axis labels
ax1.set_title("Cross-Validation Scores (TF-IDF + SVM)")
ax1.set_xlabel("Fold")
ax1.set_ylabel("Accuracy Score")

# show the plot
plt.show()
```

Fig. A.14 TF-IDF with SVM Classifier 2



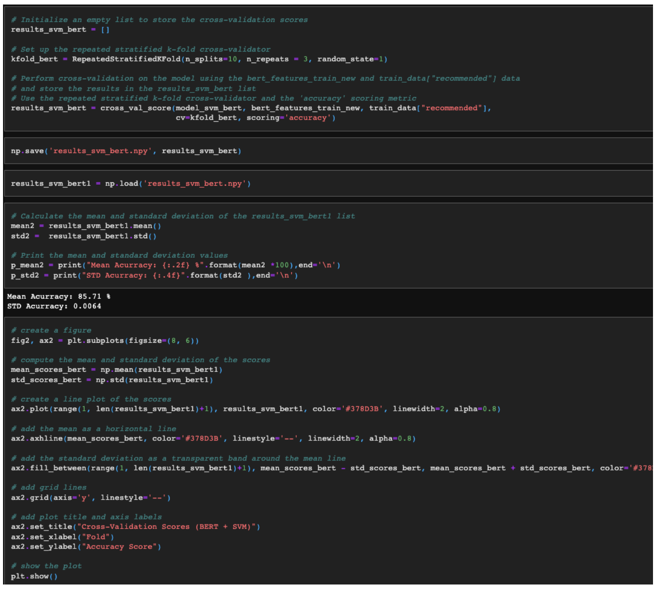
BERT + SVM

```
# Create an SVM model with a linear kernel and C = 1
model_svm_bert = SVC(kernel ='linear', C = 1)
```

```
# Train the SVM model using BERT features
model_svm_bert.fit(bert_features_train_new, train_data["recommended"])

# Evaluation
# Make predictions on the test data
y_pred_bert_svm = model_svm_bert.predict(bert_features_test_new)
# Compute the decision function for the predictions
y_prob_bert_svm = model_svm_bert.decision_function(bert_features_test_new)
```

```
modelEval(y_pred_bert_svm, y_prob_bert_svm)
```

Fig. A.15 BERT with SVM Classifier 1



```
# Initialize an empty list to store the cross-validation scores
results_svm_bert = []

# Set up the repeated stratified k-fold cross-validator
kfold_bert = RepeatedStratifiedKFold(n_splits=10, n_repeats = 3, random_state=1)

# Perform cross-validation on the model using the bert_features_train_new and train_data["recommended"] data
# and store the results in the results_svm_bert list
# Use the repeated stratified k-fold cross-validator and the 'accuracy' scoring metric
results_svm_bert = cross_val_score(model_svm_bert, bert_features_train_new, train_data["recommended"],
                                   cv=kfold_bert, scoring='accuracy')
```

```
np.save('results_svm_bert.npy', results_svm_bert)
```

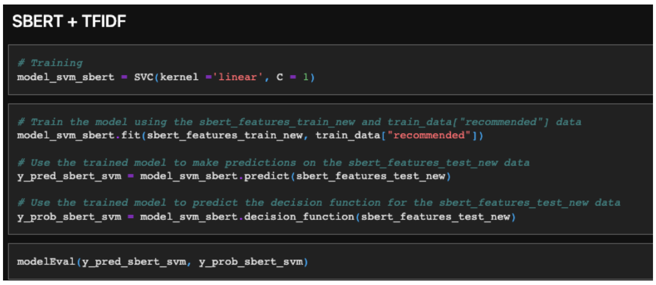
```
results_svm_bert1 = np.load('results_svm_bert.npy')
```

```
# Calculate the mean and standard deviation of the results_svm_bert1 list
mean2 = results_svm_bert1.mean()
std2 =  results_svm_bert1.std()

# Print the mean and standard deviation values
p_mean2 = print("Mean Acurracy: {:.2f} %".format(mean2 *100),end='\n')
p_std2 = print("STD Acurracy: {:.4f}".format(std2 ),end='\n')
```

```
Mean Acurracy: 85.71 %
STD Acurracy: 0.0064
```

```
# create a figure
fig2, ax2 = plt.subplots(figsize=(8, 6))

# compute the mean and standard deviation of the scores
mean_scores_bert = np.mean(results_svm_bert1)
std_scores_bert = np.std(results_svm_bert1)

# create a line plot of the scores
ax2.plot(range(1, len(results_svm_bert1)+1), results_svm_bert1, color='#378D3B', linewidth=2, alpha=0.8)

# add the mean as a horizontal line
ax2.axhline(mean_scores_bert, color='#378D3B', linestyle='--', linewidth=2, alpha=0.8)

# add the standard deviation as a transparent band around the mean line
ax2.fill_between(range(1, len(results_svm_bert1)+1), mean_scores_bert - std_scores_bert, mean_scores_bert + std_scores_bert, color='#378

# add grid lines
ax2.grid(axis='y', linestyle='--')

# add plot title and axis labels
ax2.set_title("Cross-Validation Scores (BERT + SVM)")
ax2.set_xlabel("Fold")
ax2.set_ylabel("Accuracy Score")

# show the plot
plt.show()
```

Fig. A.16 BERT with SVM Classifier 2



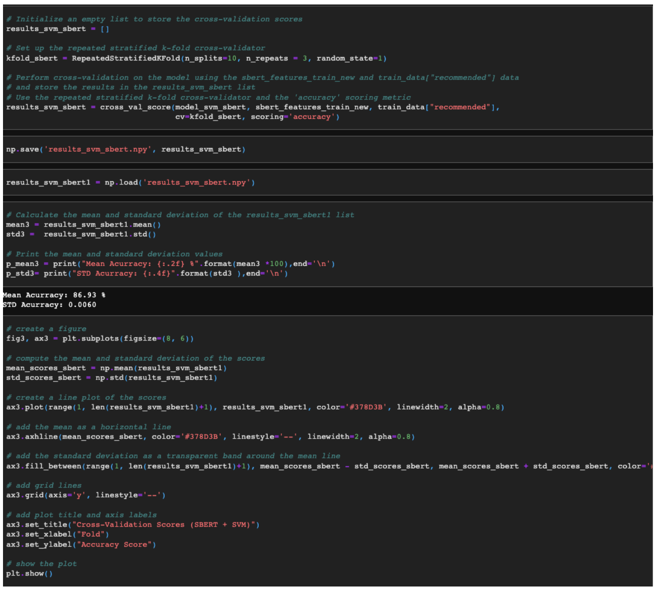
SBERT + TFIDF

```
# Training
model_svm_sbert = SVC(kernel ='linear', C = 1)
```

```
# Train the model using the sbert_features_train_new and train_data["recommended"] data
model_svm_sbert.fit(sbert_features_train_new, train_data["recommended"])

# Use the trained model to make predictions on the sbert_features_test_new data
y_pred_sbert_svm = model_svm_sbert.predict(sbert_features_test_new)

# Use the trained model to predict the decision function for the sbert_features_test_new data
y_prob_sbert_svm = model_svm_sbert.decision_function(sbert_features_test_new)
```

```
modelEval(y_pred_sbert_svm, y_prob_sbert_svm)
```

Fig. A.17 SBERT with SVM Classifier 1



```
# Initialize an empty list to store the cross-validation scores
results_svm_sbert = []

# Set up the repeated stratified k-fold cross-validator
kfold_sbert = RepeatedStratifiedKFold(n_splits=10, n_repeats = 3, random_state=1)

# Perform cross-validation on the model using the sbert_features_train_new and train_data["recommended"] data
# and store the results in the results_svm_sbert list
# Use the repeated stratified k-fold cross-validator and the 'accuracy' scoring metric
results_svm_sbert = cross_val_score(model_svm_sbert, sbert_features_train_new, train_data["recommended"],
                                    cv=kfold_sbert, scoring='accuracy')
```

```
np.save('results_svm_sbert.npy', results_svm_sbert)
```

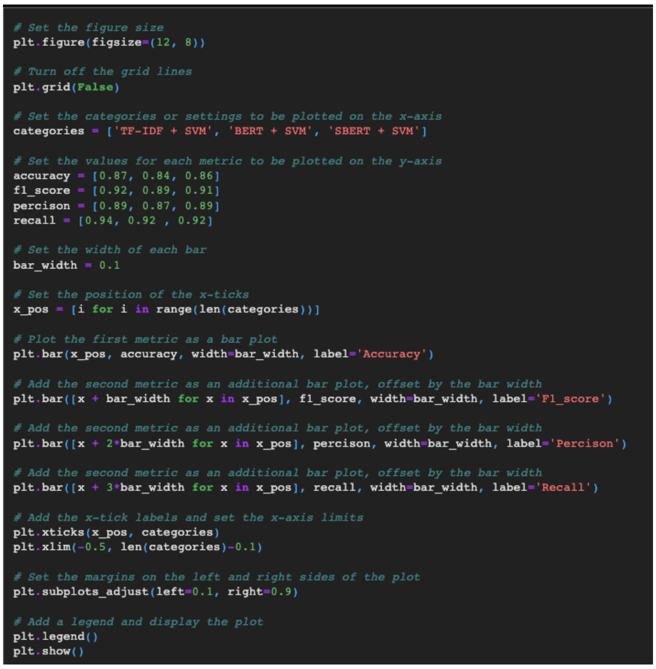
```
results_svm_sbert1 = np.load('results_svm_sbert.npy')
```

```
# Calculate the mean and standard deviation of the results_svm_sbert1 list
mean3 = results_svm_sbert1.mean()
std3 =  results_svm_sbert1.std()

# Print the mean and standard deviation values
p_mean3 = print("Mean Acurracy: {:.2f} %".format(mean3 *100),end='\n')
p_std3= print("STD Acurracy: {:.4f}".format(std3 ),end='\n')
```

```
Mean Acurracy: 86.93 %
STD Acurracy: 0.0060
```

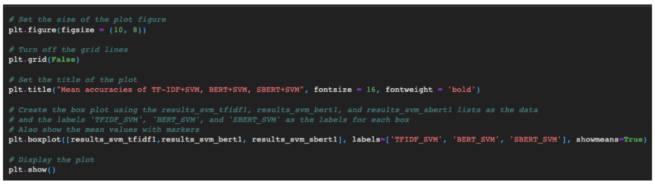
```
# create a figure
fig3, ax3 = plt.subplots(figsize=(8, 6))

# compute the mean and standard deviation of the scores
mean_scores_sbert = np.mean(results_svm_sbert1)
std_scores_sbert = np.std(results_svm_sbert1)

# create a line plot of the scores
ax3.plot(range(1, len(results_svm_sbert1)+1), results_svm_sbert1, color='#378D3B', linewidth=2, alpha=0.8)

# add the mean as a horizontal line
ax3.axhline(mean_scores_sbert, color='#378D3B', linestyle='--', linewidth=2, alpha=0.8)

# add the standard deviation as a transparent band around the mean line
ax3.fill_between(range(1, len(results_svm_sbert1)+1), mean_scores_sbert - std_scores_sbert, mean_scores_sbert + std_scores_sbert, color='

# add grid lines
ax3.grid(axis='y', linestyle='--')

# add plot title and axis labels
ax3.set_title("Cross-Validation Scores (SBERT + SVM)")
ax3.set_xlabel("Fold")
ax3.set_ylabel("Accuracy Score")

# show the plot
plt.show()
```

Fig. A.18 SBERT with SVM Classifier 2



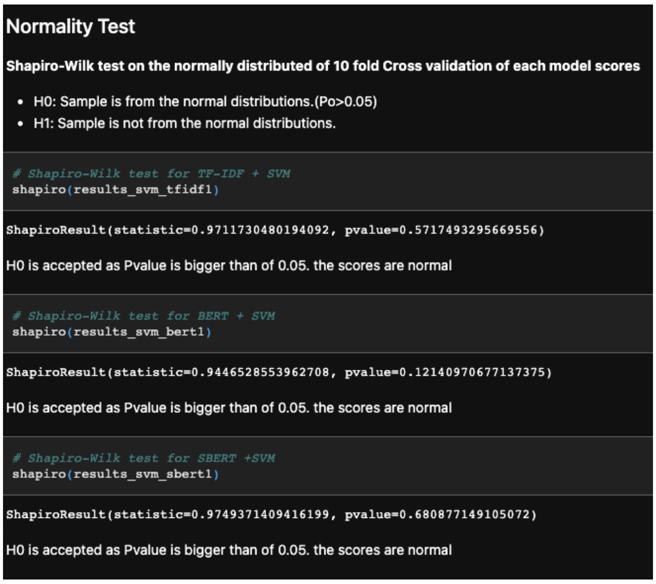
```
# Set the figure size
plt.figure(figsize=(12, 8))

# Turn off the grid lines
plt.grid(False)

# Set the categories or settings to be plotted on the x-axis
categories = ['TF-IDF + SVM', 'BERT + SVM', 'SBERT + SVM']

# Set the values for each metric to be plotted on the y-axis
accuracy = [0.87, 0.84, 0.86]
f1_score = [0.92, 0.89, 0.91]
percison = [0.89, 0.87, 0.89]
recall = [0.94, 0.92 , 0.92]

# Set the width of each bar
bar_width = 0.1

# Set the position of the x-ticks
x_pos = [i for i in range(len(categories))]

# Plot the first metric as a bar plot
plt.bar(x_pos, accuracy, width=bar_width, label='Accuracy')

# Add the second metric as an additional bar plot, offset by the bar width
plt.bar([x + bar_width for x in x_pos], f1_score, width=bar_width, label='F1_score')

# Add the second metric as an additional bar plot, offset by the bar width
plt.bar([x + 2*bar_width for x in x_pos], percison, width=bar_width, label='Percison')

# Add the second metric as an additional bar plot, offset by the bar width
plt.bar([x + 3*bar_width for x in x_pos], recall, width=bar_width, label='Recall')

# Add the x-tick labels and set the x-axis limits
plt.xticks(x_pos, categories)
plt.xlim(-0.5, len(categories)-0.1)

# Set the margins on the left and right sides of the plot
plt.subplots_adjust(left=0.1, right=0.9)

# Add a legend and display the plot
plt.legend()
plt.show()
```

Fig. A.19 Visualisation of All Evaluation Metrics



```
# Set the size of the plot figure
plt.figure(figsize = (10, 8))

# Turn off the grid lines
plt.grid(False)

# Set the title of the plot
plt.title("Mean accuracies of TF-IDF+SVM, BERT+SVM, SBERT+SVM", fontsize = 16, fontweight = 'bold')

# Create the box plot using the results_svm_tfidf1, results_svm_bert1, and results_svm_sbert1 lists as the data
# and the labels 'TFIDF_SVM', 'BERT_SVM', and 'SBERT_SVM' as the labels for each box
# Also show the mean values with markers
plt.boxplot([results_svm_tfidf1,results_svm_bert1, results_svm_sbert1], labels=['TFIDF_SVM', 'BERT_SVM', 'SBERT_SVM'], showmeans=True)

# Display the plot
plt.show()
```

Fig. A.20 Visualisation of Mean Accuracy (10-fold Cross Validation)



Normality Test

Shapiro-Wilk test on the normally distributed of 10 fold Cross validation of each model scores

- H0: Sample is from the normal distributions.(Po>0.05)
- H1: Sample is not from the normal distributions.

```
# Shapiro-Wilk test for TF-IDF + SVM
shapiro(results_svm_tfidf1)
```

```
ShapiroResult(statistic=0.9711730480194092, pvalue=0.5717493295669556)
```

H0 is accepted as Pvalue is bigger than of 0.05. the scores are normal

```
# Shapiro-Wilk test for BERT + SVM
shapiro(results_svm_bert1)
```

```
ShapiroResult(statistic=0.9446528553962708, pvalue=0.12140970677137375)
```

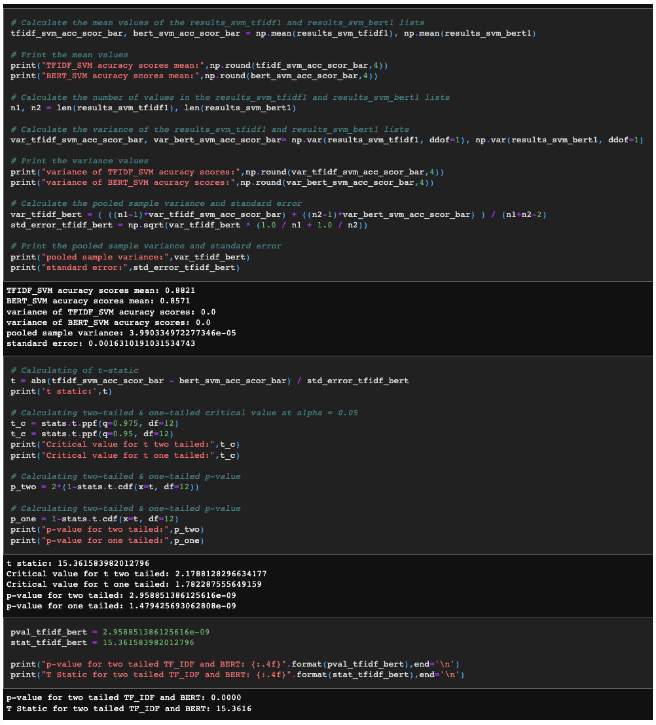
H0 is accepted as Pvalue is bigger than of 0.05. the scores are normal

```
# Shapiro-Wilk test for SBERT +SVM
shapiro(results_svm_sbert1)
```

```
ShapiroResult(statistic=0.9749371409416199, pvalue=0.680877149105072)
```

H0 is accepted as Pvalue is bigger than of 0.05. the scores are normal

Fig. A.21 Normality Test: Shapiro-Wilk Test



```
# Calculate the mean values of the results_svm_tfidf1 and results_svm_bert1 lists
tfidf_svm_acc_scor_bar, bert_svm_acc_scor_bar = np.mean(results_svm_tfidf1), np.mean(results_svm_bert1)

# Print the mean values
print("TFIDF_SVM acuracy scores mean:",np.round(tfidf_svm_acc_scor_bar,4))
print("BERT_SVM acuracy scores mean:",np.round(bert_svm_acc_scor_bar,4))

# Calculate the number of values in the results_svm_tfidf1 and results_svm_bert1 lists
n1, n2 = len(results_svm_tfidf1), len(results_svm_bert1)

# Calculate the variance of the results_svm_tfidf1 and results_svm_bert1 lists
var_tfidf_svm_acc_scor_bar, var_bert_svm_acc_scor_bar= np.var(results_svm_tfidf1, ddof=1), np.var(results_svm_bert1, ddof=1)

# Print the variance values
print("variance of TFIDF_SVM acuracy scores:",np.round(var_tfidf_svm_acc_scor_bar,4))
print("variance of BERT_SVM acuracy scores:",np.round(var_bert_svm_acc_scor_bar,4))

# Calculate the pooled sample variance and standard error
var_tfidf_bert = ( ((n1-1)*var_tfidf_svm_acc_scor_bar) + ((n2-1)*var_bert_svm_acc_scor_bar) ) / (n1+n2-2)
std_error_tfidf_bert = np.sqrt(var_tfidf_bert * (1.0 / n1 + 1.0 / n2))

# Print the pooled sample variance and standard error
print("pooled sample variance:",var_tfidf_bert)
print("standard error:",std_error_tfidf_bert)
```

```
TFIDF_SVM acuracy scores mean: 0.8821
BERT_SVM acuracy scores mean: 0.8571
variance of TFIDF_SVM acuracy scores: 0.0
variance of BERT_SVM acuracy scores: 0.0
pooled sample variance: 3.990334972277346e-05
standard error: 0.0016310191031534743
```

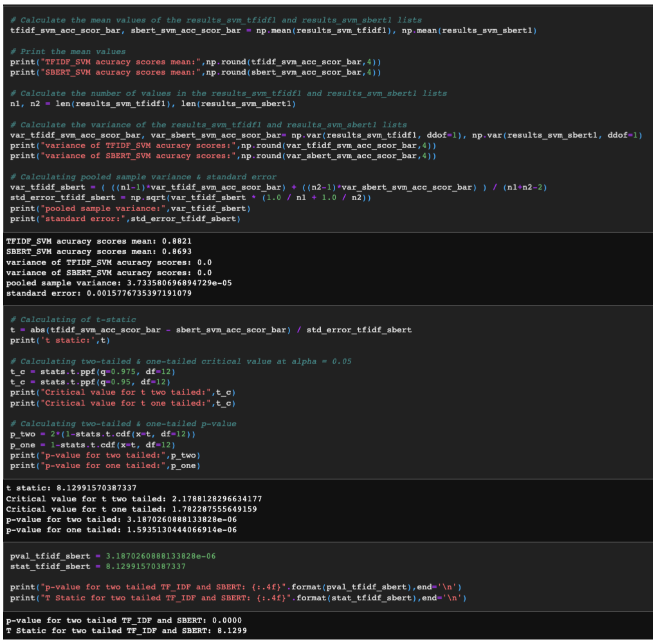
```
# Calculating of t-static
t = abs(tfidf_svm_acc_scor_bar - bert_svm_acc_scor_bar) / std_error_tfidf_bert
print('t static:',t)

# Calculating two-tailed & one-tailed critical value at alpha = 0.05
t_c = stats.t.ppf(q=0.975, df=12)
t_c = stats.t.ppf(q=0.95, df=12)
print("Critical value for t two tailed:",t_c)
print("Critical value for t one tailed:",t_c)

# Calculating two-tailed & one-tailed p-value
p_two = 2*(1-stats.t.cdf(x=t, df=12))

# Calculating two-tailed & one-tailed p-value
p_one = 1-stats.t.cdf(x=t, df=12)
print("p-value for two tailed:",p_two)
print("p-value for one tailed:",p_one)
```

```
t static: 15.361583982012796
Critical value for t two tailed: 2.1788128296634177
Critical value for t one tailed: 1.782287555649159
p-value for two tailed: 2.958851386125616e-09
p-value for one tailed: 1.479425693062808e-09
```

```
pval_tfidf_bert = 2.958851386125616e-09
stat_tfidf_bert = 15.361583982012796

print("p-value for two tailed TF_IDF and BERT: {:.4f}".format(pval_tfidf_bert),end='\n')
print("T Static for two tailed TF_IDF and BERT: {:.4f}".format(stat_tfidf_bert),end='\n')
```

```
p-value for two tailed TF_IDF and BERT: 0.0000
T Static for two tailed TF_IDF and BERT: 15.3616
```

Fig. A.22 Student T-test: TF-IDF and BERT



```
# Calculate the mean values of the results_svm_tfidf1 and results_svm_sbert1 lists
tfidf_svm_acc_scor_bar, sbert_svm_acc_scor_bar = np.mean(results_svm_tfidf1), np.mean(results_svm_sbert1)

# Print the mean values
print("TFIDF_SVM acuracy scores mean:",np.round(tfidf_svm_acc_scor_bar,4))
print("SBERT_SVM acuracy scores mean:",np.round(sbert_svm_acc_scor_bar,4))

# Calculate the number of values in the results_svm_tfidf1 and results_svm_sbert1 lists
n1, n2 = len(results_svm_tfidf1), len(results_svm_sbert1)

# Calculate the variance of the results_svm_tfidf1 and results_svm_sbert1 lists
var_tfidf_svm_acc_scor_bar, var_sbert_svm_acc_scor_bar= np.var(results_svm_tfidf1, ddof=1), np.var(results_svm_sbert1, ddof=1)
print("variance of TFIDF_SVM acuracy scores:",np.round(var_tfidf_svm_acc_scor_bar,4))
print("variance of SBERT_SVM acuracy scores:",np.round(var_sbert_svm_acc_scor_bar,4))

# Calculating pooled sample variance & standard error
var_tfidf_sbert = ( ((n1-1)*var_tfidf_svm_acc_scor_bar) + ((n2-1)*var_sbert_svm_acc_scor_bar) ) / (n1+n2-2)
std_error_tfidf_sbert = np.sqrt(var_tfidf_sbert * (1.0 / n1 + 1.0 / n2))
print("pooled sample variance:",var_tfidf_sbert)
print("standard error:",std_error_tfidf_sbert)
```

```
TFIDF_SVM acuracy scores mean: 0.8821
SBERT_SVM acuracy scores mean: 0.8693
variance of TFIDF_SVM acuracy scores: 0.0
variance of SBERT_SVM acuracy scores: 0.0
pooled sample variance: 3.733580696894729e-05
standard error: 0.0015776735397191079
```

```
# Calculating of t-static
t = abs(tfidf_svm_acc_scor_bar - sbert_svm_acc_scor_bar) / std_error_tfidf_sbert
print('t static:',t)

# Calculating two-tailed & one-tailed critical value at alpha = 0.05
t_c = stats.t.ppf(q=0.975, df=12)
t_c = stats.t.ppf(q=0.95, df=12)
print("Critical value for t two tailed:",t_c)
print("Critical value for t one tailed:",t_c)

# Calculating two-tailed & one-tailed p-value
p_two = 2*(1-stats.t.cdf(x=t, df=12))
p_one = 1-stats.t.cdf(x=t, df=12)
print("p-value for two tailed:",p_two)
print("p-value for one tailed:",p_one)
```

```
t static: 8.12991570387337
Critical value for t two tailed: 2.1788128296634177
Critical value for t one tailed: 1.782287555649159
p-value for two tailed: 3.1870260888133828e-06
p-value for one tailed: 1.5935130444066914e-06
```

```
pval_tfidf_sbert = 3.1870260888133828e-06
stat_tfidf_sbert = 8.12991570387337

print("p-value for two tailed TF_IDF and SBERT: {:.4f}".format(pval_tfidf_sbert),end='\n')
print("T Static for two tailed TF_IDF and SBERT: {:.4f}".format(stat_tfidf_sbert),end='\n')
```

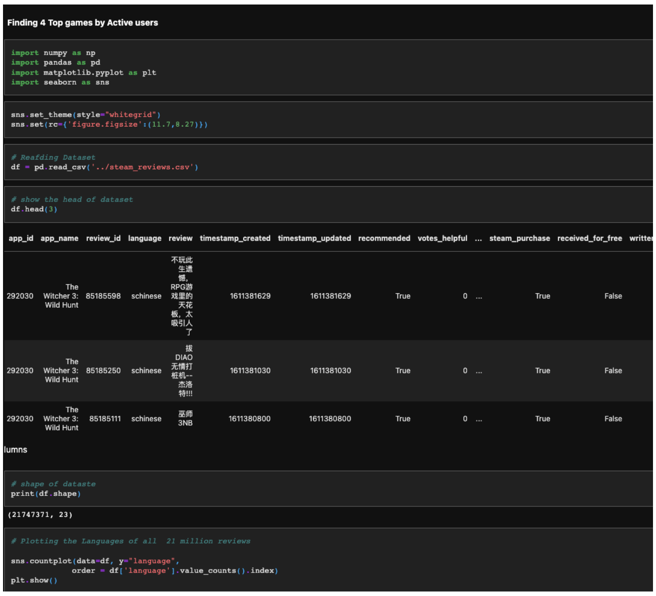
```
p-value for two tailed TF_IDF and SBERT: 0.0000
T Static for two tailed TF_IDF and SBERT: 8.1299
```

Fig. A.23 Student T-test: TF-IDF and SBERT

Appendix B

Collecting Target Data

The code for appendix B can be found on the Github under the repository "Msc_Diss_Cod" with the filename "D02124995_Mina_Jmashidian_Extract_Top4Gmaes.ipynb [1]



Finding 4 Top games by Active users

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
sns.set_theme(style="whitegrid")
sns.set(rc={'figure.figsize':(11.7,8.27)})
```

```
# Reafding Dataset
df = pd.read_csv('../steam_reviews.csv')
```

```
# show the head of dataset
df.head(3)
```

app_id	app_name	review_id	language	review	timestamp_created	timestamp_updated	recommended	votes_helpful	...	steam_purchase	received_for_free	writte
292030	The Witcher 3: Wild Hunt	85185598	schinese	不玩此生遗憾，RPG游戏里的天花板，太吸引人了	1611381629	1611381629	True	0	...	True	False	
292030	The Witcher 3: Wild Hunt	85185250	schinese	拔DIAO无情打桩机--杰洛特!!!	1611381030	1611381030	True	0	...	True	False	
292030	The Witcher 3: Wild Hunt	85185111	schinese	巫师3NB	1611380800	1611380800	True	0	...	True	False	

lumns

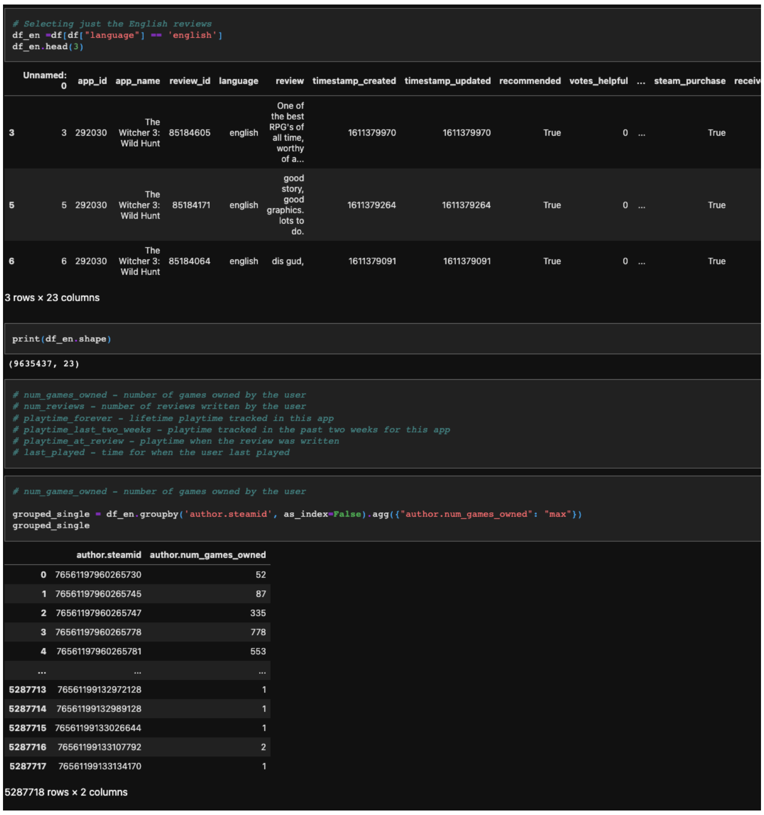
```
# shape of dataste
print(df.shape)
```

```
(21747371, 23)
```

```
# Plotting the Languages of all  21 million reviews

sns.countplot(data=df, y="language",
              order = df['language'].value_counts().index)
plt.show()
```

Fig. B.1 Target Data Collection 1

[1] https://github.com/minajm/Msc_Diss_Cod/blob/main/D20124995_Mina_Jamshidian_Extract_Top4Gmaes.ipynb



```
# Selecting just the English reviews
df_en =df[df["language"] == 'english']
df_en.head(3)
```

	Unnamed: 0	app_id	app_name	review_id	language	review	timestamp_created	timestamp_updated	recommended	votes_helpful	...	steam_purchase	receiv
3	3	292030	The Witcher 3: Wild Hunt	85184605	english	One of the best RPG's of all time, worthy of a...	1611379970	1611379970	True	0	...	True	
5	5	292030	The Witcher 3: Wild Hunt	85184171	english	good story, good graphics. lots to do.	1611379264	1611379264	True	0	...	True	
6	6	292030	The Witcher 3: Wild Hunt	85184064	english	dis gud,	1611379091	1611379091	True	0	...	True	

3 rows × 23 columns

```
print(df_en.shape)
```

```
(9635437, 23)
```

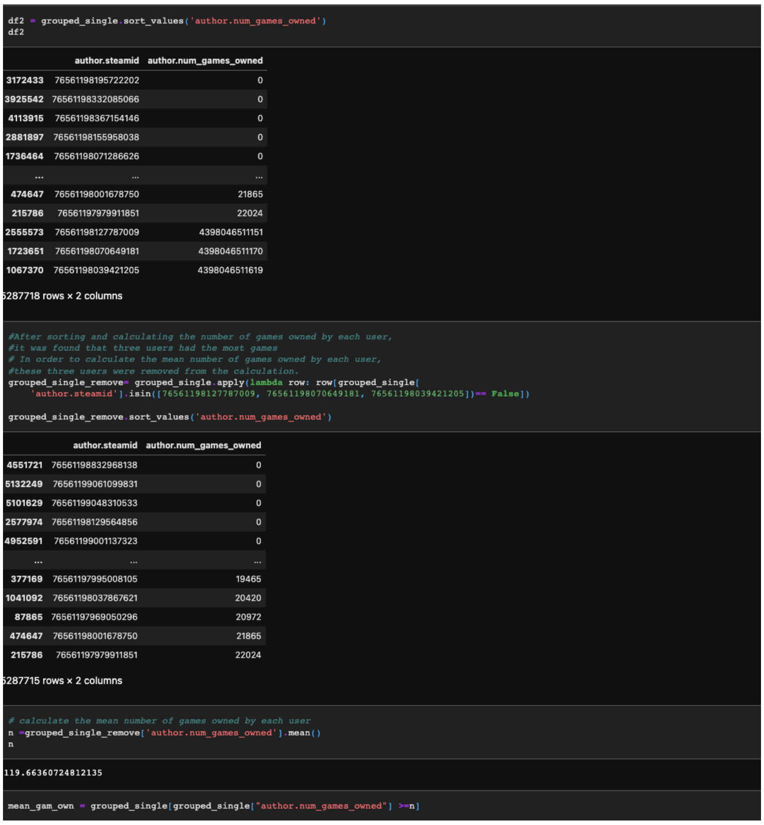
```
# num_games_owned - number of games owned by the user
# num_reviews - number of reviews written by the user
# playtime_forever - lifetime playtime tracked in this app
# playtime_last_two_weeks - playtime tracked in the past two weeks for this app
# playtime_at_review - playtime when the review was written
# last_played - time for when the user last played
```

```
# num_games_owned - number of games owned by the user

grouped_single = df_en.groupby('author.steamid', as_index=False).agg({"author.num_games_owned": "max"})
grouped_single
```

	author.steamid	author.num_games_owned
0	76561197960265730	52
1	76561197960265745	87
2	76561197960265747	335
3	76561197960265778	778
4	76561197960265781	553
...	...	...
5287713	76561199132972128	1
5287714	76561199132989128	1
5287715	76561199133026644	1
5287716	76561199133107792	2
5287717	76561199133134170	1

5287718 rows × 2 columns

Fig. B.2 Target Data Collection 2



```
df2 = grouped_single.sort_values('author.num_games_owned')
df2
```

	author.steamid	author.num_games_owned
3172433	76561198195722202	0
3925542	76561198332085066	0
4113915	76561198367154146	0
2881897	76561198155958038	0
1736464	76561198071286626	0
...	...	...
474647	76561198001678750	21865
215786	76561197979911851	22024
2555573	76561198127787009	4398046511151
1723651	76561198070649181	4398046511170
1067370	76561198039421205	4398046511619

5287718 rows × 2 columns

```
#After sorting and calculating the number of games owned by each user,
#it was found that three users had the most games
# In order to calculate the mean number of games owned by each user,
#these three users were removed from the calculation.
grouped_single_remove= grouped_single.apply(lambda row: row[grouped_single[
    'author.steamid'].isin([76561198127787009, 76561198070649181, 76561198039421205])== False])

grouped_single_remove.sort_values('author.num_games_owned')
```

	author.steamid	author.num_games_owned
4551721	76561198832968138	0
5132249	76561199061099831	0
5101629	76561199048310533	0
2577974	76561198129564856	0
4952591	76561199001137323	0
...	...	...
377169	76561197995008105	19465
1041092	76561198037867621	20420
87865	76561197969050296	20972
474647	76561198001678750	21865
215786	76561197979911851	22024

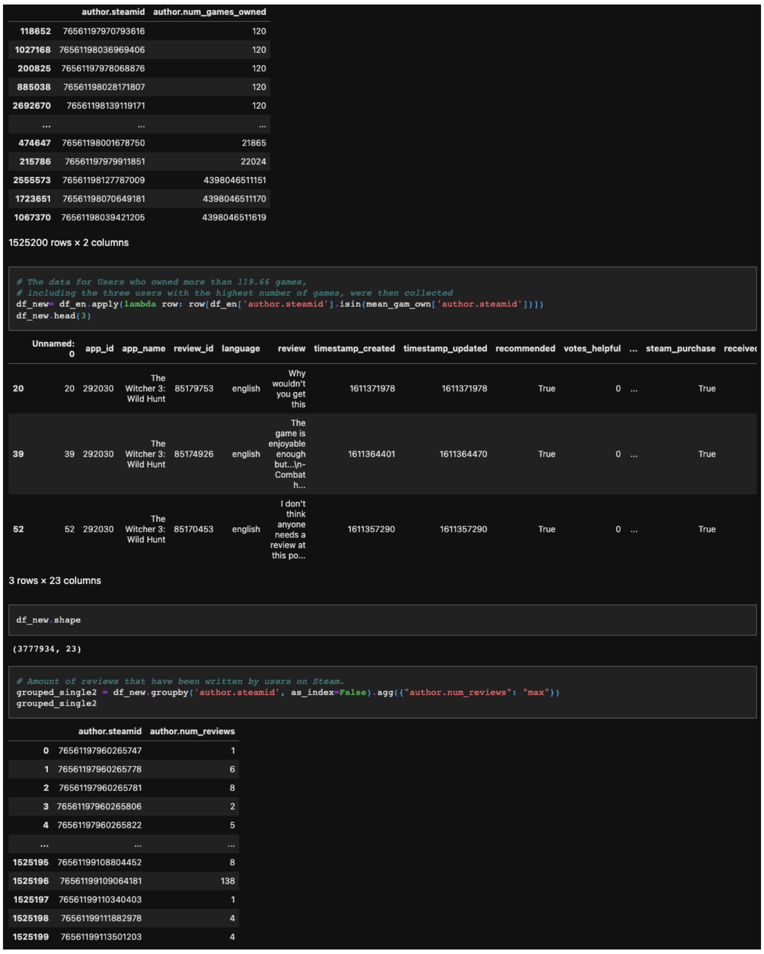
5287715 rows × 2 columns

```
# calculate the mean number of games owned by each user
n =grouped_single_remove['author.num_games_owned'].mean()
n
```

```
119.66360724812135
```

```
mean_gam_own = grouped_single[grouped_single["author.num_games_owned"] >=n]
```

Fig. B.3 Target Data Collection 3



	author.steamid	author.num_games_owned
118652	76561197970793616	120
1027168	76561198036969406	120
200825	76561197978068876	120
885038	76561198028171807	120
2692670	76561198139119171	120
...	...	...
474647	76561198001678750	21865
215786	76561197979911851	22024
2555573	76561198127787009	4398046511151
1723651	76561198070649181	4398046511170
1067370	76561198039421205	4398046511619

1525200 rows × 2 columns

```
# The data for Users who owned more than 119.66 games,
# including the three users with the highest number of games, were then collected
df_new= df_en.apply(lambda row: row[df_en['author.steamid'].isin(mean_gam_own['author.steamid'])])
df_new.head(3)
```

	Unnamed: 0	app_id	app_name	review_id	language	review	timestamp_created	timestamp_updated	recommended	votes_helpful	...	steam_purchase	receivec
20	20	292030	The Witcher 3: Wild Hunt	85179753	english	Why wouldn't you get this	1611371978	1611371978	True	0	...	True	
39	39	292030	The Witcher 3: Wild Hunt	85174926	english	The game is enjoyable enough but...\n-Combat h...	1611364401	1611364470	True	0	...	True	
52	52	292030	The Witcher 3: Wild Hunt	85170453	english	I don't think anyone needs a review at this po...	1611357290	1611357290	True	0	...	True	

3 rows × 23 columns

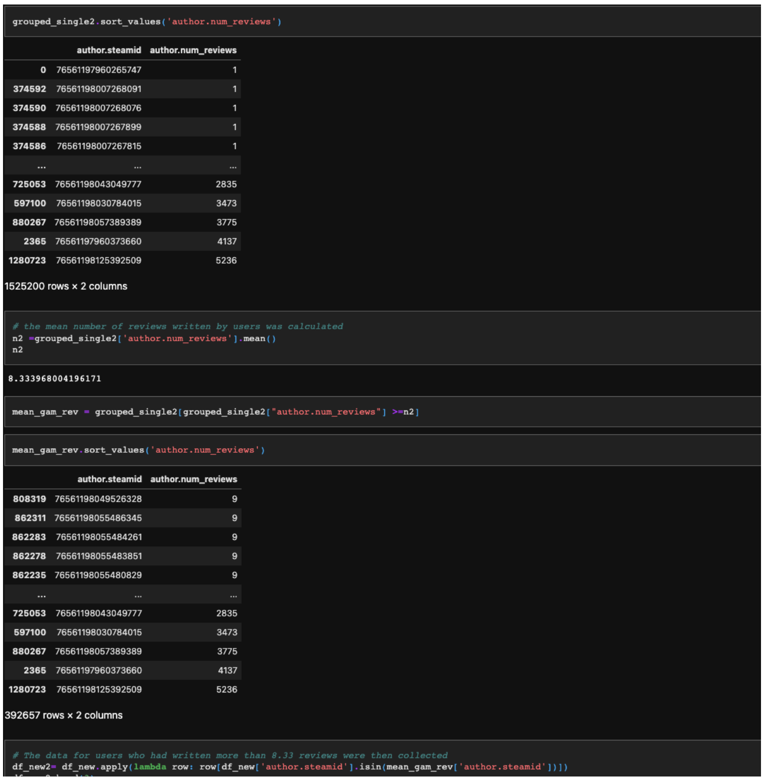
```
df_new.shape
```

```
(3777934, 23)
```

```
# Amount of reviews that have been written by users on Steam.
grouped_single2 = df_new.groupby('author.steamid', as_index=False).agg({"author.num_reviews": "max"})
grouped_single2
```

	author.steamid	author.num_reviews
0	76561197960265747	1
1	76561197960265778	6
2	76561197960265781	8
3	76561197960265806	2
4	76561197960265822	5
...	...	...
1525195	76561199108804452	8
1525196	76561199109064181	138
1525197	76561199110340403	1
1525198	76561199111882978	4
1525199	76561199113501203	4

Fig. B.4 Target Data Collection 4



```
grouped_single2.sort_values('author.num_reviews')
```

```
              author.steamid  author.num_reviews
      0    76561197960265747                   1
 374592    76561198007268091                   1
 374590    76561198007268076                   1
 374588    76561198007267899                   1
 374586    76561198007267815                   1
    ...                  ...                 ...
 725053    76561198043049777                2835
 597100    76561198030784015                3473
 880267    76561198057389389                3775
   2365    76561197960373660                4137
1280723    76561198125392509                5236

1525200 rows × 2 columns
```

```
# the mean number of reviews written by users was calculated
n2 =grouped_single2['author.num_reviews'].mean()
n2
```

```
8.333968004196171
```

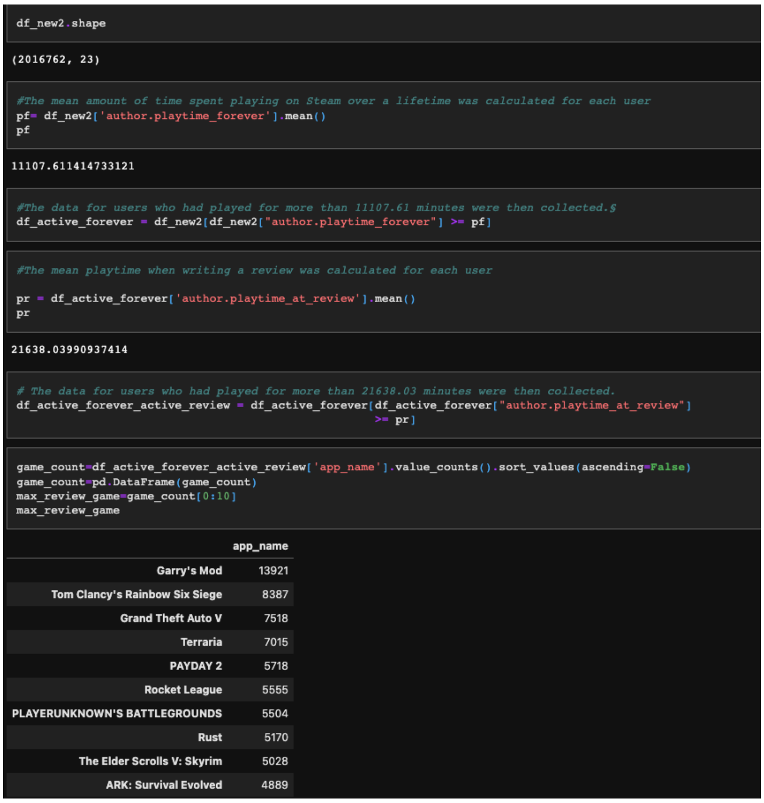
```
mean_gam_rev = grouped_single2[grouped_single2["author.num_reviews"] >=n2]
```

```
mean_gam_rev.sort_values('author.num_reviews')
```

```
              author.steamid  author.num_reviews
 808319    76561198049526328                   9
 862311    76561198055486345                   9
 862283    76561198055484261                   9
 862278    76561198055483851                   9
 862235    76561198055480829                   9
    ...                  ...                 ...
 725053    76561198043049777                2835
 597100    76561198030784015                3473
 880267    76561198057389389                3775
   2365    76561197960373660                4137
1280723    76561198125392509                5236

392657 rows × 2 columns
```

```
# The data for users who had written more than 8.33 reviews were then collected
df_new2= df_new.apply(lambda row: row[df_new['author.steamid'].isin(mean_gam_rev['author.steamid'])])
```

Fig. B.5 Target Data Collection 5



```
df_new2.shape
```

```
(2016762, 23)
```

```
#The mean amount of time spent playing on Steam over a lifetime was calculated for each user
pf= df_new2['author.playtime_forever'].mean()
pf
```

```
11107.611414733121
```

```
#The data for users who had played for more than 11107.61 minutes were then collected.$
df_active_forever = df_new2[df_new2["author.playtime_forever"] >= pf]
```

```
#The mean playtime when writing a review was calculated for each user

pr = df_active_forever['author.playtime_at_review'].mean()
pr
```

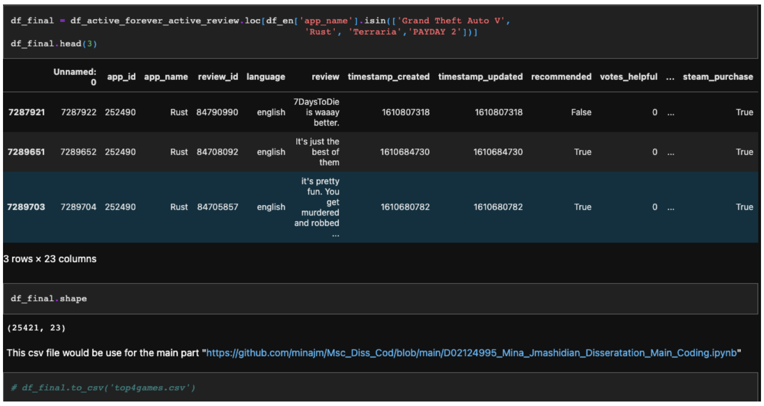
```
21638.03990937414
```

```
# The data for users who had played for more than 21638.03 minutes were then collected.
df_active_forever_active_review = df_active_forever[df_active_forever["author.playtime_at_review"]
                                                    >= pr]
```

```
game_count=df_active_forever_active_review['app_name'].value_counts().sort_values(ascending=False)
game_count=pd.DataFrame(game_count)
max_review_game=game_count[0:10]
max_review_game
```

```
                            app_name
                 Garry's Mod   13921
 Tom Clancy's Rainbow Six Siege  8387
            Grand Theft Auto V  7518
                      Terraria  7015
                      PAYDAY 2  5718
                 Rocket League  5555
 PLAYERUNKNOWN'S BATTLEGROUNDS  5504
                          Rust  5170
      The Elder Scrolls V: Skyrim  5028
            ARK: Survival Evolved  4889
```

Fig. B.6 Target Data Collection 6



```
df_final = df_active_forever_active_review.loc[df_en['app_name'].isin(['Grand Theft Auto V',
                                                               'Rust', 'Terraria','PAYDAY 2'])]
df_final.head(3)
```

	Unnamed: 0	app_id	app_name	review_id	language	review	timestamp_created	timestamp_updated	recommended	votes_helpful	...	steam_purchase
7287921	7287922	252490	Rust	84790990	english	7DaysToDie is waaay better.	1610807318	1610807318	False	0	...	True
7289651	7289652	252490	Rust	84708092	english	It's just the best of them	1610684730	1610684730	True	0	...	True
7289703	7289704	252490	Rust	84705857	english	it's pretty fun. You get murdered and robbed ...	1610680782	1610680782	True	0	...	True

3 rows × 23 columns

```
df_final.shape
```

```
(25421, 23)
```

This csv file would be use for the main part "https://github.com/minajm/Msc_Diss_Cod/blob/main/D02124995_Mina_Jmashidian_Disseratation_Main_Coding.ipynb"

```
# df_final.to_csv('top4games.csv')
```

Fig. B.7 Target Data Collection 7