Trends of Artificial Intelligence in Business Informatics

Semester Project:

Analysis of customer satisfaction using sentiment analysis

People:

Roghieh Farajialamooti, roghieh.farajialamooti001@stud.fh-dortmund.de

Seda Sensoy, seda.sensoy003@stud.fh-dortmund.de

Sraboni Bhuiyan, sraboni.bhuiyan001@stud.fh-dortmund.de

Professor:

Sebastian Bab

Winter Semester 2023

Index

Introduction

This study presents a comprehensive approach to gather and analyze sentiment from 1500 comments related to the "Lord of the Rings" series on YouTube. The data acquisition involved obtaining API and using Python to download comments. Initial sentiment labeling using Text Blob was refined through manual review due to inaccuracies, addressing challenges of bias and labeling errors. Preprocessing with NLTK included tokenization and Parts of Speech Tagging, overcoming issues faced with regular expressions for tokenization.

Sentiment analysis employed TF-IDF vectorization, with parameter tuning challenges discussed. Naïve Bayes and SVM machine learning models demonstrated effective sentiment prediction, with SVM achieving 92% accuracy. The LSTM deep learning model, applied to TF-IDF vectorized data, showed unsatisfactory results. However, a second LSTM model, utilizing a pre-trained Word2Vec model, achieved 84% accuracy, demonstrating balanced performance across sentiment categories.

The models were tested on some sentences, and consistently, Naïve Bayes and SVM showed a tendency to bias towards negative results. However, the LSTM model trained on TF-IDF vectorized data leaned towards positive sentiment. In contrast, the LSTM model trained on data vectorized using Word2Vec consistently provided accurate labels for all cases.

Data Gathering

The data used here is a database containing 1500 comments related to the "Lord of the Rings" series on the YouTube platform. To download these comments, the acquisition of an API Credential is required. The process involved several steps:

1- Create a new project in the google cloud platform.
2- Create an API Credential.
3- Enable YouTube API.
4- Download Comments using API in Python code.

The video is under the link: The Lord of the Rings (2002) - The final Battle (Of The Hornburg) - Part 1 [4K] (youtube.com)

The script initiates a request to the comment threads of a specific video ID using the provided API credentials. The objective is to access the snippet part of the comment threads with the text formatted as plain text. Then, these comments are accumulated in a list, facilitating further analysis later. For the initial pre-labeling process, Text Blob was utilized. The determination of positive, negative, or neutral labels was based on the polarity of the comments. Specifically, if the polarity was higher than 0, the comment was categorized as positive; if it was less than 0, it was considered negative, and if it equaled zero, it was marked as neutral. However, it is crucial to note that this pre-labeling served as an initial step, and the comments underwent a subsequent manual review and correction process to ensure the accuracy of the assigned labels.

Data Gathering Challenges

Initially, two challenges were encountered:

Firstly, the comments exhibited a predominant bias towards positive sentiment. To address this imbalance and create a more balanced dataset, focus was shifted to the Lord of the Rings series. It was observed that people tend to express more negative sentiments in comments related to the series compared to the movie.

Secondly, for initial pre-labeling, Text Blob was employed. However, it became evident that the labels generated, particularly for negative comments, were highly inaccurate. This necessitated a reevaluation of the labeling approach for improved accuracy in sentiment analysis.

Preprocessing

NLTK (Natural Language Toolkit) was selected over regular expressions for preprocessing due to its specialized capabilities in natural language processing tasks. NLTK provides a comprehensive suite of tools and resources, including tokenization, lemmatization, and stop word removal. While regular expressions can be used to tokenize text, they can get complicated very quickly and may not be the best choice for more complex NLP tasks. In the challenges of Tokenization, this problem is being addressed.

The Tokenization and POS Tagging were done in the preprocess_and_analyse() function, and then added as a separate column to the data.

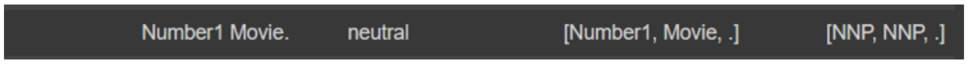
Tokenization

Tokenization is the initial step in NLP, involving the division of a text into smaller units known as "tokens." If the text is divided into sentences, it's referred to as sentence tokenization, and when it's split into individual words, it's called word tokenization. The most common method for languages with clear spaces between words is word tokenization. Furthermore, during this process, sentences are converted to lowercase.

Tokenization Challenges

In the initial stage, Regular Expressions were chosen for tokenization. However, a significant issue emerged with this approach, particularly in words like "I'm," "He's," where the apostrophe got separated from the verb. Consequently, in the POS tagging step, the verb was mislabeled as a noun. NLTK, however, correctly identifies components like "'m" or "'s" as part of the verb. Due to this critical distinction, NLTK was favored over regular expressions for the tokenization task.

The second challenge encountered involved the handling of attached numbers to digits, which required manual intervention. NLTK proved insufficient in automatically detecting and addressing this issue. An example is provided below:



```
Number1 Movie.     neutral          [Number1, Movie, .]        [NNP, NNP, .]
```

In this context, the words mistakenly attached to each other in the data were manually separated.

Parts of Speech Tagging

Parts of speech tagging, referred to as POS tagging, categorize words in a text into specific linguistic categories. These categories include nouns, verbs, adjectives, pronouns, adverbs, and more. POS tagging, also known as grammatical tagging, is achieved through tools like the NLTK POS tagger, which assigns grammatical information to each word in a sentence. Understanding the POS of words in a sentence allows for more sophisticated language analysis, which is particularly valuable in sentiment analysis. The NLTK POS tagger is a tool that automates this process, making it easier to extract grammatical insights from textual data.

Some of the most common abbreviations for POS tagging are listed below [1]:

NN	Noun
VB	Verb
JJ	Adjective
RB	Adverb
UH	Interjection
IN	Preposition

Parts of Speech Tagging Challenges

A persistent challenge in Parts of Speech Tagging is the ambiguity of words that can assume multiple roles in a sentence, such as "run," which can function as both a verb and a noun depending on the context. To address this issue, a workaround was implemented: for sentences commencing with a verb, a subject was added. For instance, "Run and get it." was transformed to "You run and get it." This approach aimed to provide clarity in disambiguating the roles of such versatile words within the sentence structure.

Named Entity Recognition

Named Entity Recognition (NER) involves identifying and categorizing named entities in text into predefined categories, such as names of people, locations, expressions of times, quantities, percentages, etc. NER aids in comprehending context and extracting pertinent information from extensive text volumes. Its applications include information retrieval, knowledge extraction, question-answering systems, and content classification. NER systems utilize linguistic grammar-based techniques, statistical models, and deep learning approaches to effectively recognize and classify entities [2].

Lemmatization

Lemmatization involves reducing words to their root form, known as the lemma. In contrast to stemming, which crudely chops off word endings, lemmatization considers the context and morphological analysis of words to bring them to their dictionary form. For example, the lemma of 'am,' 'are,' and 'is' is 'be,' and the lemma of 'mice' is 'mouse.' This process is crucial for grouping together different inflected forms of a word, enabling more accurate and effective text analysis and processing. Lemmatization finds widespread use in text preprocessing for various NLP tasks, including text normalization, information retrieval, and text mining. Its primary focus is on understanding the meaning of words in context.

Why Named Entity Recognition and Lemmatization were not considered

NER is primarily used to identify and classify entities, such as names of people. Sentiment analysis often focuses on the overall sentiment expressed in the text rather than on specific entities. In sentiment analysis, the goal is to understand the emotional tone or attitude conveyed by the text, which may not be heavily influenced by the presence of named entities.

Lemmatization involves reducing words to their base or root form. Sentiment analysis focuses on understanding the emotional tone or attitude expressed in a piece of text. It considers the overall context and how words interact with each other to convey sentiment. The sentiment of a sentence is not solely determined by individual words but by their combination and the broader context in which they appear. Additionally, machine learning models used in sentiment analysis, are trained on data where the sentiment labels are associated with the original forms of words. They learn patterns and relationships between words and sentiment in their natural forms.

Sentiment Analysis

To conduct sentiment analysis, the data must undergo preparation. At this stage, a vectorization technique is employed to enable machine processing. Initially, the comments are relabeled with numeric labels, where positive is assigned the label 2, neutral is assigned 1, and negative is assigned 0. Next, the data is divided into training and testing sets using scikit-learn's train_test_split. The key consideration here is to employ stratification, ensuring that the comments are distributed evenly across both the training and testing sets based on their numeric sentiment labels. After that, TF-IDF is utilized. This method will be elaborated upon, and the challenges in parameter tuning will be discussed. Subsequently, three models are applied to the data: first, Naive Bayes and SVM as machine learning approaches, and lastly, LSTM as a deep learning approach.

TF-IDF

Term Frequency-Inverse Document Frequency (TF-IDF) is a statistical measure employed to assess the significance of a word in a document within a given corpus. The TF-IDF value increases in proportion to the frequency of a word within the document but is balanced by the word's frequency in the entire corpus. This balancing act accounts for the fact that certain words are generally more common than others. [3]

TF-IDF Parameters

Ngram_range: N-grams play a crucial role in vectorization, as they help capture patterns and relationships between words in a text. In this case, an n-gram of 3 is employed, encompassing unigrams (individual words), bigrams (two-word sequences), and trigrams (three-word sequences). This choice allows for the identification of patterns within sequences of these varying word lengths. Bigrams and trigrams, specifically, prove useful in capturing negations, where phrases like "not happy" are semantically distinct from "happy."

Min_df: The parameter min_df is set to specify the minimum number of documents a term must appear in to be included in the vectorization process. In this instance, a term must appear in at least 5 documents to be considered.

Max_features: The parameter max_features limits the maximum number of unique terms considered during vectorization, and it is set to the number of unique words in the entire dataset which is 3080 words.

Stop_words: Stop words, such as "the" and "a," are deemed unimportant in sentiment analysis. Therefore, the stop_words parameter is configured to include the default set of English stop words.

These parameter choices were determined through a process of trial and error, emphasizing their significance in fine-tuning the vectorization process for effective sentiment analysis.

Naïve Bayes

Naïve Bayes classifiers, known for their simplicity and efficiency, use Bayes' theorem for predictions. The term "naïve" stems from the assumption of feature independence in the data. This classifier is commonly employed for tasks like text categorization and falls under the generative learning family. It models input distribution for specific categories, assuming input features are independent, resulting in fast and accurate predictions. The Multinomial is a specific implementation of the Naive Bayes algorithm, widely used for text classification tasks, especially when features represent word frequencies [4], [5].

The Naive Bayes classifier in this case operates on data that is tokenized, tagged as POS, and then vectorized using TF-IDF vectorization. It demonstrates robust performance across sentiment classes, achieving high precision, recall, and F1-score for each sentiment category. The overall accuracy of 82% signifies effective sentiment prediction on the given dataset. The classifier's balanced performance across classes suggests its reliability in accurately distinguishing sentiments. The metrics for the naïve bayes model is as follows:

	Precision	Recall	F1-Score	Support
Positive	0.85	0.86	0.86	86
Neutral	0.80	0.77	0.79	86
Negative	0.80	0.83	0.81	86
Accuracy			0.82	258
Macro Avg	0.82	0.82	0.82	258
Weighted Avg	0.82	0.82	0.82	258

Machine Learning Approach

Machine learning models for sentiment analysis are powerful tools that automate the classification of sentiments in extensive text datasets. Here, SVM, a supervised machine learning algorithm, is explained.

SVM

Support Vector Machines (SVM) operate by identifying the optimal hyperplane that effectively separates different classes within the input space, using a kernel—a function that transforms the input data from the original feature space into a higher-dimensional space. The selection of a kernel depends on the unique characteristics of the data, with common choices being: Linear Kernel, Polynomial Kernel, and Radial Basis Function Kerne.

The primary objective of SVM is to discover the hyperplane that maximizes the margin which is the distance between the decision boundary (hyperplane) and the nearest data points from each class, which are called support vectors. Gamma influences the shape of the decision boundary. A small gamma value results in a broader, smoother decision boundary, while a large gamma value leads to a more complex, intricate decision boundary. A larger margin signifies better generalization on new, unseen data. SVM introduces the concept of a soft margin, allowing for some misclassifications to achieve a wider margin. The regularization parameter (C) governs the trade-off between achieving a smooth decision boundary and minimizing misclassifications. A smaller C encourages a wider margin, characterizing a soft margin, while a larger C results in a narrower margin, known as a hard margin. The optimization objective of SVM is to minimize classification errors while simultaneously maximizing the margin. [6]

In this case, the quest for the optimal SVM classifier involves a grid search over a range of hyperparameters. The parameters utilized in the GridSearchCV of scikit-learn include verbose=10, controlling the verbosity of the output with higher values providing more detailed information about the search progress. The cv parameter is set to 5, indicating 5-fold cross-validation. This entails splitting the dataset into 5 subsets, and the model is trained and evaluated five times, each subset serving as the test set once. The n_jobs parameter shows the number of CPU cores utilized during the grid search, with a setting of -1 which means to use all available cores. The hyperparameters under consideration include C (with values 0.1, 1, 10 and 100), gamma (with options 'scale' or 'auto'), and the choice of kernel ('linear' or 'rbf'). The grid search systematically creates SVM models using different combinations of these parameters. Subsequently, it computes the mean test score and ranks the test scores, determining the combination of parameters that yields the best model performance. The chosen best estimator among 5 folds for each of 16 candidates, totaling 80 fits grid search results is: SVC(C=10, random_state=1) which is then applied to the test data for model evaluation. The classifier demonstrates robust performance across all classes, demonstrating high precision, recall, and F1-scores. Both the weighted averages and macro averages consistently align, underscoring the model's balance in performance. The overall accuracy, standing at 90%, underscores the effectiveness of the model in making correct predictions across the entire dataset. The metrics for the SVM classification is as follows:

	Precision	Recall	F1-Score	Support
Positive	0.90	0.92	0.91	86
Neutral	0.87	0.85	0.86	86
Negative	0.92	0.92	0.92	86
Accuracy			0.90	258
Macro Avg	0.90	0.90	0.90	258
Weighted Avg	0.90	0.90	0.90	258

Deep Learning Approach

Deep learning models represent a more advanced tier of machine learning, specializing in handling large and complex data sets. Long Short-Term Memory (LSTM) networks, a specific type of deep learning model is explained here.

LSTM

Long Short-Term Memory (LSTM) networks belong to the category of recurrent neural networks (RNNs) designed to capture long-term dependencies in sequential data. They prove particularly effective in tasks where the order of data and context are crucial, such as language modeling and sentiment analysis. LSTMs are chosen for their proficiency in handling sequential data and their ability to remember information over extended sequences, making them ideal for analyzing the specific context in textual data.

One of the primary challenges in utilizing LSTMs for sentiment analysis is the model's complexity, resulting in longer training times and a need for more data to achieve optimal performance. Due to their deep and intricate structure, LSTMs are susceptible to overfitting, particularly when trained on limited data. Therefore, a suggestion would be to apply the model on a larger dataset.

Two LSTM models are employed here. The first one operates on vectorized data using TF-IDF, while the other utilizes a pre-trained Word2Vec model to automatically handle the embedding process without the need for an additional layer for training. The only difference between these two models lies in the preprocessing step. In the first model, the data utilizes vectorization from the TF-IDF vectorizer. The maximum sequence length is determined by choosing between the maximum length of the train and test data, while the input dimension is set to the number of unique words. In the second model, a tokenizer is fitted on the train and test data to convert the text data into sequences of integers, where each integer represents a specific word. The pad_sequences()

function is then employed to ensure that all sequences have the same length, padding shorter sequences with zeros.

When constructing the LSTM model, a bidirectional architecture is employed to capture context from both past and future data points within a sequence. The model comprises an embedding layer, transforming text input into dense vectors of fixed size (configured with an output dimension of 50 in the code), two bidirectional LSTM layers with 50 units each for deep learning of text sequences, and a dense layer with a softmax activation function for classification into three categories. To prevent overfitting, a dropout rate of 0.2 is applied, randomly deactivating a fraction of neurons during training to ensure the network learns more robust features that generalize better to unseen data.

The model is trained for 10 epochs with a batch size of 16, and validation data is utilized for performance evaluation. It is compiled using the 'adam' optimizer and 'sparse_categorical_crossentropy' as the loss function. Throughout the epochs, the model shows improvement, indicating effective learning.

As observed in the metrics report for the first LSTM model, which was executed on TF-IDF vectorized data, the model's performance on the test dataset is unsatisfactory, evident from a low accuracy of 33.33%. The elevated test loss of 1.0987 indicates significant deviations in the model's predictions from the actual sentiments. For the 'positive' and 'neutral' classes, precision, recall, and F1-score are all reported as 0.00, underscoring the model's inability to effectively predict these sentiments. On the contrary, for the 'negative' class, the model achieves a recall of 1.00, correctly identifying all instances of negative sentiment. However, precision and F1-score are low at 0.33 and 0.50, respectively. The model encounters challenges in generalizing to sentiments other than 'negative,' resulting in an unbalanced performance across different sentiment categories. Further optimization or adjustments to the model architecture, hyperparameters, or dataset may be necessary to enhance overall sentiment classification accuracy. The metrics for the first LSTM model is as follows:

	Precision	Recall	F1-Score	Support
Positive	0	0	0	86
Neutral	0	0	0	86
Negative	0.40	1	0.57	86
Accuracy			0.40	258
Macro Avg	0.13	0.33	0.19	258
Weighted Avg	0.16	0.40	0.23	258

However, the second model demonstrates notable success, with test accuracy of 84 percent. The pretrained model achieves a balanced performance across different sentiment categories, demonstrating high precision and recall for each class. The macro average F1-score is 0.85, indicating overall effectiveness. The weighted average F1-score is 0.83, considering the class imbalance. These metrics collectively highlight the model's proficiency in accurately classifying sentiments in the dataset. The metrics for the second LSTM model is as follows:

	Precision	Recall	F1-Score	Support
Positive	0.84	0.87	0.85	86
Neutral	0.83	0.79	0.81	86
Negative	0.83	0.83	0.83	86
Accuracy			0.83	258

Macro Avg	0.83	0.83	0.83	258
Weighted Avg	0.83	0.83	0.83	258

Comparison of Models

Naïve Bayes

The Naïve Bayes classifier exhibits simplicity and efficiency, achieving an overall accuracy of 81%. Its balanced performance across sentiment classes, with high precision, recall, and F1-scores for each category, underscores its reliability in accurately distinguishing sentiments. However, it might lack the complexity to capture intricate patterns in the data, especially when dealing with nuanced sentiments. The Confusion Matrix for this model is provided below:

	Predicted Positive	Predicted Neutral	Predicted Negative
Actual Positive	74	10	2
Actual Neutral	18	63	5
Actual Negative	10	7	69

SVM

The Support Vector Machine (SVM) model, optimized through a grid search over hyperparameters, achieves an overall accuracy of 92%. This indicates its effectiveness in making correct predictions across the entire dataset. The SVM classifier demonstrates robust performance, with high precision, recall, and F1-scores for each sentiment category. Its ability to handle non-linear relationships and capture complex patterns in the data contributes to its outstanding performance, and it has the lowest misclassification in the confusion matrix.

	Predicted Positive	Predicted Neutral	Predicted Negative
Actual Positive	79	7	0
Actual Neutral	11	75	0
Actual Negative	0	14	72

LSTM

The Long Short-Term Memory (LSTM) models showcase varying degrees of success. The first LSTM model, using TF-IDF vectorization, performs unsatisfactorily with a low accuracy of 33%. Its challenges in generalizing to sentiments other than 'negative' highlight potential limitations in the model's architecture or hyperparameters. On the other hand, the second LSTM model, utilizing a pre-trained Word2Vec model, achieves a test accuracy of 84%, demonstrating notable success. Its balanced performance across sentiment categories, with high precision, recall, and F1-scores, indicates proficiency in accurately classifying sentiments.

The confusion matrix for the LSTM with TF-IDF vectorization is shown below:

	Predicted Positive	Predicted Neutral	Predicted Negative
Actual Positive	0	86	0
Actual Neutral	0	86	0

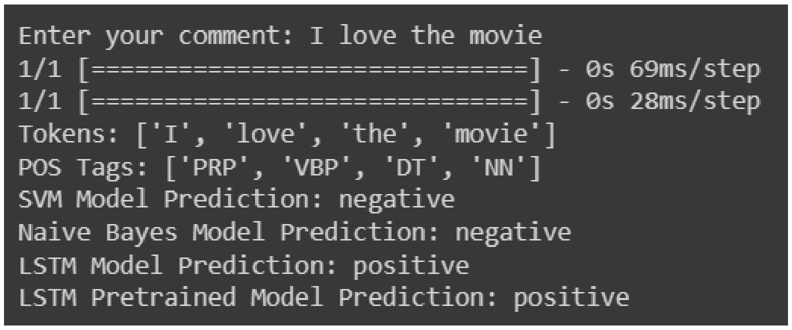
Actual Negative	0	0	86

The confusion matrix for the LSTM with Word2Vec vectorization is shown below:

	Predicted Positive	Predicted Neutral	Predicted Negative
Actual Positive	74	10	2
Actual Neutral	10	76	0
Actual Negative	12	16	68

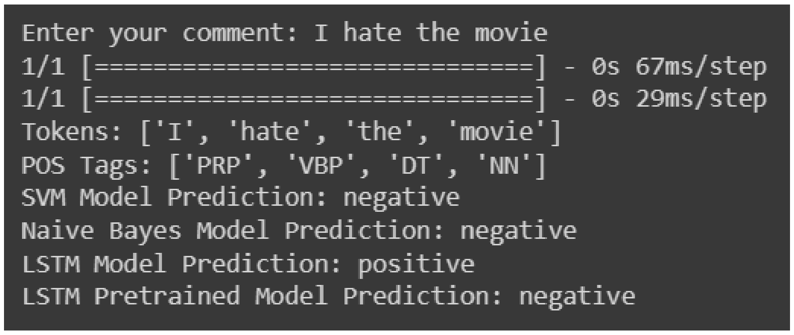
The models were tested on the given sentences, and consistently, Naïve Bayes and SVM showed a tendency to bias towards negative results. However, the LSTM model trained on TF-IDF vectorized data leaned towards positive sentiment. In contrast, the LSTM model trained on data vectorized using Word2Vec consistently provided accurate labels for all cases.

Sentence: I love the movie



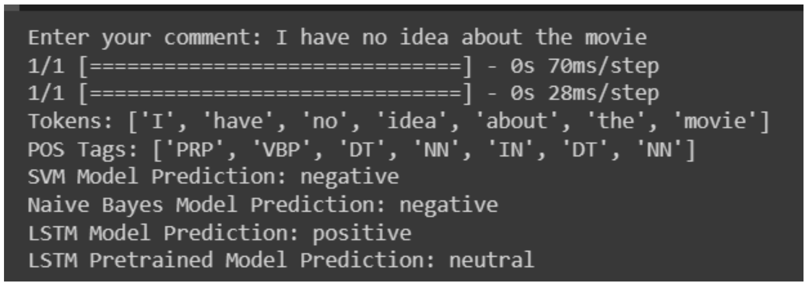
```
Enter your comment: I love the movie
1/1 [==============================] - 0s 69ms/step
1/1 [==============================] - 0s 28ms/step
Tokens: ['I', 'love', 'the', 'movie']
POS Tags: ['PRP', 'VBP', 'DT', 'NN']
SVM Model Prediction: negative
Naive Bayes Model Prediction: negative
LSTM Model Prediction: positive
LSTM Pretrained Model Prediction: positive
```

Sentence: I hate the movie



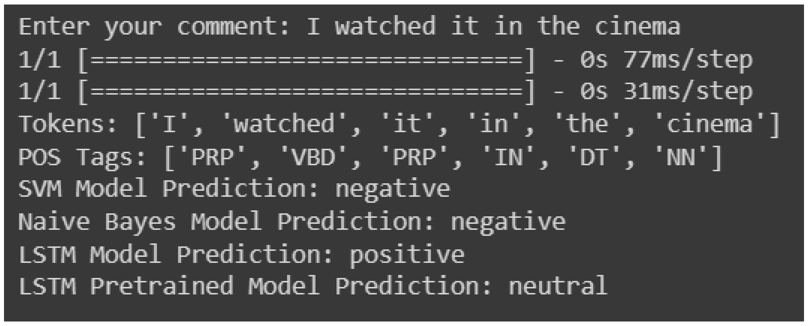
```
Enter your comment: I hate the movie
1/1 [==============================] - 0s 67ms/step
1/1 [==============================] - 0s 29ms/step
Tokens: ['I', 'hate', 'the', 'movie']
POS Tags: ['PRP', 'VBP', 'DT', 'NN']
SVM Model Prediction: negative
Naive Bayes Model Prediction: negative
LSTM Model Prediction: positive
LSTM Pretrained Model Prediction: negative
```

Sentence: I have no idea about the movie



```
Enter your comment: I have no idea about the movie
1/1 [==============================] - 0s 70ms/step
1/1 [==============================] - 0s 28ms/step
Tokens: ['I', 'have', 'no', 'idea', 'about', 'the', 'movie']
POS Tags: ['PRP', 'VBP', 'DT', 'NN', 'IN', 'DT', 'NN']
SVM Model Prediction: negative
Naive Bayes Model Prediction: negative
LSTM Model Prediction: positive
LSTM Pretrained Model Prediction: neutral
```

Sentence: I watched it in the cinema



```
Enter your comment: I watched it in the cinema
1/1 [==============================] - 0s 77ms/step
1/1 [==============================] - 0s 31ms/step
Tokens: ['I', 'watched', 'it', 'in', 'the', 'cinema']
POS Tags: ['PRP', 'VBD', 'PRP', 'IN', 'DT', 'NN']
SVM Model Prediction: negative
Naive Bayes Model Prediction: negative
LSTM Model Prediction: positive
LSTM Pretrained Model Prediction: neutral
```

Conclusion

Among the models, LSTM with embedded word vectorization exhibited the best results and highest prediction accuracy, correctly assigning all labels. According to the confusion matrix, SVM had the lowest misclassifications during training. However, it is worth noting that LSTM with TF-IDF requires a larger dataset to achieve precise results.

References

[1] "POS Tagging with NLTK and Chunking in NLP," [Online]. Available: https://www.guru99.com/pos-tagging-chunking-nltk.html. [Accessed 10 01 2024].

[2] "named entity recognition (NER)," [Online]. Available: https://www.techtarget.com/whatis/definition/named-entity-recognition-NER. [Accessed 10 01 2024].

[3] "Understanding TF-IDF (Term Frequency-Inverse Document Frequency)," [Online]. Available: https://www.geeksforgeeks.org/understanding-tf-idf-term-frequency-inverse-document-frequency/. [Accessed 10 01 2024].

[4] "Naive Bayes," [Online]. Available: https://scikit-learn.org/stable/modules/naive_bayes.html. [Accessed 10 01 2024].

[5] "sklearn.naive_bayes.MultinomialNB," [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html. [Accessed 10 01 2024].

[6] "Intuition behind C and Gamma in Support Vector Machines: A Visual and Coding Perspective," 10 01 2024. [Online]. Available: https://medium.com/@megha.natarajan/intuition-behind-c-and-gamma-in-support-vector-machines-a-visual-and-coding-perspective-310029ea4512.