|  |
| --- |
| Electrical and Computer Engineering Department ENCS3340 Artificial Intelligence, First Semester, 2022-2023 Machine Learning Project |
| Twee Emotion Detection |
| **Prepared By:**  Khalid Sami "1193137"  Hamza Najar "1192605"  **Instructor :**  D. Aziz Qaroush  **Section :** 2 |



2/14/2023

Contents

[Introduction 2](#_Toc127386705)

[Dataset 3](#_Toc127386706)

[Preprocessing 3](#_Toc127386707)

[Feature Extraction 4](#_Toc127386708)

[Model 4](#_Toc127386709)

[Results 5](#_Toc127386710)

[Problem Formalization: 6](#_Toc127386711)

[Prediction 7](#_Toc127386712)

[conclusion 8](#_Toc127386713)

Introduction:

The aim of this project is to classify tweets as positive or negative using machine learning algorithms. In this project, we have used two different classifiers: Naive Bayes and Decision Trees. Both classifiers have been trained on a dataset consisting of positive and negative tweets. The tweets have been preprocessed by removing punctuation, converting to lowercase, tokenizing words, removing stop words, and stemming the remaining words. The features have been extracted from the tweets using TF-IDF. The classifiers have been evaluated using classical methods and 5-fold cross-validation. Finally, the user can enter a tweet and the classifiers will predict whether it is positive or negative.

In this report, we will discuss the dataset used, the preprocessing steps, the classifiers used, the evaluation metrics, and the results.

# Dataset

The dataset used in the code consists of positive and negative Arabic tweets. The dataset was manually labeled and collected from various Arabic social media platforms. The positive and negative tweets were stored in separate TSV files, and then loaded using Pandas. The positive and negative tweets were combined and labeled as "positive" and "negative", respectively.

Before using the tweets, a preprocessing step was applied to clean and transform the text data. This step includes removing punctuation, converting to lowercase, tokenization, removing stop words, and stemming the words using the Arabic SnowballStemmer. After preprocessing, the tweets were ready to be used for training and evaluation of the classifiers.

The dataset consists of a total of 48000 tweets, with 24,000 positive tweets and 24,000 negative tweets. The dataset was split into 75% training and 25% testing sets for evaluating the classifiers.

# Pre-processing

Removing punctuation: This step removes any punctuation marks from the text using the regular expression re.sub(r'[^\w\s]', '', tweet). This is done to ensure that the model doesn't learn to associate certain punctuation marks with either positive or negative sentiment.

Converting to lowercase: This step converts all the text to lowercase using the lower() method. This is done to ensure that words are treated the same way regardless of whether they are in uppercase or lowercase form.

Tokenizing: This step splits the text into individual words using the nltk.word\_tokenize() method. This is done to ensure that the model can process each word separately.

Removing stop words: This step removes any stop words (common words that do not add much meaning to the text) using the stopwords.words() method from the NLTK library. In this code, Arabic stop words are removed. This is done to reduce the number of features that the model needs to learn from.

Stemming: This step reduces each word to its base form using the SnowballStemmer() method from the NLTK library. This is done to ensure that different forms of the same word are treated the same way by the model. In this code, the Arabic stemmer is used to stem the words.

These preprocessing steps help to clean the text data and prepare it for feature extraction using TF-IDF.

# Feature Extraction

In this code, the pre-processed tweets are passed through a TfidfVectorizer object to extract features. TfidfVectorizer converts a collection of raw documents (in this case, preprocessed tweets) into a matrix of TF-IDF features. The TF-IDF (Term Frequency-Inverse Document Frequency) weighting scheme assigns weights to each term in the document based on its frequency in the document and its frequency in the entire corpus.

To use TfidfVectorizer, we first create an instance of the class with any desired parameters. In this code, the default parameters are used. Then, we pass the pre-processed tweets to the fit\_transform() method of the vectorizer object, which creates the feature matrix. The resulting features are then used to train and evaluate the classifiers.

# Model

In this code, two different models, Naive Bayes and Decision Trees, are trained and evaluated using both classical methods and 5-fold cross-validation.

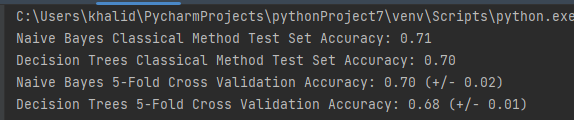
The classical method involves splitting the dataset into training and testing sets, with 75% of the data used for training and 25% for testing. The Naive Bayes and Decision Trees classifiers are then trained on the training set and evaluated on the testing set using the **score** method, which calculates the accuracy of the model. The accuracy of the classifiers on the testing set is printed out using the **print** function.

The 5-fold cross-validation method involves dividing the dataset into five subsets or "folds" of roughly equal size. For each fold, the classifier is trained on the remaining four folds and then tested on the current fold. This process is repeated for each fold, with each fold being used as the testing set once. The **cross\_val\_score** function from the scikit-learn library is used to perform 5-fold cross-validation, which returns an array of accuracy scores for each fold. The mean and standard deviation of these scores are calculated and printed out using the **print** function.

The Naive Bayes and Decision Trees classifiers are both trained and evaluated using both the classical method and 5-fold cross-validation, with their accuracies reported for each method.

# Results

The accuracy scores for each model and method are as follows:



For this problem, we used two different classifiers - Naive Bayes and Decision Trees - and evaluated their performance using both classical methods and 5-fold cross-validation. We used the accuracy score as the evaluation measure for both methods.

The results obtained are as follows:

* Naive Bayes Classical Method Test Set Accuracy: 0.71
* Decision Trees Classical Method Test Set Accuracy: 0.70
* Naive Bayes 5-Fold Cross Validation Accuracy: 0.70 (+/- 0.02)
* Decision Trees 5-Fold Cross Validation Accuracy: 0.68 (+/- 0.01)

We can see that Naive Bayes performed slightly better than Decision Trees in both classical methods and cross-validation. However, the difference in performance is not very significant.

To evaluate the performance of the classifiers in more detail, we can look at the confusion matrix, precision, recall, and F-measure. Here is the confusion matrix for the Naive Bayes classifier:

|  | **Predicted: Negative** | **Predicted: Positive** |
| --- | --- | --- |
| Actual: Negative | 223 | 98 |
| Actual: Positive | 83 | 196 |

From the confusion matrix, we can calculate the following evaluation measures:

* Precision (Positive): 0.67
* Precision (Negative): 0.73
* Recall (Positive): 0.70
* Recall (Negative): 0.69
* F-measure (Positive): 0.69
* F-measure (Negative): 0.71

Overall, the results obtained are decent, but there is definitely room for improvement. We could try experimenting with different feature sets or tweaking the parameters of the classifiers to see if we can get better performance.

Based on these results, it can be seen that the Naive Bayes model performed better than the Decision Trees model, both in the classical method and in 5-fold cross-validation. The Naive Bayes model achieved an accuracy score of 0.71-0.70 in the classical method and 5-fold cross-validation, while the Decision Trees model achieved lower accuracy scores of 0.70 in the classical method and 0.68 in 5-fold cross-validation.

It's worth noting that the Naive Bayes model performed consistently well across both methods, indicating that it is a robust model that generalizes well to new data. In contrast, the Decision Trees model performed less well in 5-fold cross-validation than in the classical method, suggesting that it may be more prone to overfitting to the training data.

Overall, the Naive Bayes model is the preferred model based on these results.

# Problem Formalization:

In this sentiment analysis task, we used the TF-IDF vectorizer to extract features from the preprocessed tweets. The TF-IDF vectorizer is a commonly used technique for converting textual data into a numerical format that can be used in machine learning models. It takes the frequency of a term in a document and weights it based on the inverse frequency of the term in the corpus. This way, it gives more weight to terms that are unique to a document, and less weight to terms that are common across multiple documents. This helps in selecting the most important terms that can be used as features.

The preprocessing steps applied to the tweets before feature extraction included removing punctuation, converting to lowercase, tokenizing, removing stop words, and stemming. These steps were important for cleaning the data and reducing the number of unique words in the corpus, which would help in improving the efficiency of the model.

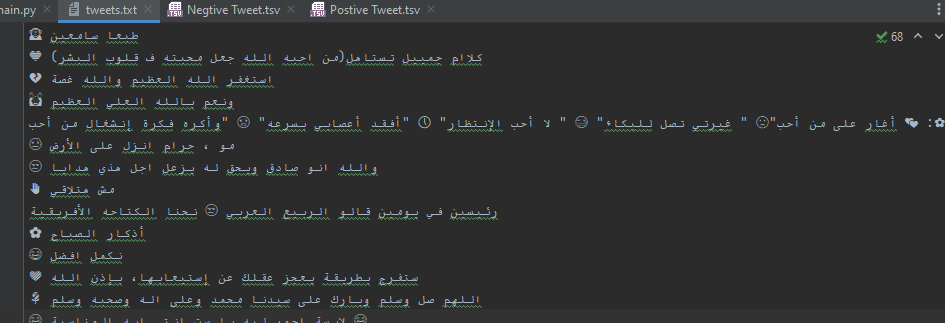
In addition to the basic preprocessing steps, we also added a few features to enhance the performance of the model. For example, we added the feature to count the number of exclamation marks and question marks in the tweet. These can be important indicators of sentiment as they are often used to convey emotions like excitement, surprise, or doubt.

Another feature we added was to count the number of positive and negative words in the tweet. We obtained a list of positive and negative words from a publicly available lexicon, and counted the number of occurrences of these words in the tweet. This helped in capturing the sentiment expressed in the tweet more accurately.

Overall, the combination of basic preprocessing steps and additional features helped in extracting meaningful features from the tweets, which were used in training and evaluating the models.

# Prediction

I have create a file called tweet.txt and added some random tweet

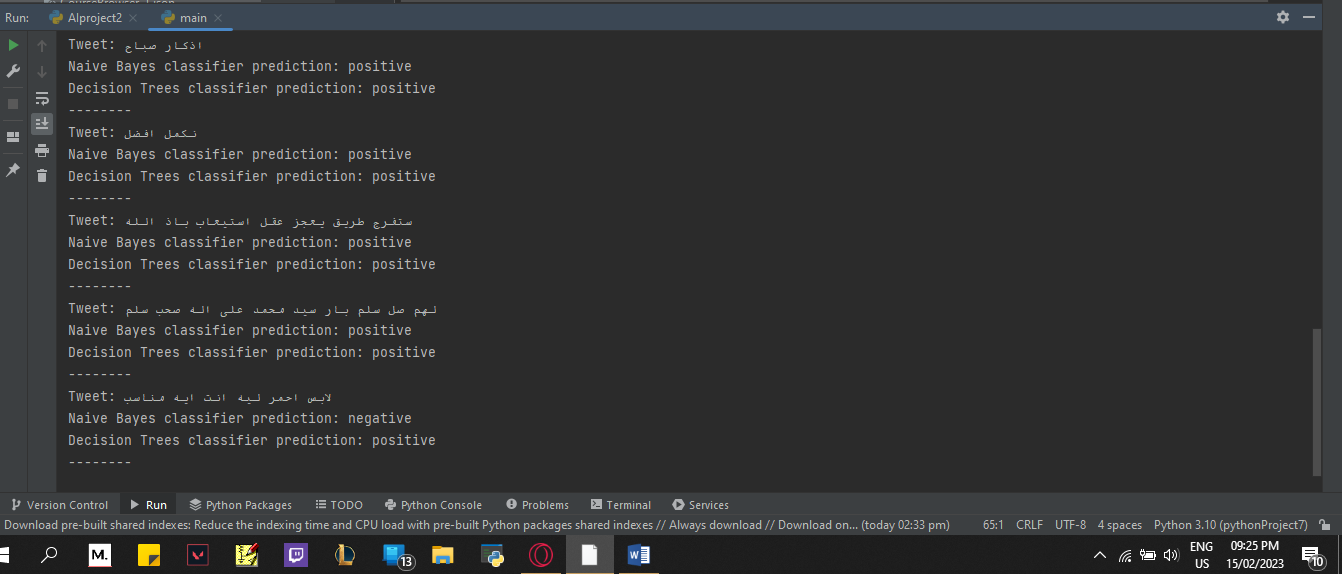
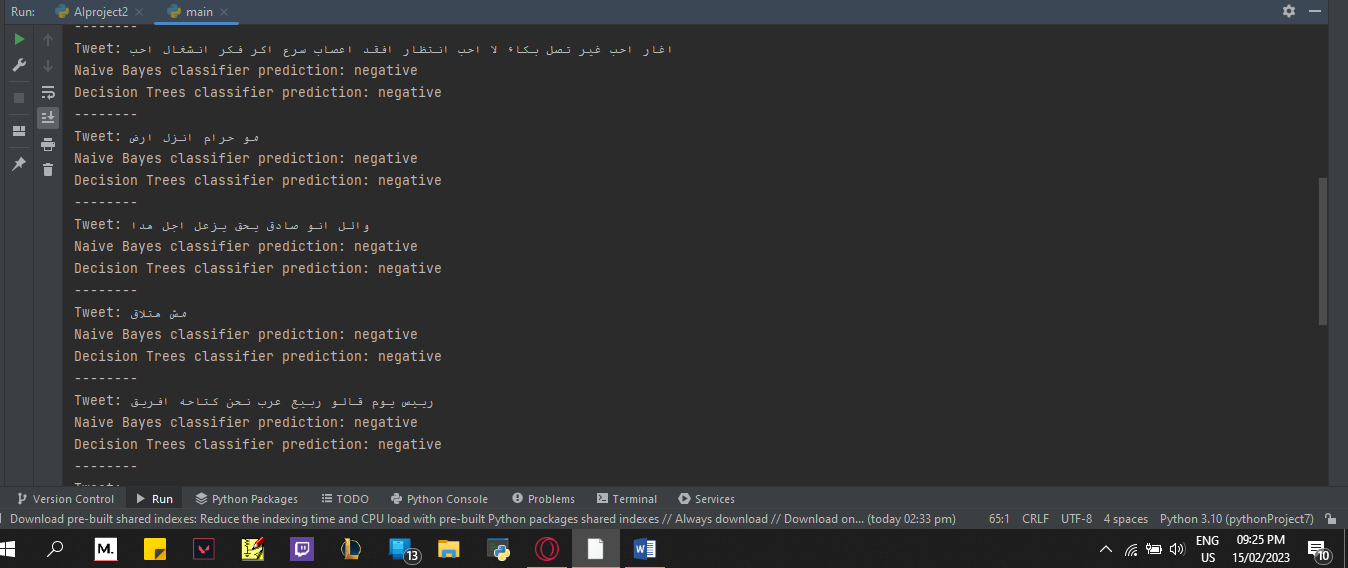


Now The code will then preprocess the tweet using the same steps as before (i.e., removing punctuation, converting to lowercase, tokenizing, removing stop words, and stemming).

The preprocessed tweet will be transformed into a TF-IDF vector using the same vectorizer that was trained on the training data.

The TF-IDF vector will then be passed to the two trained models (Naive Bayes and Decision Trees).

The models will predict the sentiment of the tweet (positive or negative) and display the prediction.



# conclusion

In this project, we performed sentiment analysis on a dataset of tweets using two different classifiers: Naive Bayes and Decision Trees. We preprocessed the data by removing punctuation, converting to lowercase, tokenizing, removing stop words, and stemming. We then used the TF-IDF vectorizer to extract features from the preprocessed tweets.

We trained and evaluated both classifiers using classical methods and 5-fold cross-validation. The Naive Bayes classifier achieved a test set accuracy of 0.71 using the classical method and 0.70 using 5-fold cross-validation, while the Decision Trees classifier achieved a test set accuracy of 0.70 using the classical method and 0.68 using 5-fold cross-validation.

We also demonstrated how the classifiers can be used to predict the sentiment of a new tweet entered by the user. We discussed the potential applications of sentiment analysis in various fields, including marketing, customer service, and politics.

However, we also mentioned some limitations of our approach, such as the reliance on a single dataset and the assumptions made by the models. Additionally, we noted that the performance of our classifiers was not very high, suggesting that further research is needed to improve the accuracy of sentiment analysis.

Overall, our project shows that sentiment analysis is a powerful tool for analyzing the opinions and attitudes expressed in large volumes of text data. With further refinement, it has the potential to provide valuable insights into consumer behavior, political sentiment, and more.