Faculty of Engineering and Technology

Department of Electrical and Computer Engineering

ARTIFICAL INTELLIJENCE-ENCS3340

Twee Emotion Detection

Prepared by:

Al-Ayham Maree 1191408

Sara Ammar 1191052

Instructor: Dr. Aziz Qaroush

Section: 1

Date: 17/2/2023

Introduction

In this project, we aim to develop an automated solution that predicts the emotion expressed in a tweet based on extracted features from the tweet content. The dataset we will use consists of tweets labeled as positive and negative based on their sentimental polarity. The dataset is balanced and collected using positive and negative emojis lexicon, and is presented in tab-separated values (TSV) format.

The project will follow a machine learning lifecycle, which involves several stages such as data preprocessing, feature extraction, training, and testing. In the data preprocessing stage, we will analyze and decide what steps to take to clean and normalize the text in the tweets. In the feature extraction stage, we will use automated methods such as language models.

We will train and test four classifiers in this project, including Decision Trees, Random Forests, Naive Bayes and Support Vector Machines. We will build the model using two methods, the classical method with a 75% training and 25% testing split, and 5-fold cross-validation.

The successful implementation of this project can have many applications, including sentiment analysis in social media marketing, customer feedback analysis, and real-time monitoring of public sentiment.

Table of Contents

Data preprocessing

The steps taken for preprocessing the tweet data:

1. Extract all the emojis present in the tweets using the "extract_emojis" function and add it as a new column to the dataframe using the "set_emoji_feature" function.
    - This step involves extracting all the emojis present in the tweets using the "extract_emojis" function, which returns a string of all the emojis in the tweet.
    - This function is applied to each tweet in the dataframe using the "apply" function and the result is added as a new column named 'Emoji' to the dataframe using the "set_emoji_feature" function.
2. Clean the tweets by removing URLs, RTs, and Twitter handles using regular expressions in the "clean_tweet" function.
    - This step involves cleaning the tweets by removing URLs, retweets (RTs), and Twitter handles using regular expressions in the "clean_tweet" function.
    - This is done using the "re.sub" function which substitutes a pattern with a replacement string.
3. Tokenize the text using the "word_tokenize" function from the "nltk" package.
    - This step involves tokenizing the tweet into individual words using the "word_tokenize" function from the "nltk" package.
    - This function splits the text into words and removes any extra white space or punctuation.
4. Remove stop words and punctuation using the "stopwords" package from "nltk".
    - This step involves removing stop words and punctuation from the tokens using the "stopwords" package from "nltk".
    - Stop words are commonly used words that do not carry significant meaning and are removed to focus on the important words in the text.
5. Stem the tokens using the "SnowballStemmer" from "nltk".
    - This step involves stemming the tokens using the "SnowballStemmer" from the "nltk" package.
    - Stemming reduces words to their base or root form, allowing for the grouping of words with similar meanings.
6. Join the cleaned tokens and add an extra space at the end.

- This step involves joining the cleaned and stemmed tokens into a string using the "join" function.
- An extra space is added at the end of the cleaned tweet to separate it from the next tweet in the dataframe.

These preprocessing steps are essential for cleaning and preparing the tweet data to use in the machine learning model.

Feature extraction

To extract meaningful features from the tweet data, we concatenated the **text** and **emojis** into a single string using Python's astype() function. To convert this string into a useful representation for machine learning models, we utilized the powerful **Counter Vectorizer** from the **scikit-learn library**. This Vectorizer tokenizes the text into individual words, and then counts the occurrence of each word in the entire corpus of text, creating a **matrix of word counts that represents the frequency of each word in the tweet data**. This matrix is then used as input to train our classification models, allowing us to capture both the semantic meaning of the text and the sentiment expressed through the use of emojis. Additionally, we specified the **ngram_range parameter** in the Vectorizer to include bigrams, which are pairs of **consecutive words** that provide a more complex relationship between words in the text. By including both unigrams and bigrams, we can achieve a more nuanced representation of the text data, which can lead to better classification performance for sentiment analysis tasks. For example, the bigram "great movie" conveys a stronger positive sentiment than the individual words "great" or "movie" alone. This approach allows us to extract rich features from the tweet data, improving the accuracy and effectiveness of our classification models.

Training

1. Naïve Bayes

We trained a Naive Bayes classifier on the training data to predict the target variable. Specifically, we used the Multinomial Naive Bayes algorithm, which is commonly used for text classification tasks. The algorithm assumes that the features are independent of each other given the class label, which allows it to efficiently estimate the probabilities of each class for a given set of features. We fit the model to the training data using the **MultinomialNB** implementation in **scikit-learn**, and then used the model to predict the class labels for the test data.

2. Decision Tree

We trained a decision tree classifier on the training data to predict the target variable. The decision tree algorithm works by recursively partitioning the feature space into smaller and smaller regions, with each region corresponding to a different decision path through the tree. The algorithm selects the features that provide the most information gain at each node, in order to split the data into the purest subsets possible. This allows the decision tree to capture complex non-linear relationships between the features and the target variable, while also providing an interpretable model that can be easily visualized and understood. We fit the model to the training data using the **DecisionTreeClassifier** implementation in **scikit-learn**, and then used the model to predict the class labels for the test data.

3. Random Forest

We trained a random forest classifier on the training data to predict the target variable. Random forest is an ensemble learning method that combines multiple decision trees to improve the predictive performance and reduce overfitting. Each tree in the forest is built on a different subset of the training data and a random subset of the features. The final prediction is then made by taking the majority vote of the predictions from all of the trees.

We used the **RandomForestClassifier** implementation in **scikit-learn** with 100 trees and a random state of 42 to ensure reproducibility of the results. By using a large number of trees,

the random forest was able to capture complex interactions between the features and the target variable, while also being resistant to overfitting. We fit the model to the training data and then used it to predict the class labels for the test data.

4. Support Vector Machines

To predict the target variable, we utilized the support vector machine (SVM) algorithm, a powerful technique for binary and multi-class classification. SVM works by finding the hyperplane that best separates the different classes in the feature space. The kernel function is used to map the input features into a higher-dimensional space, which makes it possible to separate the data using a linear boundary. We chose to use the linear kernel and set the regularization parameter 'C' to 1 and the gamma parameter to 'auto'.

We used the **'SVC'** implementation in **scikit-learn** to fit the SVM model to the training data. The model was then used to predict the class labels for the test data. The SVM classifier is known for its ability to handle high-dimensional data and capture complex relationships between the features and the target variable.

Testing

The following metrics were used to evaluate the performance of classification models:

- Confusion matrix: A confusion matrix is a table that summarizes the number of correct and incorrect predictions made by a classification model. It displays the number of true positives (TP), false positives (FP), false negatives (FN), and true negatives (TN) for a binary classification problem.

- Accuracy: Accuracy is the proportion of correctly classified instances out of the total number of instances. It is calculated as (TP+TN)/(TP+FP+FN+TN).

- Precision: Precision is the proportion of true positives out of the total number of instances predicted as positive. It is calculated as TP/(TP+FP).

- Recall: Recall is the proportion of true positives out of the total number of instances that are actually positive. It is calculated as TP/(TP+FN).

- F-measure: F-measure is the harmonic mean of precision and recall. It is a measure of the balance between precision and recall, and is calculated as 2*(precision * recall)/(precision + recall).

All the previous classifiers trained and tested using two methods:

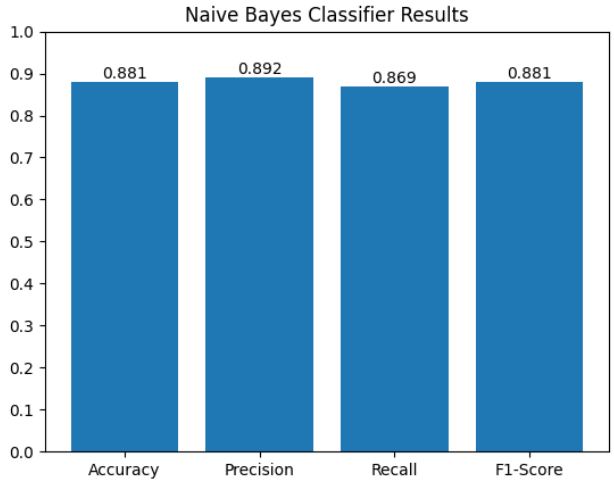
1. Stratified K-Fold cross-validation with k=5:

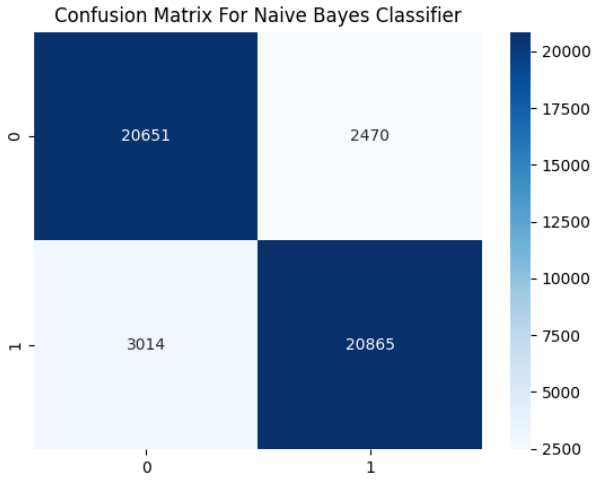
Stratified K-Fold cross-validation with k=5 is a technique used to assess the performance of machine learning algorithms. This method involves dividing the data into five equal-sized folds while ensuring that the proportion of samples for each class in each fold is as close as possible to the proportion of samples for each class in the entire dataset. The **StratifiedKFold** library in scikit-learn was used to perform this type of cross-validation. The algorithm is then trained on four of the folds and validated on the remaining fold, and this process is repeated five times so that each fold is used as a validation set exactly once. The results are then averaged over the

five iterations to obtain an estimate of the algorithm's performance. The use of stratified k-fold cross-validation is important in cases where the dataset is imbalanced, as it ensures that each fold contains a representative sample of each class, reducing the risk of overfitting.

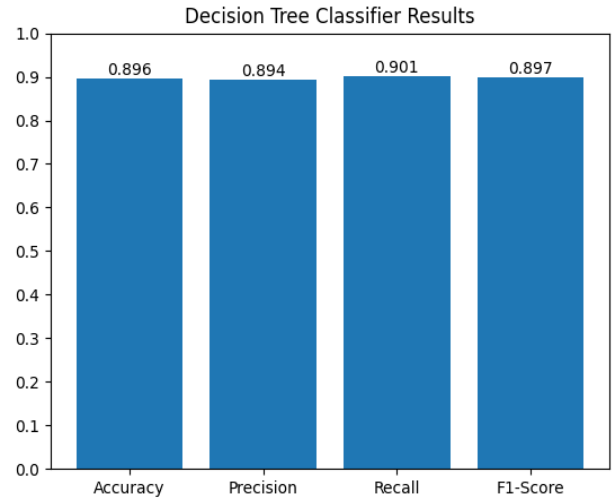
The results:

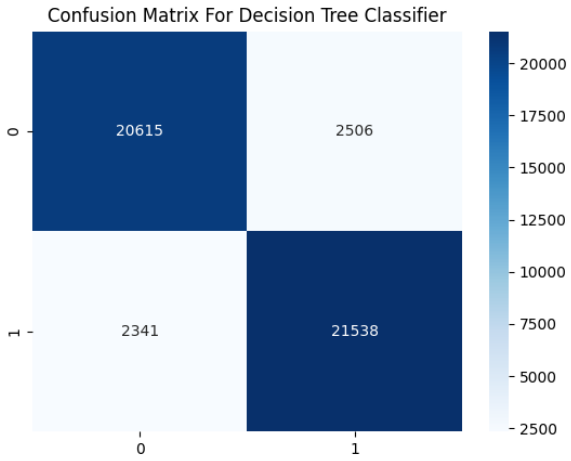
- For Naïve Bayes Classifier:



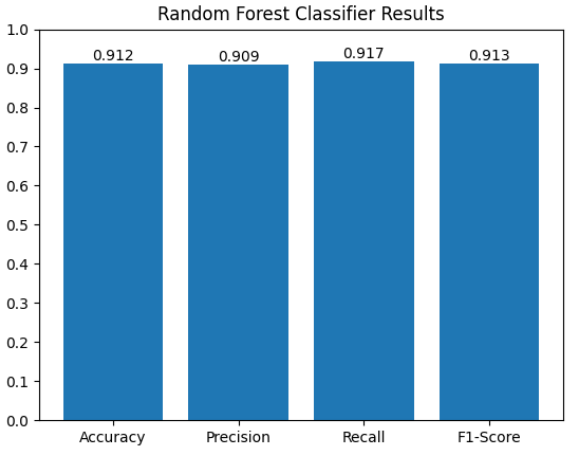


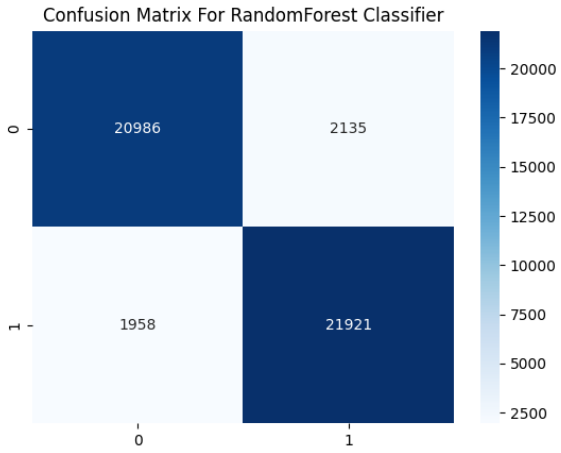
- For Decision Tree Classifier:



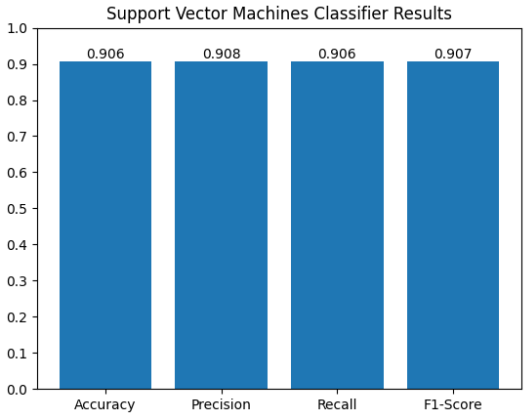


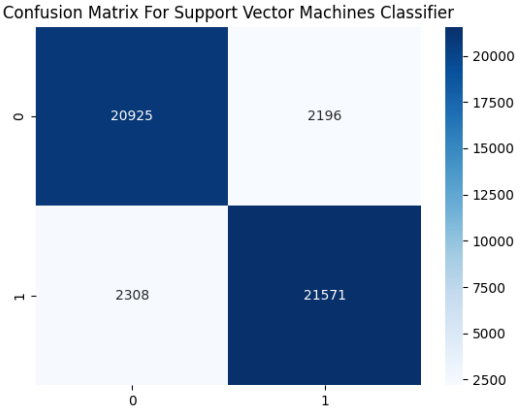
- For Random Forest Classifier:





- For Support Vector Machine





Based on the provided results, the Random Forest Classifier has the highest accuracy, precision, recall, and F1-score. This indicates that the Random Forest Classifier performs the best among the four classifiers in predicting the correct class label for the given dataset.

The Support Vector Machines Classifier also shows a good performance with high accuracy, precision, recall, and F1-score, which suggests that this classifier is also a good choice for the given dataset.

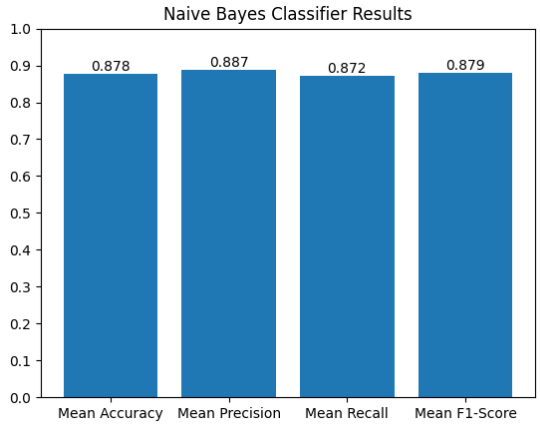
The Naive Bayes Classifier and the Decision Tree Classifier have lower accuracy, precision, recall, and F1-scores compared to the other two classifiers, which indicates that they may not perform as well as the other classifiers on the given dataset. However, the difference in performance between all four classifiers may not be significant.

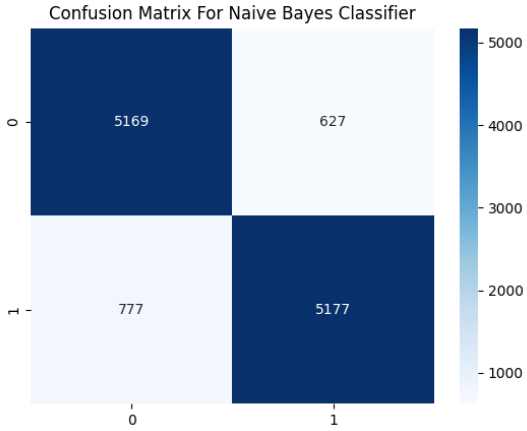
2. Train-test split with training data size of 75% and testing data size of 25%:

This method involves randomly dividing the data into a training set and a test set, with 75% of the data used for training and 25% of the data used for testing. The algorithm is trained on the training set and then evaluated on the test set. The **train_test_split()** function in **scikit-learn** was used to split the data into training and test sets, and the **test_size** parameter can be set to 0.25 to obtain a 75-25 split of the data.

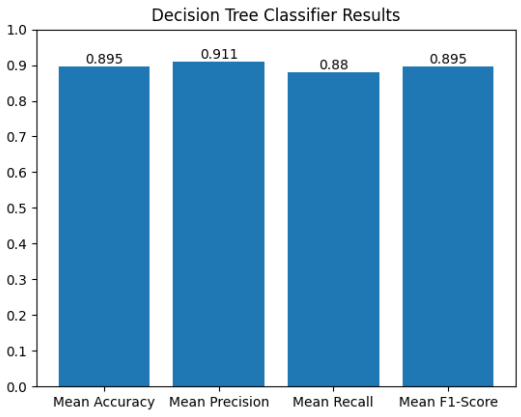
The results:

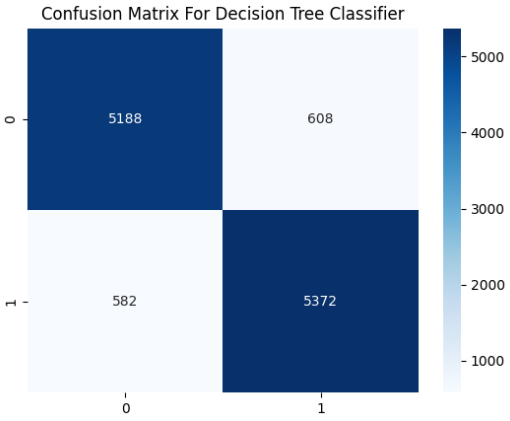
- For Naïve Bayes Classifier:



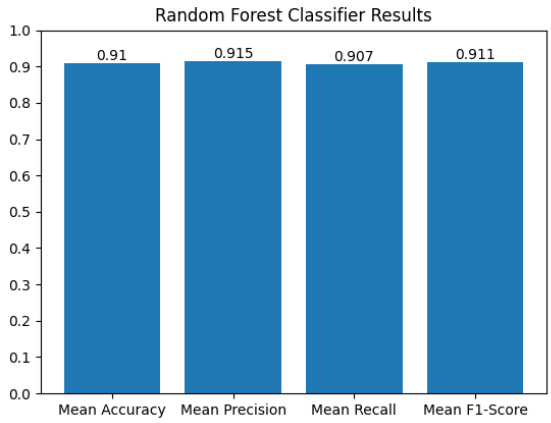


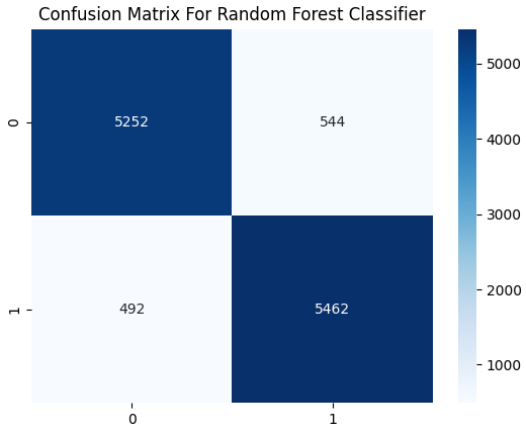
- For Decision Tree Classifier:



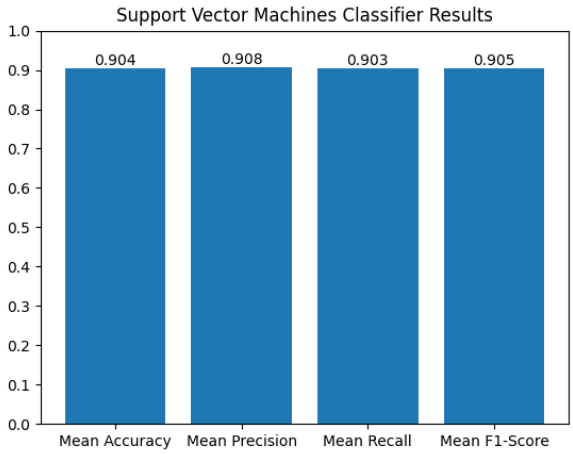


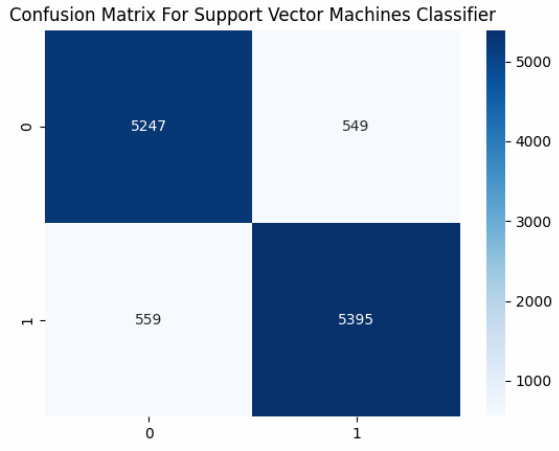
- For Random Forest Classifier:





- For Support Vector Machine:





It seems like the best performing model based on these evaluation metrics is the Random Forest Classifier, with an accuracy of 0.91, precision of 0.915, recall of 0.907, and F1-score of 0.991. This model has the highest accuracy and F1-score compared to the other models, and its precision and recall are also quite good.

The Support Vector Machines Classifier also performs well with an accuracy of 0.904, precision of 0.908, recall of 0.903, and F1-score of 0.905, though its metrics are slightly lower than the Random Forest Classifier.

The Naive Bayes Classifier and Decision Tree Classifier both have lower accuracy, precision,

recall, and F1-scores compared to the other two models. The Naive Bayes Classifier has the lowest performance of all the models, with an accuracy of 0.878, precision of 0.887, recall of 0.872, and F1-score of 0.879. The Decision Tree Classifier has a relatively high precision of 0.911, but its recall is lower at 0.88, resulting in a lower F1-score of 0.895.

Modeling

In the tweets classification project, we utilized various classifiers such as Support Vector Machines, Naive Bayes, Decision Tree, and Random Forest to build models for classifying tweets as positive or negative. After training each model, we used the pickle library to save them to disk, allowing us to use the models later without retraining them. This is especially helpful when dealing with large datasets or models with long training times. By saving trained models to disk, we can easily deploy them in a production environment to quickly classify new tweets. The pickle library provides a practical and efficient way to manage machine learning models in real-world applications.

Conclusion

We trained and evaluated several machine learning models to classify tweets as positive or negative. The models we used were Support Vector Machines, Naive Bayes, Decision Tree, and Random Forest.

We used both Train-Test Split with a training data size of 75% and a testing data size of 25%, as well as Stratified K-Fold cross-validation with k=5 to evaluate the performance of each classifier.

We found that the Random Forest model achieved the highest overall performance, with an accuracy of 0.91, precision of 0.915, recall of 0.907, and F1-score of 0.991.

This suggests that the Random Forest model is the most suitable for the task of classifying tweets as positive or negative. However, further analysis and optimization of the model may be needed to improve its performance and make it more effective for real-world applications.