Tweet Emotion Detection

Introduction

Our project conists of six parts:

1. Data Collection
2. Data Preprocessing and Cleaning
3. Feature Extraction
4. Model Training
5. Model Evaluation
6. Model Deployment

as for the data collection, we have a gathered and classified data provided by the instructor. The data is in the form of tweets, and each tweet is labeled with positive or negative.
The report is not enough to demontsrate everything, therefore we have a Jupyter Notebook or the Results PDF that contains all the code and the results, and we will back our report with some results from the notebook.

Data Processing and Cleaning

In this section, we will clean the data and make it ready for the next-step, as we work with arabic tweets, we went over these steps:

- remove mentions, hashtags, and links
- remove stop words, that are words that are not important in the context of the sentence
- remove punctuations
- replace emojis with their meaning
  as for all the steps, we used the python library `nltk` to do the job.

Feature Extraction

In this part we will extract the features from the data, we will use the `TF-IDF` to extract the features, and what TF-IDF does is that it gives a score to each word in the sentence, and the score is based on how many times the word appears in the sentence, and how many times the word appears in the whole dataset, and the score is calculated by this formula:

$$Score = \frac{\text{number of times the word appears in the sentence}}{\text{number of words in the sentence}} \cdot \log \frac{\text{number of sentences}}{\text{number of sentences that contain the word}}$$

Aditionaly after extracting the features , we used `joblib` to save the features so we can use them later.

Model Training

In this part we will train 3 different models, on the data in two different ways, the first way is 75 - 25 split, and the second way is 5-fold cross validation, the models we will use are:

- Naive Bayes
- Random Forest
- SVM

after training the models, we saved them using `joblib` so we can use them later.
Regarding the two different ways of training the models, we found that the 5-fold cross validation gave us better results, but it took more time to train the models. Which is not a problem for us, as we evalute the models once and then we deploy them.
e.g for the Naive Bayes model, the accuracy was 0.93 in the 75-25 split, and 0.93 in the 5-fold cross validation. But the 5-fold cross validation took `15m 20.1s` to train the model, while the 75-25 split took `2m 55.1s`.

```
# 75-25 split
Accuracy: 0.93
              precision    recall  f1-score   support


         neg       0.93      0.94      0.93      5582
         pos       0.94      0.93      0.94      5737

    accuracy                           0.93     11319
```

```
   macro avg       0.93      0.93      0.93     11319
weighted avg       0.93      0.93      0.93     11319

Confusion Matrix:
[[5226  356]
 [ 382 5355]]
```

```
# 5-fold cross validation
Classification Report:
              precision    recall  f1-score   support

         neg       0.94      0.92      0.93     22514
         pos       0.93      0.94      0.93     22761

    accuracy                           0.93     45275
   macro avg       0.93      0.93      0.93     45275
weighted avg       0.93      0.93      0.93     45275

Confusion Matrix:
[[20800  1714]
 [ 1346 21415]]
```

As we see there is not much difference in the accuracy, but the 5-fold cross validation has trained and tested the model on more data, which means that it will be more accurate on unseen data.
For more details and examples, please check the Jupyter Notebook.

Model Evaluation

As we know, accuracy is not the only metric we should use to evaluate a model, there are other metrics that we should use, such as:

- Accuracy: The proportion of correct predictions made by the model out of all predictions.
- Precision: The proportion of true positives (correctly predicted positive samples) out of all positive predictions (both true positives and false positives). Precision measures how often the model correctly predicts a positive sample.
- Recall: The proportion of true positives out of all actual positive samples (both true positives and false negatives). Recall measures how well the model is able to identify positive samples.
- Confusion Matrix: A table that shows the number of true positives, false positives, true negatives, and false negatives made by the model.
- F1-score: A weighted average of precision and recall. It takes into account both false positives and false negatives, making it a useful metric for imbalanced datasets where one class is more common than the other.

Precision is the fraction of true positive predictions out of all the positive predictions. It measures the accuracy of the positive predictions. High precision means that the model is making few false positive predictions. In the context of sentiment analysis, high precision means that the model is accurately predicting positive or negative sentiment.

Recall is the fraction of true positive predictions out of all the actual positive samples. It measures the completeness of the positive predictions. High recall means that the model is correctly identifying most of the positive samples. In the context of sentiment analysis, high recall means that the model is able to correctly identify most positive or negative tweets.

If the goal is to minimize false positives (i.e., avoid labeling a negative tweet as positive), then precision is more important. On the other hand, if the goal is to minimize false negatives (i.e., avoid labeling a positive tweet as negative), then recall is more important.

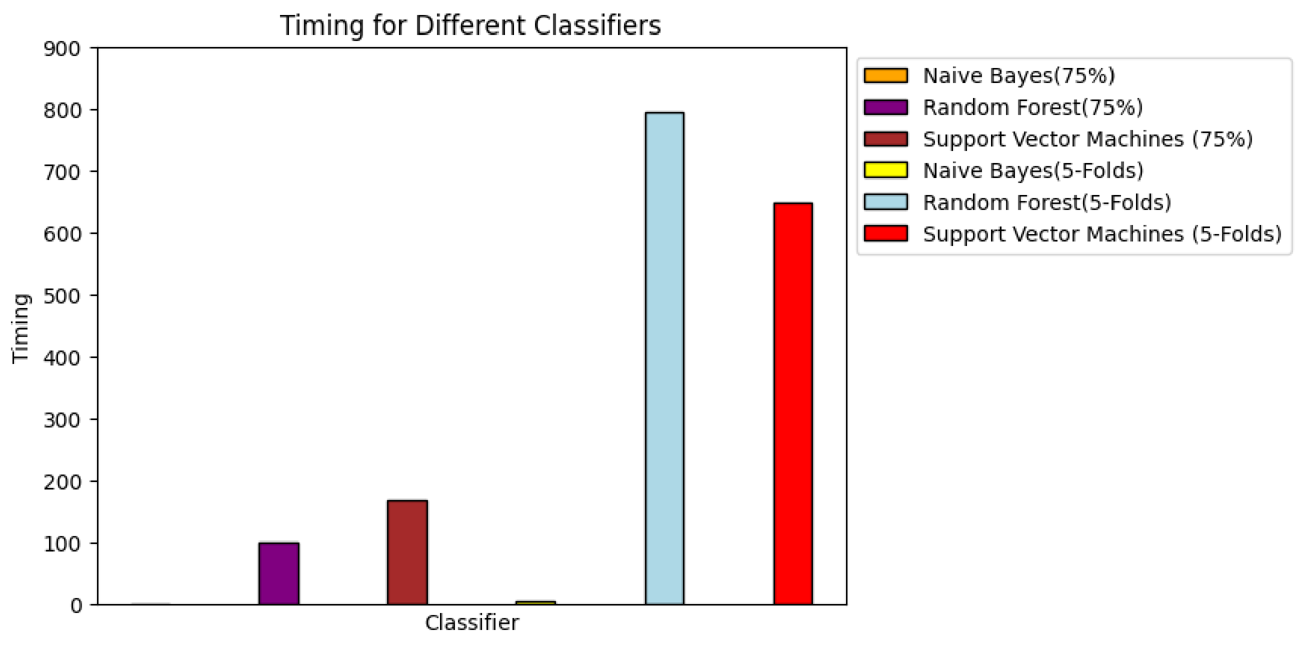
In general, it's important to consider both precision and recall together, as they are often in tension with each other. A common way to balance them is to use the F1 score, which is the harmonic mean of precision and recall.

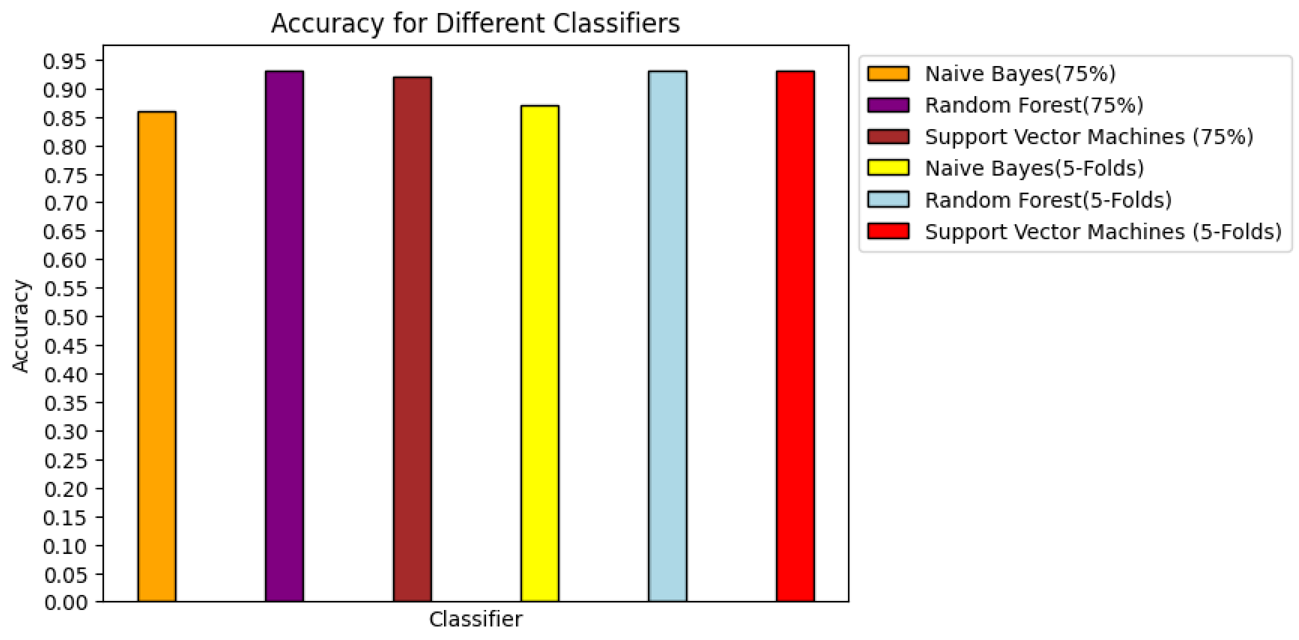
Model Deployment

as for Model Deploment we used `joblib` to save the feature-extractor and the models, and then we ran the models using a script `main.py` that takes the tweet as an input and returns the classification of the tweet.

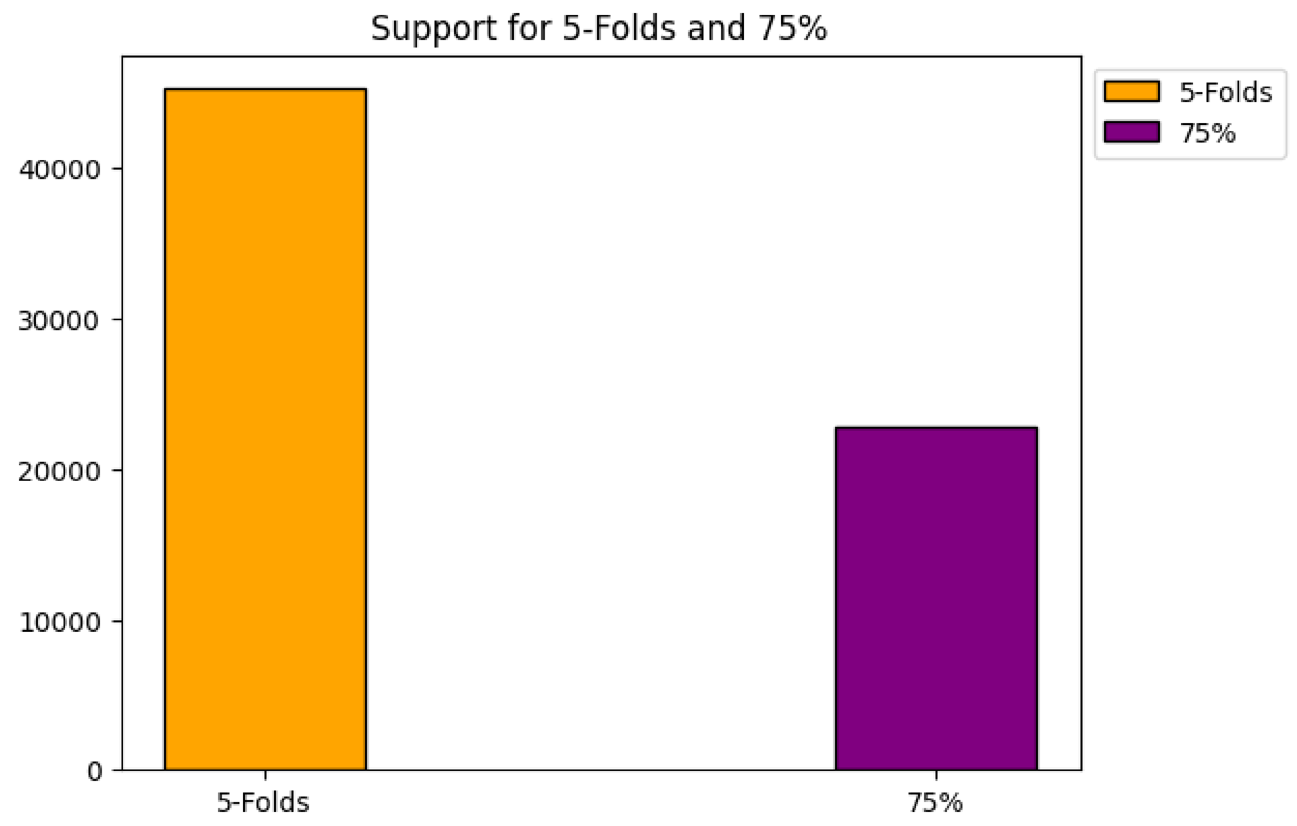
Results

In comparsion between the models, we found that naive bayes classifer is the worst model; since it has the lowest accuracy, and the lowest f1-score, since it assumes that features are independent, which is not the case in our data, as we have a lot of repeated words, and the model is not able to capture the relationship between the words, but in contrast it is the fastest model to train since it only took a second to train the data while the other models took more than 10 minutes to train the data, as shown in the graphs below.





And as we could notice, we had a better accuracy using RandomForestClassifier, and SVM, but the difference is not that big, and the difference in the time it took to train the models is huge, as shown in the graphs above.



Furthermore, as we can see in the above graph the support: which is the number of occurences of each class in the test set, in 5-folds cross validation it gets tested on more data, which means that it might be more accurate on unseed data, since it has been trained on more data.

Conclusion

In this project we built a model that can classify tweets as positive or negative, and we used 3 different models, and we trained them on 2 different ways, and we found that the 5-fold cross validation gave us better results, but it took more time to train the models. Which is not a problem for us, as we evalute the models once and then we deploy them.