TATIA Report

Sentiment Analysis of Arabic Tweets

Abstract: This project aims to develop a sentiment analysis predictor for Arabic tweets, with a focus on identifying negative or positive sentiments. Different analysis techniques are explored, including various word vectorizers, classifiers, and models. The goal is to compare these different methods and determine their effectiveness in handling the challenges of the Arabic language. One notable technique explored is word embedding, which represents words in a vector space and utilizes the distance between them for sentiment analysis. Through this project, we aim to gain a better understanding of the challenges and potential solutions for sentiment analysis in the Arabic language.

Keywords: Sentiment Analysis, Arabic tweets, Word Vectorizers, Classifiers, Word Embedding.

Introduction: Sentiment analysis have the been the subject of different studies since the beginning of Artificial Intelligence and classification models, It is a fundamental part of Natural Language Processing, It has many use-cases and could help businesses, workers and AI enthusiasts improve their lives.

Note: A more detailed and interactive version of the project is available in the form of a notebook. The results can be reproduced on `Kaggle`. Comments and descriptions have been added to the notebook.

Link to our Kaggle notebook: https://www.kaggle.com/code/fayssalelansari/arabic-tweets-sentiment-analysis

The notebook is linked to this GitHub Repo: https://github.com/fayssalElAnsari/tatia-project

When the Kaggle notebook is saved, the modifications are pushed to the GitHub Repository automatically.

2. Analysis Steps:

a. Loading Dataset

We start off by deciding which a dataset, we have the option between choosing an already classified data (which we used in this project) or we could fetch a collection of tweets by hashtag or keyword using the official Twitter API. We will use the Twitter API later on to test our model that was trained with this dataset: https://www.kaggle.com/datasets/mksaad/arabic-sentiment-twitter-corpus, if the data isn't already split into training and testing groups we will need to split it.

b. Normalization

This is a necessary step in order for our program to be able to make sense of the data given to it. This step has three main sub-steps:

b.1. Pre-processing

This could be a combination of different methods: removing stop words, removing or decoding emojis, removing punctuation and/or removing special characters.

b.2. Tokenization

This step is only required if we're building our tools from scratch since most famous vectorizers take in a text as is, then automatically tokenize it into different features.

b.3. Extracting root

Could be Lemmatization or a simple Stemming. It could help lower the total number of features. And better connect different tokens by combining them into one root token.

c. Vectorization

This is the main goal of the previous step, after this step the program will be able to better understand different features and the relationships between them by turning each token into a vector. There are many available vectorizers which use different methods (bag of words, frequency, embeddings, neural networks…). In this project we will try a collection of these methods then compare the results.

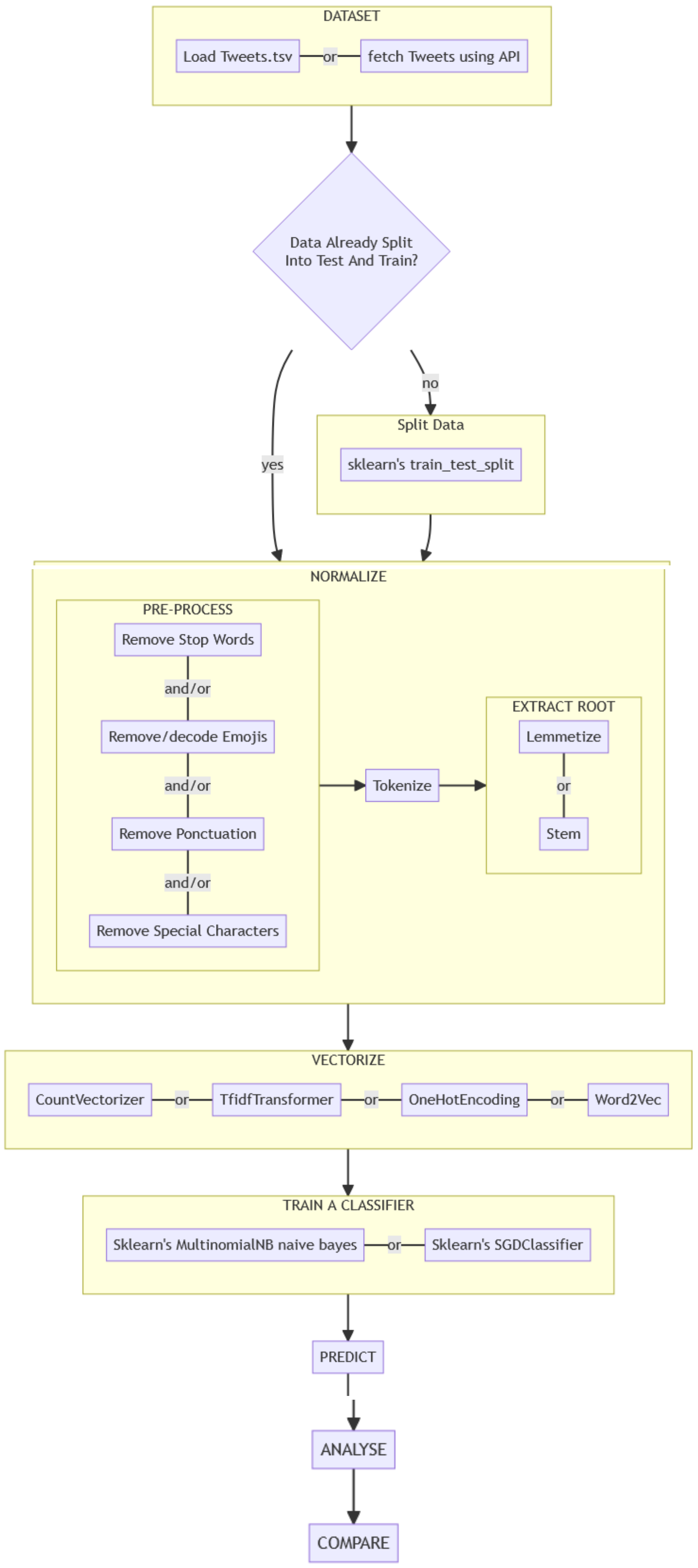
d. Training

This is the last heavy step, in this project we've used many classifiers and compared the results of each one, the best classifier for our dataset was Facebook's fastext produced a precision score of 95%, followed by sklearn's SGDClassifier which gave us a 92% precision score. For SGDClassifier we used it only to classify, all the previous steps were implemented by us.

e. Analyzing & comparing

The final step is Analyzing and Comparing different used methods with different parameters to produced better prediction results.

2.1. Diagram:


DATASET
Load Tweets.tsv
or
fetch Tweets using API
Data Already Split Into Test And Train?
yes
no
Split Data
sklearn's train_test_split
NORMALIZE
PRE-PROCESS
Remove Stop Words
and/or
Remove/decode Emojis
and/or
Remove Ponctuation
and/or
Remove Special Characters
Tokenize
EXTRACT ROOT
Lemmetize
or
Stem
VECTORIZE
CountVectorizer
or
TfidfTransformer
or
OneHotEncoding
or
Word2Vec
TRAIN A CLASSIFIER
Sklearn's MultinomialNB naive bayes
or
Sklearn's SGDClassifier
PREDICT
ANALYSE
COMPARE

CountVectorizer

Here we'll convert our collection of tweets to a matrix of token counts, this method takes in many parameters which could make the previous steps unnecessary, it could take in a list of stop words, a tokenizer , preprocessor and much more.

This implementation produces a sparse representation of the counts using scipy.sparse.csr_matrix. source

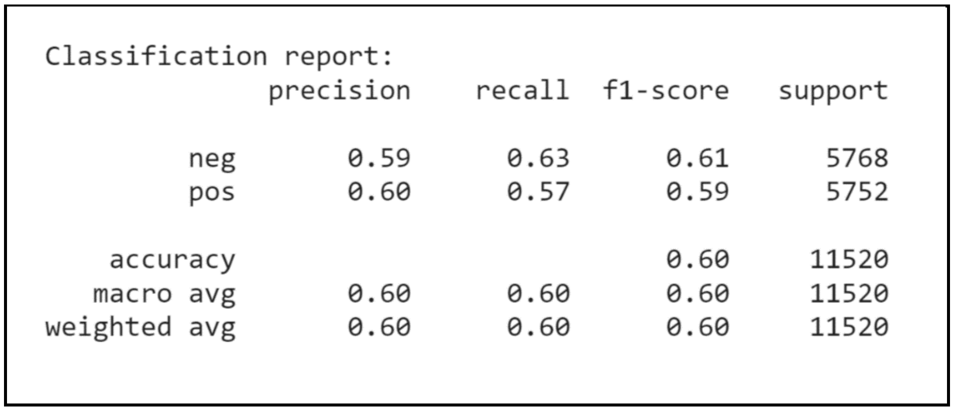
A **sparce matrix** is used to store matrices with a lot of zero values in a way to not take a lot of space and make it easier to go through the whole matrix in a short period of time.

NOTE: we didn’t use the tokenized text, only the pre-processed directly with vectorizers.

One hot encoding:

One hot encoding is a method of representing text data as numerical data that can be used as input for machine learning models. In the context of our project, this involves converting each unique word in the dataset into a binary vector of the same length as the vocabulary size. The vector is filled with zeros, except for the index corresponding to the word, which is set to 1. This method can be used for sentiment prediction by training a classifier on these encoded vectors, with the sentiment (positive or negative) as the target variable.

We can use this method to train a classifier, and then evaluate its performance using the classification report and confusion matrix shown below.



```
Classification report:
              precision    recall  f1-score   support

         neg       0.59      0.63      0.61      5768
         pos       0.60      0.57      0.59      5752

    accuracy                           0.60     11520
   macro avg       0.60      0.60      0.60     11520
weighted avg       0.60      0.60      0.60     11520
```

Figure 1: Classification report for one hot encoding method

The classification report provides metrics such as precision, recall and F1-score, which give an idea of the model's accuracy and ability to identify positive and negative sentiments.

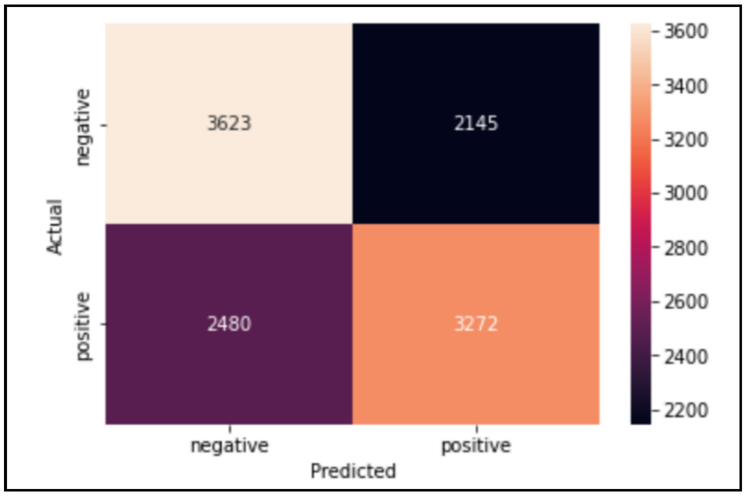


Figure 2: Confusion matrix for one-hot encoding method

The confusion matrix provides a more detailed breakdown of the model's performance by displaying the number of true positives, true negatives, false positives and false negatives.

Limitations :

One hot encoding is commonly used in natural language processing tasks such as sentiment analysis because it allows words to be represented as numerical values that can be input into machine learning models. However, it has several limitations when used for sentiment analysis of Arabic tweets.

Firstly, one hot encoding does not take into account the order or context of the words in the text, which can be important for understanding the sentiment of a tweet.

Another limitation of one hot encoding is that it is not able to handle unknown words or out-of-vocabulary words. These are words that are not present in the vocabulary used to train the model, but may appear in the test data. The model will not be able to predict the sentiment of tweets that contain unknown words, which can lead to a decrease in accuracy.

Conclusions :

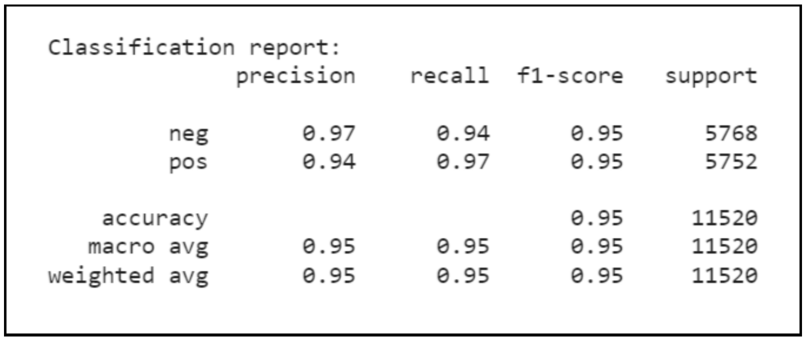
One-hot encoding is not suitable for sentiment analysis of Arabic tweets, due to the inflected nature of the language, difficulty in handling diacritics, and large number of unique words.

One hot encoding and FFNNs are not recommended for this task, instead other methods such as word embeddings or predefined models with support for Arabic language should be used.

Fasttext:

FastText is a text classification library developed by Facebook that can be used for sentiment analysis and other natural language processing tasks.
We used fasttext to train a supervised classifier. The classifier was trained using the training data, and then tested on the testing data to evaluate its performance.

After training and testing the classifier, we analyzed the classification report and the confusion matrix as shown below.



```
Classification report:
              precision    recall  f1-score   support

         neg       0.97      0.94      0.95      5768
         pos       0.94      0.97      0.95      5752

    accuracy                           0.95     11520
   macro avg       0.95      0.95      0.95     11520
weighted avg       0.95      0.95      0.95     11520
```

Figure 3: Classification report for Fasttext

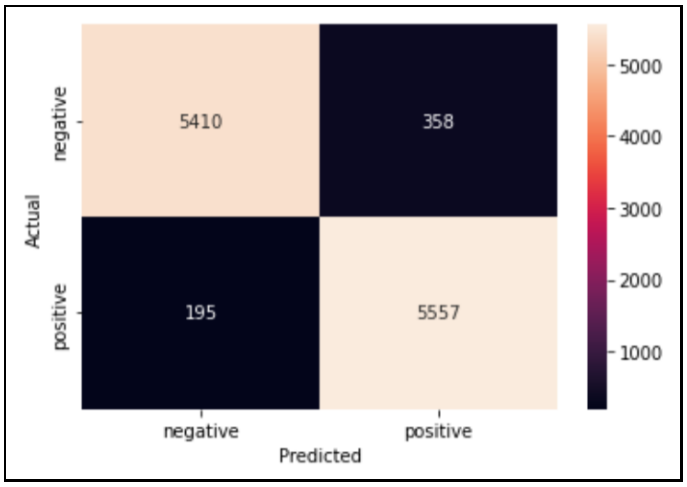


Figure 4: Confusion matrix for Fasttext

The results of our analysis showed that the classifier achieved a high precision score. This means that the classifier was able to accurately identify the positive/negative tweets in the testing data.

The reason for this is that fasttext is a powerful library that uses a combination of subword information and character n-grams to create word representations. This allows it to effectively handle the complexity of the Arabic language, which has many variations and dialects.

Limitations:

One limitation is that FastText requires a large amount of data to train its model effectively. This can be a challenge when working with a relatively small dataset, such as a collection of Arabic tweets.

Another limitation is that FastText may not be able to handle misspelt words and typos, which are common in informal text such as tweets, and we were able to experience this limitation while working with it.

Conclusions :

FastText is a library for text classification and representation learning that is particularly well-suited for handling the complexity of natural language.
One of the reasons that it can achieve high accuracy scores for sentiment analysis of Arabic tweets is its ability to take into account subword information. This is particularly beneficial for Arabic, as it is an inflected language with a large number of unique words.

Word embeddings:

Word embeddings are dense vector representations of words that capture their meaning and context. These embeddings can be used to represent words in a vector space, where similar words will be close together and dissimilar words will be far apart.

In our project, we used FastText to create word embeddings for Arabic text. We first trained the embeddings on a large dataset of Arabic tweets, and then used the embeddings to represent individual words in a vector space like the following figure.

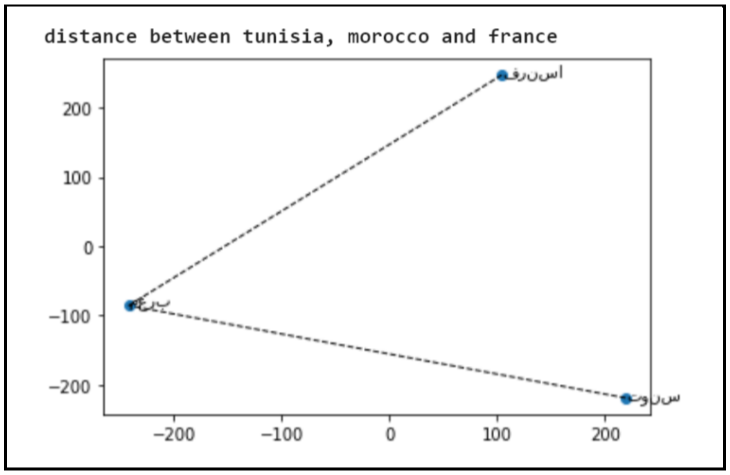
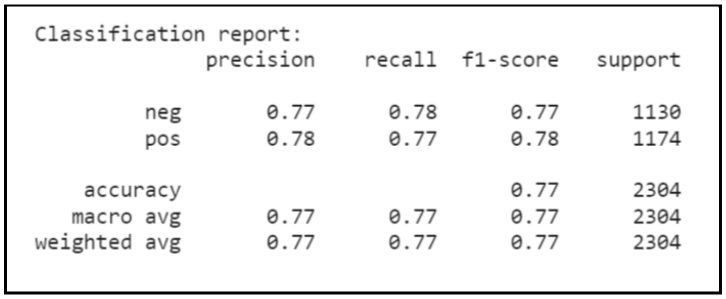


Figure 5: Vectorial representation for the distance between France, Tunisia and Morocco

We tested the embeddings by finding the nearest neighbors of random words and found that they were semantically similar.

After creating the embeddings, we used them as input for a sentiment predictor. We trained a classifier using the embeddings as input, and tested it on a separate dataset of tweets. We found that the classifier had a good precision score, which means that it was able to correctly identify positive and negative tweets.



```
Classification report:
              precision    recall  f1-score   support

         neg       0.77      0.78      0.77      1130
         pos       0.78      0.77      0.78      1174

    accuracy                           0.77      2304
   macro avg       0.77      0.77      0.77      2304
weighted avg       0.77      0.77      0.77      2304
```

Figure 6: Classification report for word embedding based sentiment predictor

One possible reason for the relatively good performance of the classifier is that the embeddings were able to capture the meaning and context of the words in the text, which made it easier for the classifier to distinguish between positive and negative tweets.

Limitations:

However, the limitations of using fasttext for this project is that the embeddings are not perfect, and the embeddings may not capture all the nuances of meaning and context in the text.

Additionally, the embeddings are based on the training data, which means that if the training data is biased in some way, the embeddings will also be biased.

Finally, the embeddings are not context-aware, they are only based on the words and its co-occurrences in the training data.

Conclusions:

In conclusion, using fasttext for word embedding proved to be an effective method for sentiment analysis of Arabic tweets. By representing words in a vectorial space, we were able to see the distance between words and use that information to train a sentiment classifier. However, it is important to note that this method is not without limitations.

Other ideas to implement

1. webapp:
   - make a front end connected to a trained model or give the user the ability to train his own model.
   - in the webpage there will be a way to fetch a predetermined number of tweets using a keyword or a hashtag (should be in Arabic).
2. Since for the same step there could be different methods and for each method different methods add an easier way to change the method and the parameters, could be modified in the web page also.
3. show graphs for different stats of the chosen models and methods
4. a way to run all possible (or a lot) methods and parameters and order them by decreasing order of validity
5. add progress bar when creating and modifying DataFrames. [done]

Definitions of Common Terms

Normalizing: Also called text cleansing, is the process of preprocessing text data to be understood and used by Natural Language Processors and other text analytics software. It includes many other substeps such as tokenizing, case conversion, correcting spelling, removing stop words, stemming and lemmatization.

Tokenization: It is essentially splitting a phrase, sentence, paragraph, or an entire text document into smaller units, such as individual words or terms. Each of these smaller units are called tokens. Usually we split a text document by word to obtain a token for each individual word in a list of tokens.

Stop words: A list of the most common words in a language

Vectorization: The process of writing tokens of text data in the form of vectors, since computer programs are only able to understand numbers it is important to vectorize text data in order to process it. Also called Feature extraction; There are many feature-extraction methods the most famous being bag of words based frequency features.

Feature Matrix: In this matrix each row represents a document (here tweet) and each column represents a feature (processed token). It is used by ML and statistical models to provide predictions.

Stemming: Consists of extracting only the smallest incompressible part of a word not containing any suffixes or prefixes or any other inflections. Then converting it to the root or the source of the meaning of that word.

lemmatization: Similar to Stemming is the same process of removing inflections from a feature (token) but without converting it to its origin of meaning. It is less representative for making prediction since more words can have the same stem than the the words having the same lemma.

Bag of Words Model: One of the most popular yet simple Feature Extraction methods, for each document we build a set of words without any regard of the order of presence (hence the name bag of words). But keeping count for each word.

Confusion Matrix: lso known as an error matrix, is a specific table layout that allows visualization of the performance of an algorithm, typically a supervised learning one (in unsupervised learning it is usually called a matching matrix). Each row of the matrix represents the instances in an actual class while each column represents the instances in a predicted class, or vice versa – both variants are found in the literature. The name stems from the fact that it makes it easy to see whether the system is confusing two classes (i.e. commonly mislabeling one as another).