Emotion Detection in Arabic Text Using Classical and Deep Learning Techniques

By:

Abdulrahman Abuhani ~ 2132462
Belal Khaled ~ 2136873
Yaseen Naser ~ 2130397

—

Dr. Esra'a Alshdaifat

—

NLP and Text Mining

—

Assignment 04

INTRODUCTION

This report presents a comprehensive pipeline for processing and analyzing Arabic text in the context of emotional tone classification. It includes detailed preprocessing steps such as cleaning, tokenization, normalization, and feature extraction using Bag-of-Words, TF-IDF, and Word2Vec models. A comparative analysis of model performance is provided using metrics including accuracy, precision, recall, and F1-score. The report also highlights the unique challenges associated with Arabic language processing, such as handling diacritics, word variations, and the scarcity of high-quality pre-trained embeddings.

Preprocessing Details

All necessary libraries were imported

```
import pandas as pd
import re
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from gensim.models import KeyedVectors
from gensim.models import Word2Vec
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.metrics import accuracy_score, classification_report
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Embedding, LSTM, Bidirectional
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
```

Load and Explore the Dataset

1. Data Loading

- **The dataset was loaded from a CSV file:**

```
data = pd.read_csv("Emotional-Tone-Dataset.csv")
```

2. Dataset Exploration

- **Displayed first few records: data.head() ➔**

	ID	TWEET	LABEL
0	1	.. الاوليمبياد الجايه هكون لسه ف الكليه	none
1	2	عجز الموازنه وصل ل93.7 % من الناتج المحلي يعني...	anger
2	3	xD كنتا نيله ف حظنا الهباب	sadness
3	4	جميعنا نريد تحقيق اهدافنا لكن تونس تالقت في حر...	joy
4	5	... الاوليمبياد نظامها مختلف .. ومواعيد المونديال	none

- **Checked data types and nulls: data.info() ➔**

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10065 entries, 0 to 10064
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   ID      10065 non-null  int64 
 1    TWEET  10064 non-null  object
 2    LABEL  10065 non-null  object
dtypes: int64(1), object(2)
memory usage: 236.0+ KB
```

- **Summary statistics: data.describe() ➔**

	ID
count	10065.000000
mean	5033.000000
std	2905.659564
min	1.000000
25%	2517.000000
50%	5033.000000
75%	7549.000000
max	10065.000000

- **Cleaned column names by stripping Whitespace ➔**

```
data.columns
```

```
Index(['ID', ' TWEET', ' LABEL'], dtype='object')
```

```
data.columns = data.columns.str.strip()
data.columns
```

```
Index(['ID', 'TWEET', 'LABEL'], dtype='object')
```

Text Cleaning

- **The text cleaning function performs the following:**
  - **Remove unwanted characters: punctuation, numbers, and non-Arabic letters.**
  - **Remove diacritics: Arabic short vowels to normalize the text.**
  - **Remove stop words: such as "على", "في", "و".**

```python
def clean_text(text):
    #Check If Text is Not a String:
    if not isinstance(text, str):
        return ""
    #Removing Punctuation and Numbers:
    text = re.sub(r'[^\u0600-\u06FF\s]', '', text)
    #Removing Diacritics:
    text = re.sub(r'[\u064B-\u065F]', '', text)
    #Removing Stop Words:
    arabic_stopwords = set(stopwords.words('arabic'))
    text = ' '.join([word for word in text.split() if word not in arabic_stopwords])
    return text
```

  - **Applied to TWEET column:**

```python
#Applying the clean_text Function to the 'TWEET' Column (Adding New Column 'cleaned_TWEET'):
data['cleaned_TWEET'] = data['TWEET'].apply(clean_text)

#Displaying 'TWEET' Column and 'cleaned_TWEET' Column:
data[["TWEET", "cleaned_TWEET"]].head()
```

	TWEET	cleaned_TWEET
0	الاوليمبياد الجايه هكون لسه ف الكليه ..	الاوليمبياد الجايه هكون لسه الكليه
1	عجز الموازنه وصل الناتج المحلي يعني لسه اقل من 93.7 % ل...	عجز الموازنه وصل الناتج المحلي يعني لسه اقل نف...
2	كتنا نيله ف حظنا الهباب xD	كتنا نيله حظنا الهباب
3	جميعنا نريد تحقيق اهدافنا لكن تونس تالقت في حر...	جميعنا نريد تحقيق اهدافنا تونس تالقت حراسه المرمي
4	الاوليمبياد نظامها مختلف ومواعيد المونديال	الاوليمبياد نظامها مختلف ومواعيد المونديال مكا...

Tokenization

- **Tokenized the cleaned_TWEET column using word_tokenize:**

```python
#Tokenizing the 'cleaned_TWEET' Column (Adding New Column 'tokenized_TWEET'):
data['tokenized_TWEET'] = data['cleaned_TWEET'].apply(word_tokenize)

#Displaying 'cleaned_TWEET' Column and 'tokenized_TWEET' Column:
data[["cleaned_TWEET", "tokenized_TWEET"]].head()
```

	cleaned_TWEET	tokenized_TWEET
0	الاوليمبياد الجايه هكون لسه الكليه	[الاوليمبياد, الجايه, هكون, لسه, الكليه]
1	عجز الموازنه وصل الناتج المحلي يعني لسه اقل نف...	[عجز, الموازنه, وصل, الناتج, المحلي, يعني, لسه]...
2	كتنا نيله حظنا الهباب	[كتنا, نيله, حظنا, الهباب]
3	جميعنا نريد تحقيق اهدافنا تونس تالقت حراسه المرمي	[جميعنا, نريد, تحقيق, اهدافنا, تونس, تالقت, حر]...
4	الاوليمبياد نظامها مختلف ومواعيد المونديال مكا...	[الاوليمبياد, نظامها, مختلف, ومواعيد, المونديا]...

Text Normalization

- **Performed the following normalizations:**
  - Normalize "ا" → "ا" ,"آ" ,"إ" ,"أ"
  - Convert "ة" → "ه"
  - Remove repeated characters (e.g., "رائع" → "راااائع")

```
def normalize_text(text):
    text = re.sub(r'[إأآا]', 'ا', text)
    text = re.sub(r'ه' ,'ة', text)
    text = re.sub(r'(.)\1+', r'\1', text)
    return text
```

  - Applied normalization to tokenized_TWEET column:

```
#Applying Normalization to the 'tokenized_TWEET' Column (Adding New Column 'normalized_TWEET'):
data['normalized_TWEET'] = data['tokenized_TWEET'].apply(lambda x: [normalize_text(word) for word in x])

#Displaying 'tokenized_TWEET' Column and 'normalized_TWEET' Column:
data[["tokenized_TWEET", "normalized_TWEET"]].head()
```

	tokenized_TWEET	normalized_TWEET
0	[الاوليمبياد, الجايه, هكون, لسه, الكليه]	[الاوليمبياد, الجايه, هكون, لسه, الكليه]
1	[عجز, الموازنه, وصل, التاتج, المحلي, يعني, لسه...]	[عجز, الموازنه, وصل, التاتج, المحلي, يعني, لسه...]
2	[كنتا, نيله, حظنا, الهباب]	[كنتا, نيله, حظنا, الهباب]
3	[جميعنا, نريد, تحقيق, اهدافنا, تونس, تالقت, حر...]	[جميعنا, نريد, تحقيق, اهدافنا, تونس, تالقت, حر...]
4	[الاوليمبياد, نظامها, مختلف, ومواعيد, المونديا...]	[الاوليمبياد, نظامها, مختلف, ومواعيد, المونديا...]

Feature Extraction Techniques

- **Bag-of-Words (BoW):**
  - Used CountVectorizer with top 5000 features

```
#Joining the Tokenized Words Back Into Sentences:
data['normalized_TWEET'] = data['normalized_TWEET'].apply(lambda x: ' '.join(x))

#Initializing CountVectorizer:
BOW_vectorizer = CountVectorizer(max_features=5000)  # Limit to top 5000 features

#Fitting and Transforming the Text Data:
BOW_features = BOW_vectorizer.fit_transform(data['normalized_TWEET'])

#Converting to DataFrame for Better Readability:
BOW_df = pd.DataFrame(BOW_features.toarray(), columns=BOW_vectorizer.get_feature_names_out())

BOW_df.head()
```

	اب	ابتديت	ابتسامتك	ابتسامه	ابتسم	ابحث	ابدا	ابداع	ابراهيم	ابرز	...	يوفقه	يوفقهم	يول	يوم	يوما	يومه	يومها	يومين	يونس	گل
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

5 rows × 5000 columns

- **TF-IDF:**
  - **Used TfidfVectorizer with top 5000 features:**

```
#Initializing TfidfVectorizer:
TFIDF_vectorizer = TfidfVectorizer(max_features=5000)  # Limit to top 5000 features

#Fitting and Transforming the Text Data:
TFIDF_features = TFIDF_vectorizer.fit_transform(data['normalized_TWEET'])

# Convert to DataFrame for better readability
TFIDF_df = pd.DataFrame(TFIDF_features.toarray(), columns=TFIDF_vectorizer.get_feature_names_out())

TFIDF_df.head()
```

	اب	ابتديت	ابتسامتك	ابتسامه	ابتسم	ابحث	ابدا	ابداع	ابراهيم	ابرز	...	يوفقه	يوفقهم	يول	يوم	يوما	يومه	يومها	يومين	يونس	گل
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 5000 columns

- **Word2Vec:**
  - **Loaded a pretrained model, AraVec 1.0 (Twitter-SG) using Word2Vec:**

```
# • AraVec Used Here:
try:
    #Loading the Model Using Word2Vec:
    word2vec_model = Word2Vec.load(
        r'C:\Users\BKSH\OneDrive - Hashemite University\Desktop\BKSH HU\Natural Language Processing\Assignments\Assignment 4\Models\Twt-SG'
    )
    print("Model loaded successfully!")
except Exception as e:
    print("Error loading model:", e)

def average_word2vec(tokens, model):
    #Filtering Out Words Not in the Vocabulary:
    vectors = [model.wv[word] for word in tokens if word in model.wv]
    if len(vectors) == 0:
        return np.zeros(model.wv.vector_size)
    return np.mean(vectors, axis=0)

#Applying the Function to Get Sentence Embeddings for Each Row:
data['sentence_embeddings'] = data['normalized_TWEET'].apply(
    lambda x: average_word2vec(x.split(), word2vec_model)
)

# • Example: ○ Accessing the Vector for the First Sentence:
data['sentence_embeddings'].iloc[0]
```

```
Model loaded successfully!
array([-9.59469937e-03,  1.57624990e-01, -2.03452021e-01,  1.17573902e-01,
        1.06548741e-01, -9.86032709e-02, -6.60574958e-02,  3.56602371e-02,
       -9.03370902e-02,  7.30422288e-02, -1.44537613e-01,  2.68261373e-01,
        5.66276796e-02,  3.77053432e-02,  6.25307709e-02,  6.56563193e-02,
        2.02920884e-01,  2.83338696e-01,  1.41982645e-01,  2.84037173e-01,
       -4.19083744e-01,  4.45718281e-02,  1.37060240e-01, -2.91097467e-03,
       -1.24953575e-02,  3.20910923e-02,  1.06452722e-02,  2.42769405e-01,
        1.54280469e-01, -1.27435476e-01, -1.04157425e-01,  1.28446892e-01,
        1.70553580e-01, -5.80303837e-03,  2.42562100e-01, -1.28271906e-02,
        1.57604814e-01, -1.53687134e-01,  4.56495345e-01, -8.34420044e-03,
        2.05491967e-02,  1.15231551e-01,  4.25996967e-02, -2.58710589e-02,
       -1.07627556e-01, -9.87990573e-02,  1.21960461e-01, -3.84536397e-04,
        1.10248491e-01,  9.21784788e-02,  2.91089956e-02,  2.40413785e-01,
        8.10044259e-02,  6.80156425e-02,  5.41392388e-03, -3.04839075e-01,
       -2.40030885e-01, -3.79416570e-02,  8.53282958e-02, -2.13600487e-01,
       -1.35766476e-01, -1.08155444e-01, -3.16930190e-02,  4.93715517e-02,
       -8.97521153e-02, -1.00637257e-01, -2.20862478e-01,  5.44167049e-02,
       -4.79671347e-04,  5.37967496e-02,  8.90884101e-02,  2.49592513e-01,
        1.84567526e-01,  2.87692845e-02,  3.41634274e-01,  7.72554576e-02,
        1.28891140e-01,  2.41052777e-01, -4.15517576e-02,  6.25494123e-02,
       -1.85463876e-02,  1.97138280e-01, -9.48513970e-02, -1.31227281e-02,
       -1.89216599e-01, -1.39030635e-01,  5.66492006e-02, -9.29911211e-02,
       -6.82952181e-02,  3.48320633e-01,  8.52513537e-02, -9.17389393e-02,
        1.08714782e-01, -2.10748509e-01, -2.02217907e-01, -4.16470841e-02,
        1.14398815e-01, -2.08871067e-02, -1.99447438e-01,  7.86841884e-02,
       -1.24333881e-01,  2.57603377e-01,  1.65852904e-01, -8.10827166e-02,
        3.82111296e-02,  6.71163574e-02, -2.79677600e-01,  8.09236020e-02,
        2.08997652e-01,  2.87326910e-02, -1.39246687e-01, -4.93337736e-02,
        2.69742068e-02,  9.79162902e-02, -1.52063267e-02, -1.41164929e-01,
        2.46829122e-01,  3.87747958e-02,  4.81700860e-02, -3.08904294e-02,
        1.81129709e-01,  2.41191953e-01, -1.57647461e-01, -6.15073442e-02,
       -1.46952540e-01,  1.45763662e-02,  7.03390688e-02, -3.01081359e-01,
        1.07162677e-01,  1.23069689e-01,  1.16620585e-01,  2.06164986e-01,
       -2.33981520e-01, -3.64763916e-01, -1.73861738e-02,  1.13101751e-02,
        2.14275032e-01, -1.64130181e-02, -1.99321344e-01,  1.78043529e-01,
       -2.03133494e-01,  3.53156254e-02, -1.62861422e-01,  7.35847875e-02,
       -6.44739419e-02,  6.03181124e-02,  8.41615200e-02,  9.02965367e-02,
       -7.13159442e-02,  2.08856408e-02, -1.35693103e-01, -8.34387317e-02,
       -1.33669227e-01,  1.21916093e-01,  1.84632152e-01, -1.17865160e-01,
       -3.97908986e-01, -6.37582839e-02, -3.62761587e-01, -9.20062512e-02,
       -1.98154092e-01, -1.21161141e-01,  4.53438684e-02, -4.16706726e-02,
       -9.25682634e-02, -2.51190633e-01,  1.90842152e-01, -9.84810367e-02,
       -3.73466164e-02,  1.52223483e-01,  1.03666104e-01, -1.35434002e-01,
        1.91635713e-02, -2.80295368e-02,  4.36545275e-02, -6.50647003e-03,
       -8.68569463e-02,  1.32288009e-01,  6.17309622e-02, -5.74537143e-02,
        2.23477364e-01, -1.37796402e-01,  6.74511045e-02, -7.64974430e-02,
       -2.26761863e-01,  1.42451867e-01,  3.04297712e-02,  1.25355572e-01,
        1.99635178e-02, -1.01627886e-01, -4.88474071e-02, -9.74937826e-02,
       -3.39901298e-01, -1.90254182e-01, -1.73891023e-01, -1.01652876e-01,
        3.29380631e-01,  9.75989029e-02, -5.80702908e-02,  1.63599253e-02,
        2.40258902e-01, -1.75971985e-01, -1.05988666e-01,  1.60449773e-01,
        3.76871973e-02, -3.39661390e-01, -2.04244047e-01, -2.57811010e-01,
       -1.57794863e-01,  1.78446826e-02, -6.28765374e-02, -2.37963796e-01,
       -2.90262461e-01, -1.15774274e-02,  1.39353827e-01, -3.05586040e-01,
        2.64224857e-01,  1.80097297e-01, -7.38849491e-02,  1.68107688e-01,
       -1.46572575e-01, -1.00064680e-01,  1.25209734e-01, -6.10247850e-02,
        1.42599657e-01, -1.06952205e-01, -2.14254022e-01, -9.07914620e-03,
       -6.92449370e-03,  2.51952466e-02, -1.64003983e-01,  6.80746883e-02,
        1.55186728e-01, -1.42946646e-01,  1.73069481e-02, -5.79648605e-03,
        5.15284529e-03,  1.27165586e-01, -2.48742819e-01, -6.69192299e-02,
        2.20016927e-01, -7.51211494e-02,  2.13564076e-02,  3.48673165e-01,
       -5.81951216e-02, -8.89999121e-02, -1.56511590e-01,  2.52397150e-01,
       -2.04492714e-02,  2.31753632e-01,  3.33782047e-01, -1.00066923e-01,
       -1.43792480e-02,  1.90306753e-01, -2.13991962e-02,  1.75101161e-02,
        1.46905914e-01,  9.51553434e-02, -2.69494116e-01, -2.75924683e-01,
       -3.48588943e-01,  1.04005657e-01,  1.60008758e-01, -1.74311191e-01,
        1.72385246e-01,  1.37426987e-01, -3.08452616e-03,  4.10986356e-02,
        2.50359476e-01, -1.38830543e-01, -1.01051018e-01, -3.27078514e-02,
        6.56512305e-02,  3.54793340e-01,  1.38222188e-01, -1.08092681e-01,
        3.99659947e-02, -2.58323848e-01, -1.47621306e-02,  1.06845103e-01,
        1.17141165e-01, -2.51076996e-01,  1.15051270e-01,  1.33106306e-01,
       -2.53528774e-01, -1.57530978e-01, -1.52284145e-01, -3.85973901e-02,
       -1.61608145e-01, -9.51233953e-02,  4.56629023e-02, -3.10599238e-01,
       -2.55535170e-03,  2.15873763e-01, -8.37611780e-02,  1.75826624e-01,
       -1.05619930e-01,  3.70748997e-01, -1.24416135e-01,  1.09951971e-02],
      dtype=float32)
```

Train-Test Split

- **Performed an 80/20 split for all feature types:**

```
# Convert Word2Vec embeddings to a NumPy array
X_word2vec = np.vstack(data['sentence_embeddings'].values)

# Target labels (ensure this column exists in your data)
y = data['LABEL']

# Split for BoW
X_train_bow, X_test_bow, y_train, y_test = train_test_split(BOW_df, y, test_size=0.2, random_state=42)

# Split for TF-IDF
X_train_tfidf, X_test_tfidf, _, _ = train_test_split(TFIDF_df, y, test_size=0.2, random_state=42)

# Split for Word2Vec
X_train_w2v, X_test_w2v, _, _ = train_test_split(X_word2vec, y, test_size=0.2, random_state=42)
```

Summary

- **The dataset was cleaned, tokenized, and normalized for Arabic text.**
- **Three feature representations were created: BoW, TF-IDF, and Word2Vec using a pre-trained AraVec model.**
- **Data was split for model training and evaluation.**

Comparative analysis

Overview

A comprehensive evaluation was performed on seven classifiers—Naive Bayes, SVM (RBF), Decision Tree, Random Forest, AdaBoost, Feedforward Neural Network (FNN), and LSTM—using three text representation techniques: Bag-of-Words (BoW), TF-IDF, and Word2Vec. The performance was measured based on accuracy, macro precision, macro recall, and macro F1-score.

Summary Table

Classifier	Vectorizer	Accuracy	Macro Precision	Macro Recall	Macro F1-Score
Naive Bayes	BoW	0.623	0.62	0.62	0.60
	TF-IDF	0.611	0.64	0.60	0.59
	Word2Vec	Skipped	Skipped	Skipped	Skipped
SVM (RBF Kernel)	BoW	0.620	0.65	0.61	0.61
	TF-IDF	0.644	0.67	0.63	0.63
	Word2Vec	0.683	0.69	0.67	0.67
Decision Tree	BoW	0.564	0.58	0.56	0.57
	TF-IDF	0.545	0.55	0.54	0.55
	Word2Vec	0.373	0.37	0.37	0.37
Random Forest	BoW	0.614	0.63	0.61	0.61
	TF-IDF	0.628	0.63	0.62	0.62
	Word2Vec	0.584	0.60	0.58	0.58
AdaBoost	BoW	0.410	0.49	0.42	0.40
	TF-IDF	0.407	0.51	0.41	0.40
	Word2Vec	0.497	0.47	0.49	0.47
FNN	TF-IDF	0.599	0.61	0.60	0.60
	Word2Vec	0.674	0.67	0.67	0.66
LSTM (BiLSTM + W2V)	Word2Vec	0.676	0.70	0.67	0.68

Observations

- **The LSTM model using Word2Vec achieved the highest overall performance, excelling in all macro-averaged metrics.**
- **The SVM classifier using Word2Vec showed the best performance among traditional models.**
- **The Feedforward Neural Network performed significantly better with Word2Vec than with TF-IDF, reflecting the advantage of semantic embeddings in neural architectures.**
- **Naive Bayes could not be evaluated with Word2Vec due to its inability to process negative values.**
- **Decision Tree consistently underperformed, particularly with Word2Vec features.**
- **Random Forest demonstrated stable moderate performance, with its best results from TF-IDF.**
- **AdaBoost underperformed on BoW and TF-IDF but showed improved results with Word2Vec, achieving nearly 50% accuracy and a balanced macro F1-score of 0.47.**

Challenges Faced with Arabic Data

Overview

Despite the growing interest in natural language processing for Arabic, working with Arabic-language datasets presents a unique set of challenges that are not typically encountered with English or other widely studied languages. These difficulties arise due to the linguistic, structural, and technical characteristics of Arabic, which can hinder the effectiveness of traditional machine learning and deep learning approaches.

Challenges

- **Complex Morphology: Arabic is a morphologically rich language, with words formed through root-and-pattern systems. This makes tokenization and stemming far more complex than in English, often leading to inconsistent preprocessing results.**

- **Dialectal Variations: Arabic has many dialects (e.g., Egyptian, Levantine, Gulf) that differ significantly from Modern Standard Arabic (MSA). These dialects often coexist within the same dataset, making it difficult for models to generalize and increasing the likelihood of misclassification.**

- **Ambiguity and Diacritics: The omission of diacritics (vowel markers) in written Arabic introduces high lexical ambiguity, where the same word form can represent different meanings. Most texts are written without diacritics, which forces the model to rely solely on context.**

- **Limited Resources: Compared to English, Arabic has fewer publicly available annotated datasets, pre-trained embeddings, and NLP tools. This scarcity limits the performance and scope of traditional and deep learning models.**

- **Encoding and Font Issues: Arabic script is written right-to-left, and poor encoding support (e.g., issues with UTF-8 or character rendering in visualizations) often caused unexpected bugs or display errors during preprocessing and evaluation.**

References

Links

- https://github.com/amrmalkhatib/Emotional-Tone/tree/master
- https://github.com/bakrianoo/aravec
- https://www.dropbox.com/scl/fi/rkztc22e7hn9oq5rc67nn/Twt-SG.zip?rlkey=1dutk1dg2xyplch9crxpg724t&e=1&dl=0