Faculty of Engineering
Cairo University

Computer Engineering Department

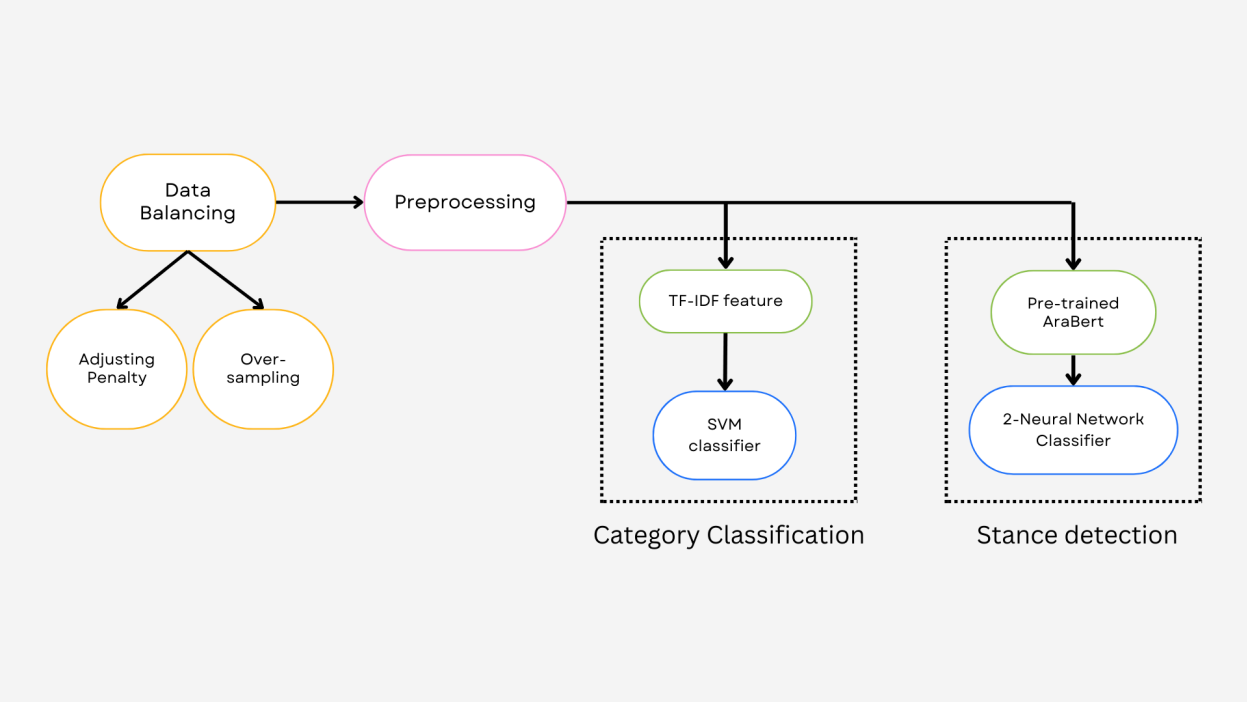
Natural Language Processing Project

Arabic Tweets Stance Detection and Category Classification

Team 12		
ghiath.ajam01@eng-st.cu.edu.eg		
Name	**Section**	**BN**
غياث عمر	2	8
ريم عماد	1	33
نوران هاني	2	34
هالة حمدي	2	35

Supervised By:
Dr. Sandra Wahid
Eng. Omar Samir

Project Pipeline



Libraries Used:

- nltk
- Arabic-Stopwords
- camel-tools==1.2.0
- farasapy
- arabert
- pandas
- scikit-learn
- gensim
- Transformers
- imblearn

Data preprocessing:

1. Removing Diacritization (التشكيل) and punctuation
2. Replacing links, numbers and mentions with <link>, <num> and <mt>
3. Converting emojis to equivalent text (😂 -> face_tearing_with_joy)
4. Normalizing letters, (ا <- أ إ آ)
5. Lemmatization using multiple tools (camel, farasa, nltk stem)
6. Converting English text to lowercase
7. Repeating hashtag words n times
8. Removing stopwords (combined nltk and Arabic-Stopwords) e.g. 'معي' , 'عندنا' , 'وأيها'
9. Removing duplicate rows
10. Tokenization using camel-tools simple word tokenizer

Additional Ideas:

1. Translate English text
2. Do everything in Arabic (emoji meanings, tokens, english text translation, …
3. Remove numbers or convert to word representation
4. Better stop words datasets, can also remove too rare / too frequent words
5. Named entity recognition

Approaching data imbalance:

We couldn't afford to neglect the imbalance in the data, as the metric chosen 'macro f1' will dramatically drop down if any class is not given proper labeling, had it been 'accuracy' we would very much happily exclude classes with probability <0.01

1. **Oversampling**, we repeatedly re-introduced random samples from minority classes, we tried both perfectly equal final distributions and scaling by a factor.

2. Adjusting **penalty** for misclassifications based on class distribution, i.e. minority classes have higher penalty. This is done using the parameter "class_weight='balanced'" in scikit-learn's classical models.

Feature Extraction

- **Bag of words (BOW)**
    - Simplified representation of data, where we don't care about the context but only about what words occurred in what tweets and by how much.
    - The feature is a matrix |Vocab| * |tweet | -> cell[word, tweet] == count(word in tweet).

- **Continuous BOW (word embeddings / vectors)**
    - Word2Vec model where a word is predicted based on surrounding words, a 3-layer NN is chosen and trained using a huge corpora.
    - The embeddings for each word is the weights between the input layer and the hidden layer.

- **Skip-gram (word embeddings / vectors)**
    - Word2Vec model where surrounding words are predicted based on current word, a 3-layer NN is chosen and trained using huge corpora.
    - The embeddings for each word is the weights between the input layer and the hidden layer.

- **TF-IDF**
    - Term Frequency-Inverse Document Frequency.
    - TF(w, tweet) = count(w in tweet), this eliminates very rare words.
    - IDF(w) = #tweets / #(tweets with the word w), this eliminates too frequent words.
    - We used both Word n-grams and Character n-grams.

- **Arabert Embeddings as a feature for SVM**
    - We took the "pooler output" from Bert, which resemble embeddings and feed them to **SVM** as a feature

Modeling:

Classical Models

- SVM
- Naive Bayes
- KNN
- Decision Trees
- Random Forest n_estimators = 1000
- Logistic Regression n_iterations = 300

Sequence Models

In the sequence models family, we've built a 3-layer LSTM followed by a linear neural network layer.

- An Embedding layer is first used to extract embedding of each token in the sentence.
- Then a 3-layer LSTM used to encode the sentence as an embedding vector. This happens as the hidden states of all tokens in the lstm's last layer are averaged, to obtain a single vector summarizing all tokens of a sentence.
- Then the embedding vector is fed to a linear neural network layer for classification.

LSTM's Training Settings:

Epochs	50
Batch Size	256
Learning rate	0.001
Embedding Dimension	300
LSTM Hidden layers dimension	50

AraBert

In the transformers family, we've fine-tuned an arabic bert model on our dataset.

- The arabic bert used was `aubmindlab/bert-base-arabertv02-twitter` from hugging face. We've chosen this model because it was trained on ~60 Million Arabic tweets.
- As per the documentation, we've used the preprocessing and tokenizer that was used when the model authors built their model.

Fine-tuning:

- We use the araBert as a feature extractor, by first freezing the bert's parameters, then passing the data through this arabert model, and producing the embedding as output. The sentence embedding is calculated by taking the last layer hidden-state of the first token of the sequence [CLS token].
- Then the sentence embeddings enters a classifier head we've built. The classifier head consists of 2 neural network layers in order to fine tune the weights of the model on our data.

Arabert Training Settings:

Epochs	50
Batch Size	16
Learning rate	0.001

Arabert embeddings with SVM classifier

Extract sentences' embeddings using arabert, then train these embeddings using a linear-kernel SVM.

Evaluation steps:

Data	Features	Classifier Model	Acc	F1	S/C
Original Data	Arabert	2 NN layers	82	52	S
Oversampling	Arabert	2 NN layers	73.8	58	S
Oversampling No Duplicates	Arabert	2 NN layers	76.2	60	S
Oversampling	Arabert batch=128	2 NN layers	77.6	61	S
Oversampling	Arabert embeddings	Linear SVM	53	29	S
Original Data	Embedding layer	3-layer LSTM + 1 NN layer	54.2	27.2	C
Oversampling data	Embedding layer	3-layer LSTM + 1 NN layer	56.6	25.9	C
Original Data	Arabert	2 NN layers	41	31	C
Original Data	HF - Arabert with lower learning rate		72.2	33.2	C
			84.1	65.2	S
Light Oversampling	HF - Arabert		82	64	S
	HF - Arabert batch=16		83	62.8	S
Oversampling	Arabert + 2 NN Layers + Our Preprocessing		73	57.3	S
Original Data	BOW	Linear SVM with Balanced Weights	76	51	S
			55	29	C
		Naive Bayes	46	25	C
			65	42	S
		Logistic Regression with custom penalty	59	29	C

Original Data	BOW	Logistic Regression with balanced weights	55	30	C
			77	55	S
		KNN k=5	79	40	S
		Ridge Classifier	76	50	S
		Random Forest n_estim=1000	80	46	S
		Decision Tree balanced	73	50	S
	BOW TFIDF_W TFIDF_C	Logistic Regression balanced	77	54	S
	TFIDF_W TFIDF_C	SVM rbf-kernel, balanced	68	25	C
		NB	44	24	C
	BOW TFIDF_C TFIDF_W	Linear SVM Balanced	54	28	C
	BOW TFIDF_C TFIDF_W	Linear SVM Balanced	77	52	S
Original Data	TFIDF_C TFIDF_W	Linear SVM Balanced, farasa lemmatize + non-lemmas	65	34	C
Original Data			80	56	S
OverSampled			77	53	S
Light Oversampling			79	51	S
Original Data	CBOW	Naive Bayes	51	31	S
		KNN k=5	61	36	S
		Linear SVM - balanced	80	31	S
		Logistic Regression - balanced	73	38	S
		Ridge - balanced	78	37	S
		Random Forest	80	30	S
		Decision Tree	59	31	S

Original Data	Skip Gram	Naive Bayes	63	37	S
		KNN k=5	71	36	S
		Linear SVM - balanced	80	32	S
		Ridge - balanced	80	40	S
		Random Forest	80	30	S
		Decision Tree	71	29	S

Models of choice:

We chose the models that scored highest macro F1 score against the ‘dev’ dataset

Stance:

- We used pre-trained deep learning model “**Arabert**” with HuggingFace implementation with slow learning rate and doing many epochs while remembering best macro 1-score so far

Training Settings:

Epochs	5
Batch Size	32
Learning rate	1e-5

Category:

- We used the classical model **linear SVM**, after achieving the same f1-score as **Arabert**, we decided to go with **LSVM** instead and trained on both train and dev datasets before predicting the test set.