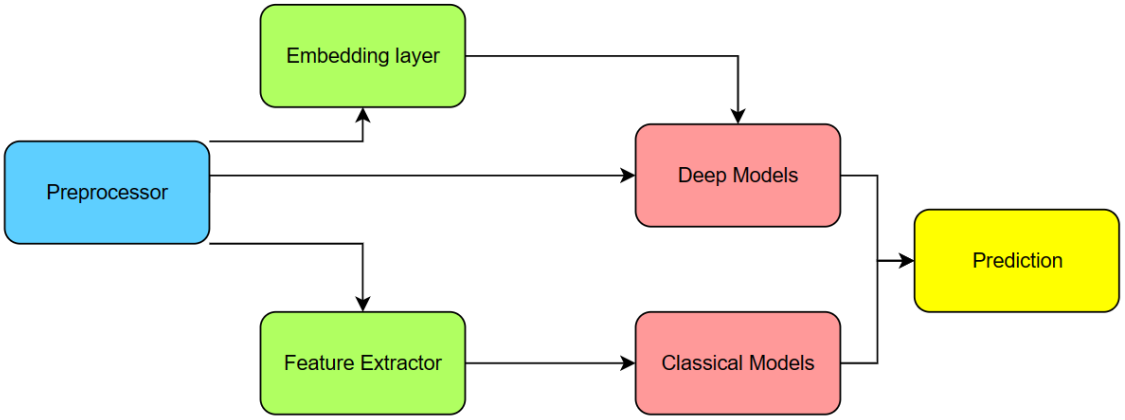
NLP Project Team 1 Document

Abdelrahman Jamal Sayed
Radwa Ahmed Mohamed
Iten Emad Eldin Saleh
Ayman Mohamed Reda

1. Project Pipeline:
2. A detailed description of each phase in your pipeline:
    a. Data preprocessing
    b. Feature extraction
    c. Model training
3. Evaluation: Report the macro F1-score (and all other metrics you tried) for all trials you did.
4. Specify what model you used for the test set submission on Kaggle and the reason for choosing it.

Project Pipeline:



Preprocessing:

- We defined our own tokenizer splitting on whitespaces & punctuations.
- Cleaning tokens from punctuation.
- Removing tashkeel.
- Replacing variants of all arabic letters to their original parents (ؤ to و and أ to ا and so on).
- Removing all extra Arabic characters (like المد).
- Remove any non-arabic characters

- Defined our own set of stop words consisting of 2 & 3 characters (so as not to lose much semantics), then removed them.
- We used NLTK Stemmer to stem our output tokens.
- We created **11** variants of our dataset, each with either Downsampling or Upsampling or both. Each with either equalised Stance Classes, or equalised Category Classes. All those Variants along with our Original Dataset without any up/downSampling.
- Our current best results for the **stance detection** is by using the dataset variant with **1000** tuples per stance & the best results for **category detection** is with the variant with **750** tuples per category.
- All dataset variants are stored and loaded as pickle files (.pkl).
- There's **another preprocessing pipeline** only for extracting the arabert pre-trained word embeddings for each word and using those embeddings on some sequential DL model but this method proved to be much worse than just training the Arabert model using transfer-learning.

Feature Extraction:

Our classical handPicked features:

- Tf-Idf :
    - Some modifications were added to resolve the issue of the train and test TF-IDF being of equal sizes. The modification was adding an unknown word to replace any words unseen in the train that are found in the test set with that unknown word.

    - SMOTE was sometimes used to oversample the TF-IDF.
- N-gram:
    - All sentences were padded to be of fixed length

Trainable features/embeddings:

- Word2Vec: CBOW using gensim models.
- Word2Vec: Skip-Gram using gensim models.

Classical Classifiers:

Main classical Classifiers

- Multinomial Naive Bayes
- SVM Linear
- Random Forest
- Logistic Regression
- Model parameters are optimised using **GridSearchCV** like Random Forest
- 10-Fold Cross validation with random forest but it was overfitting

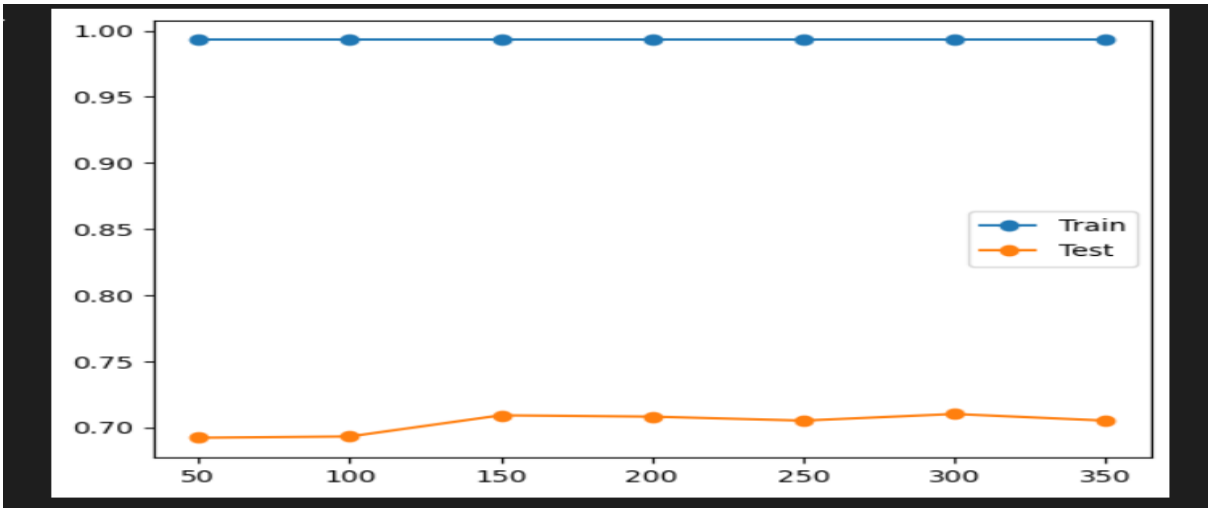
Results with train 3

- Features: TF-IDF

Classifier	Naive Bayes	Logistic Regression	SVM linear	Random Forest (100)	Random Forest(300)
Stance	69.6	67.6	67.4	69.3	71.0
Category	56.6	63.08	65.0	65.3	64.5

- Grid Search RandomForest for stance 0.67 accuracy , with f1-score :0.36,0.4,0.79
  and macro avg :0.52

Random Forest accuracy for stance:

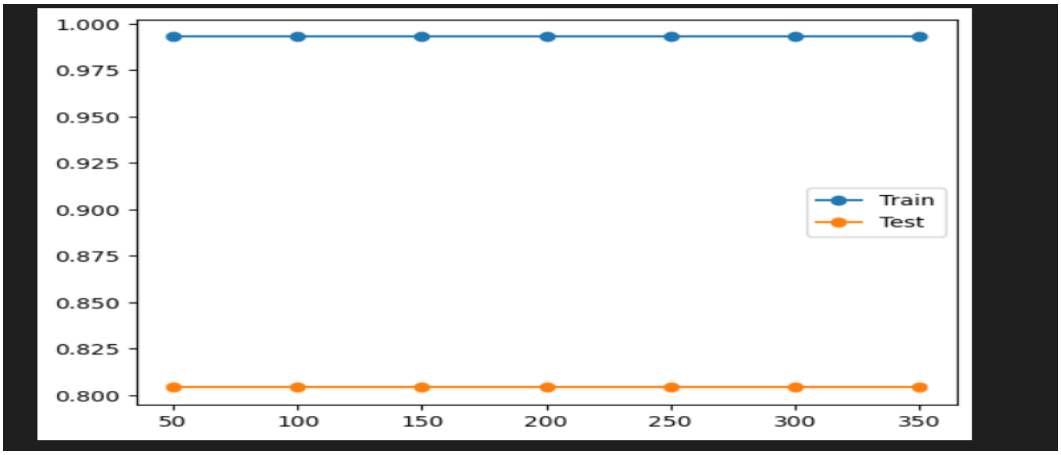


Classifier	Naive Bayes	Logistic Regression	SVM linear	Random Forest (100)
Stance F1-Score	0.3,0.39, 0.83	0.33,0.34,0. 81	0.32,0. 36,0.81	0.41,0.41, 0.82
Stance Macro avg	0.51	0.49	0.5	0.52
Category F1-Score	0.72,0.16, 0,0,0,0,0 ,021,0.20 ,0	0.75,0.79,0. 04,0.08,0,0, 0.67,0.47,0. 27,0	0.76,0.7 9,0,0.8, 0,0,0.6 7,0.48,0 .29,0.0	0.75,0.72, 0,0.8,0,0, 0,0.4,0.34 ,0
Category Macro avg	0.13	0.31	0.31	0.23

- Features: TF-IDF and Bigrams

Classifier	Naive Bayes	Logistic Regression	SVM linear	Random Forest (100)	Random Forest(300)	10 Fold with cross-va lidation
Stance	12.6	80.4	12.6	80.4	80.4	84.0
Category	54.5	54.5	54.5	54.5	64.5	—

- Random Forest for stance graph for accuracy

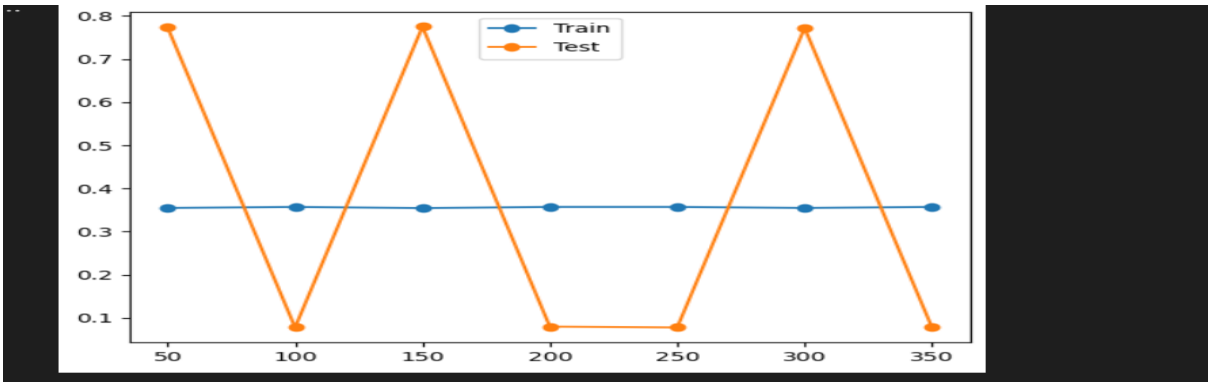


Classifier	Naive Bayes	Logistic Regression	SVM linear	Random Forest (100)
Stance F1-Score	0,0.22,0	0.0.0.89	0,0.22, 0	(0,0,0.89)
Stance Macro avg	0.07	0.3	0.07	0.3
Category F1-Score	0.71,0,0, 0,0,0,0,0 ,0,0	0.71,0,0,0,0, 0,0,0,0,0	0.71,0,0 ,0,0,0,0 ,0,0,0	0.71,0,0,0, 0,0,0,0,0, 0
Category Macro avg	0.07	0.07	0.07	0.07

- Features: Bigram
  This is so simple and overfitted so it is not our major focus to use it without another feature

Classifier	Naive Bayes	Logistic Regression	SVM linear	Random Forest (100)	Random Forest(300)
Stance	7.0	79.7	80.0	77.8	7.8
Category	54.5	54.5	54.5	53.3	53.3

Random Forest with Bigram for stance with different forests

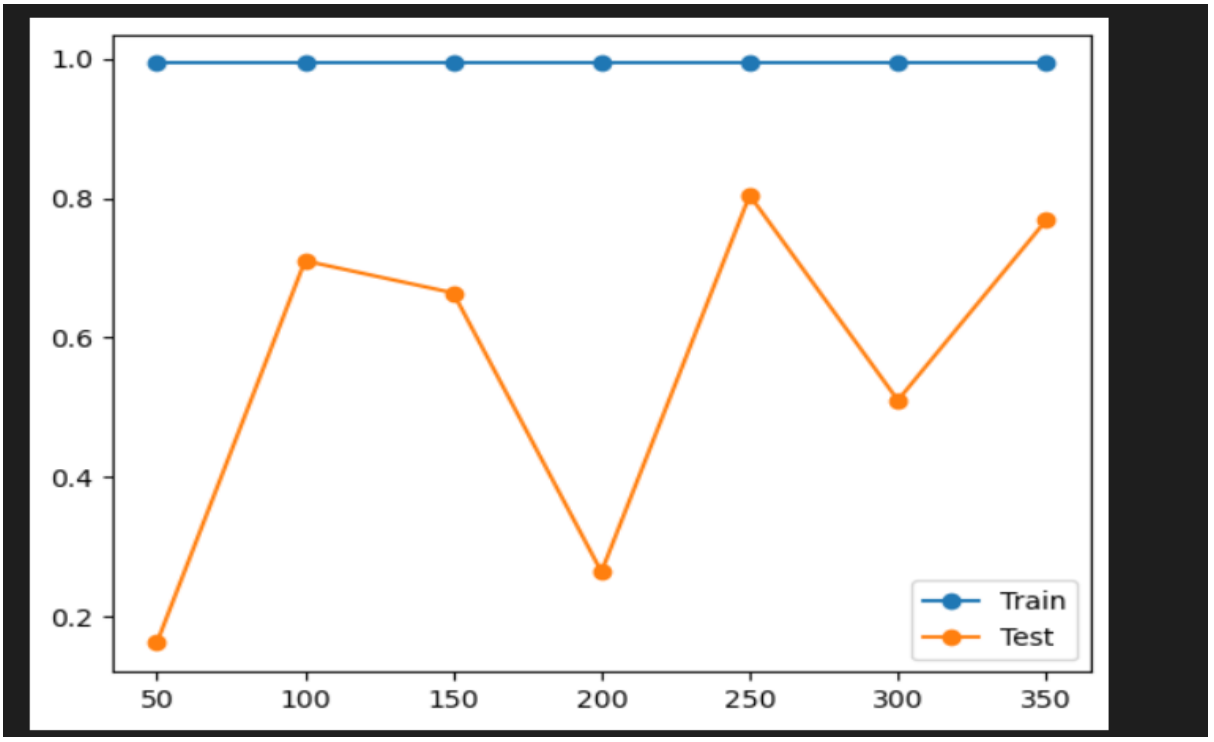


Classifier	Random Forest (100)
Stance F1-Score	0.0,0.2,0.89
Stance Macro avg	0.30
Category F1-Score	0.7,0,0,0,0,0,0,0,0.04,0
Category Macro avg	0.07

- CBOW & Skip Gram & TF-IDF Features accuracy:

Classifier	Naive Bayes	Logistic Regression	SVM linear	Random Forest (100)	Random Forest(350)
Stance	67.7	70.5	12.7	71.0	76.8
Category	49.7	57.1996	13.9	12.8	54.555

CBOW & Skip Gram & TF-IDF Feature accuracy
Random Forest with stance accuracy :



Classifier	Naive Bayes	Logistic Regression	SVM linear	Random Forest (350)
Stance F1-Score	0.32,0.4, 0.81	0.28,0.14,0. 84	015,0.2 4,0.03	0,0.15,0.8 7
Stance Macro avg	0.51	0.42	0.14	0.34
Category F1-Score	0.71,0,0, 0,0,0,0,0 .11,0,0	0.69,0.61,0. 1,0.08,0.09, 0,0.67,0.47,, 0.32,0	0.02,0.1 ,0,0.08, 0,0,0,0. 23,0.16, 0	0.71,0,0,0, 0,0,0,0,0, 0
Category Macro avg	0.08	0.3	0.06	0.07

Deep Learning models:

- Pytorch's RNN
- AraBert's Transformer (Using transfer learning to train the last Fully Connected layer)
- XML Roberta (Using transfer learning to train the last Fully Connected layer)
- Qarib
- MarBert

The Word embeddings used in the RNN were extracted from AraBert's pretrained embeddings.

Note: All the DL models for stance & category were trained using both the original dataset as well as all the other variants that equalised either the stance or the category (with different number of samples per the relevant class). The following are the best results after training on all the variants

Here are the dataset variants

Dataset	Size	How it was made
Train 1	6988	**The original dataset**
Train 2	1500	Equalising all stances to **500** tweets per **stance**
Train 3	3000	Equalising all stances to **1000** tweets per **stance**
Train 4	7500	Equalising all stances to **2500** tweets per **stance**
Train 5	500	Equalising all categories to **50** tweets per **category**
Train 6	1000	Equalising all categories to **100** tweets per **category**
Train 7	1500	Equalising all categories to **150** tweets per **category**
Train 8	2500	Equalising all categories to **250** tweets per **category**
Train 9	5000	Equalising all categories to **500** tweets per **category**
Train 10	7500	Equalising all categories to **750** tweets per **category**
Train 11	10000	Equalising all categories to **1000** tweets per **category**

- Arabert Word Embeddings with RNN

Task	f1-score	Macro avg
stance	0.00,0.00,0.85	0.30
category	0.71, the rest are 0.0	0.07

- TF-IDF with RNN

Task	f1-score	Macro avg
stance	0.05,0.17,0.77	0.33
category	0.18,0.17,0.14,0,0,0,0,0,0,0	0.05

- TFXLMRoberta with stance

Train data	f1-score	accuracy	Macro avg
Train 1	0.36,0.54,.91	0.83	0.605
Train 2	0.42,0.49,0.88	0.78	0.603

  - Bottom line is, with the rest of the variants for the stance detection, it keeps hovering at around 0.60 macro avg and the maximum observed is 0.63.

- TFXLMRoberta with category

Train data	f1-score	accuracy	Macro avg
Train 5	0.3934, 0;73, 0.22, 0.2, 0.13, 0.15, 0.095, 0.32, 0.32, 0.08	0.374	0.2693
Train 6	0.43, 0.77, 0.28, 0.16, 0.25, 0.172, 0.00, 0.432, 0.28, 0.03	0.409	0.28
Train 9	0.55, 0.866, 0.282, 0.218, 0.23, 0.26, 0.00, 0.554, 0.452, 0.1	0.524	0.351
Train 10	0.6356, 0.848, 0.33,	0.587	0.361

	0.21, 0.285, 0.25, 0.00, 0.583, 0.467, 0.00		

- Bottom line is that the best accuracy for categories is generally obtained at Train 9 or Train 10 & generally hovers around 0.36

- XLMRoberta with stance

Train data	f1-score	accuracy	Macro avg
Train 4	0.43,0.5,0.9	0.807	0.612

- This is the same as the above model but using the pytorch implementation, the observations are the same if a little worse & the model generally seemed to take more time to train & way more memory as well.

- XLMRoberta with category

Train data	f1-score	accuracy	Macro avg
Train 5	0.393,0.73,0.271,0.2, 0.132,0.148,0.095,0. 32,0.08	0.374	0.269

- Similar results to TFXLMRoberta, best results hover around 0.36 at max

- Arabert with stance

Train data	f1-score	accuracy	Macro avg
Train 3	0.57,0.90,0.53	0.81	0.66

- Train 3 is the best result with 0.66 F1-Macro Average, the result of the results (train 1, 2 & 4) hover at or around 0.655.

- Arabert with category

Train data	accuracy	Macro avg
Train 10	0.588	0.438
Train 11	0.522	0.392

    - Train 10 ended up being the best overall dataset variant giving us nearly 0.44 macro average with the rest (train 1, 5 -> 11) giving results ranging from 0.25 up to 0.42

- QARiB with Stance Detection (3 epochs & Train 1, max_len=55)

	precision	recall	f1-score	support
0	0.51	0.44	0.47	70
1	0.53	0.47	0.50	126
2	0.90	0.92	0.91	804
accuracy			0.83	1000
macro avg	0.64	0.61	0.63	1000
weighted avg	0.82	0.83	0.83	1000

- QARiB with Category Classification (3 epochs on train 10, max_len=55)

	precision	recall	f1-score	support
0	0.83	0.43	0.57	545
1	0.84	0.90	0.87	145
2	0.24	0.68	0.36	82
3	0.08	0.10	0.09	20
4	0.31	0.33	0.32	15
5	0.21	0.30	0.25	10
6	0.50	0.50	0.50	2
7	0.45	0.75	0.57	128
8	0.35	0.50	0.41	36
9	0.11	0.06	0.08	17
accuracy			0.55	1000
macro avg	0.39	0.46	0.40	1000
weighted avg	0.68	0.55	0.56	1000

- QARiB with Category Classification (3 epochs on train 10)

	precision	recall	f1-score	support
0	0.80	0.49	0.61	545
1	0.82	0.93	0.87	145
2	0.23	0.59	0.33	82
3	0.11	0.10	0.10	20
4	0.28	0.33	0.30	15
5	0.23	0.30	0.26	10
6	1.00	0.50	0.67	2
7	0.51	0.65	0.57	128
8	0.40	0.56	0.47	36
9	0.07	0.12	0.09	17
accuracy			0.56	1000
macro avg	0.44	0.46	0.43	1000
weighted avg	0.67	0.56	0.59	1000

Final models:

- For both the stance & category detection, we ended up using two arabert models trained on different variants of the dataset (Train 3 for stance & train 10 for category).
- They were chosen because they gave the best F1-scores & especially the macro acreage scores. They ended up yielding nearly the best accuracy along with actually learning the dataset and predicting based on the tweet contents, not just predicting a single class & plainly overfitting.