Report 1

May 26, 2023

1 Motivation

Social media platforms have become an integral part of our daily lives, where users express their opinions and share their experiences on a wide range of topics. With the massive volume of data generated on these platforms, it's becoming increasingly challenging to manually classify and analyze tweets based on their topics. This is particularly true for Arabic tweets, which present unique linguistic challenges and require specialized tools and techniques.

That's where our NLP project comes in! By leveraging the latest advancements in NLP and machine learning, we aim to develop an accurate and efficient model that can classify Arabic tweets into different categories based on their topics. This will enable individuals and organizations to gain valuable insights into what people are talking about and make informed decisions based on real-time data.

Our project has the potential to make a significant impact on various fields, such as marketing, politics, and social sciences. With the increasing importance of Arabic as a global language, our project also has the potential to contribute to the development of Arabic language processing tools and resources.

We will fine-tune a pretrained model MARBert model. The network architecture is the same as BERTBase (12 layers, 768 hidden units, 12 heads).

2 Related Work

There have been several related works on Arabic tweet classification that may be relevant to this project. One of them is this paper which proposes a model for Arabic sentiment analysis using a Twitter dataset and deep learning models with Arabic word embedding. It uses the supervised deep learning algorithms on the proposed dataset. The dataset contains 51,000 tweets.The experiment has been carried out by applying the deep learning models, Convolutional Neural Network and Long Short-Term Memory while comparing the results of different machine learning techniques such as Naive Bayes and Support Vector Machine. The accuracy of the AraBERT model is 0.92

3 Dataset

In our project we used SANAD dataset. SANAD Dataset is a large collection of Arabic news articles. The articles were collected using Python scripts written specifically for three popular news websites: AlKhaleej, AlArabiya and Akhbarona. SANAD has seven categories, which are: Culture, Finance, Medical, Politics, Religion, Sports and Technology.For this project, we used a balanced subset of the dataset in which each category contains 6,500 articles, so we have in total 45,500 articles

4 Challenges

The limited availability of Arabic NLP resources is a significant challenge in developing NLP models for Arabic text classification, including tweets. Unlike English, Arabic is a morphologically rich language with a complex grammar and unique linguistic features, such as the presence of diacritics and

the right-to-left script.

Collecting and preprocessing a large and diverse Arabic text dataset can help address the limited availability of NLP resources. Preprocessing techniques such as stemming, normalization, and stop word removal can also help to simplify the text and reduce the complexity of the language.

5 Preprocessing Steps

Preprocessing the data involves the following steps

1. **Remove stop words** : For this task, we used the Arabic stop words corpus that is provided by NLTK. However, this list of Arabic stop words provided by NLTK was not comprehensive, so we added an extra stop words using an external file contains more than Arabic 2000 stop words.
2. **Remove English letters.**
3. **Remove numeric digits : for this task, we take into consideration both of the Arabic and English digits to be removed.**
4. **Remove Punctuations :** For this task, we used the constant punctuations that is provided by string module in python. This constant contains all ASCII punctuations characters like "!", "؟", etc.. However, this set was not sufficient as the Arabic contains more different punctuations such as the Arabic punctuation "", so we add manually a customized set of extra punctuations to be more comprehensive.
5. **Remove extra spaces.**
6. **Remove Arabic diacritics :** lso known as tashkeel such as Fatha, Kasra, Damma and so on.
7. **Normalize arabic letter :** this involves replacing different forms of the same letter with a standard form. This can improve the accuracy of text analysis and classification by reducing the number of unique characters that the algorithm has to consider.
8. **Normalize words :** This is done by applying stemming, and we used for this ISRIStemmer algorithm from NLTK.

6 Data Analysis

First, we checked if out data contains null values and we found it does not. However, we found that we have 15 duplicate values (rows) in the dataset, so we removed them. Before removing duplicates, the date was balanced, 6,500 articles for each category. But, because that the number of removed rows is very small, the data is still considered as balanced as shown in Figure 1. Also we used word cloud to figure out what is the common words in each category, There aresome examples in the end of the report

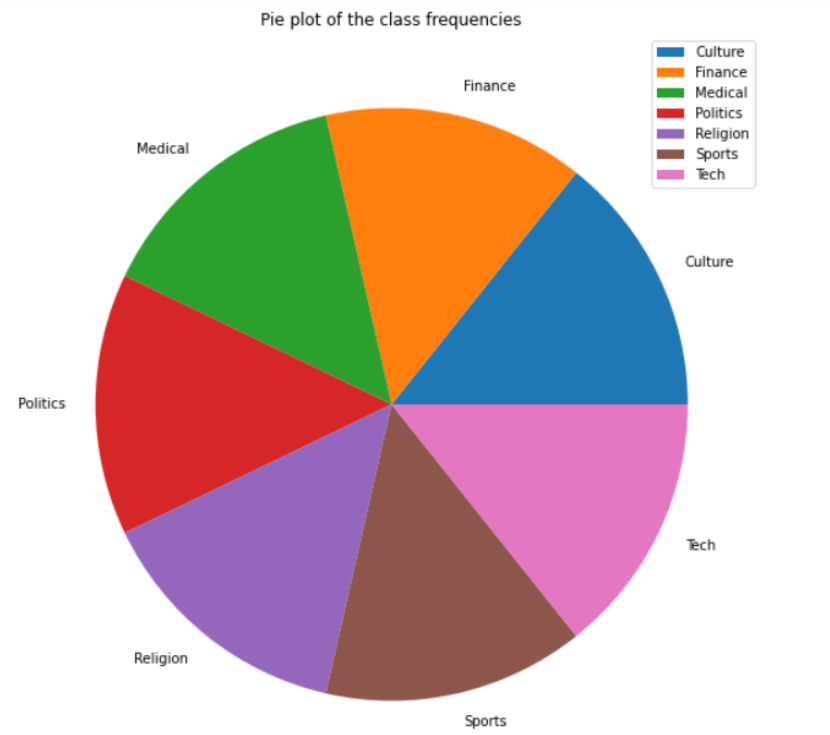

Figure 1: classes chart.

Figure 2: Politics word cloud.

Figure 3: Finance word cloud.

7 BERT Architecture

The MARBERT model, also known as AraBERT (Arabic BERT), is based on the BERT (Bidirectional Encoder Representations from Transformers) architecture. BERT is a powerful pre-trained language model developed by Google that has achieved state-of-the-art performance on various natural language processing tasks. The architecture of MARBERT follows the same principles as BERT but is specifically designed and trained for Arabic text. Here is a high-level overview of the MARBERT model architecture.

Let's start by explaining the architecture of BERT (Bidirectional Encoder Representations from Transformers) and then move on to MARBERT (Multilingual Arabic BERT).

1. **BERT :** BERT is a transformer-based model that utilizes the Transformer architecture, which was introduced in the "Attention Is All You Need" paper. It consists of an encoder stack that is composed of multiple layers of self-attention and feed-forward neural networks. Here are the key components of BERT :

   (a) Input Embeddings : BERT takes variable-length input sequences, typically tokenized sentences. Each token is initially represented as the sum of three embeddings :

      i. Token Embeddings: Word-level embeddings for each token in the input sequence.

      ii. Segment Embeddings: Segment-level embeddings to distinguish different sentences or segments within the input.

      iii. Positional Embeddings: Positional information embeddings to indicate the position of each token in the sequence.

   (b) Transformer Encoder Layers: BERT utilizes a stack of transformer encoder layers. Each layer has two sub-layers:

      i. Multi-Head Self-Attention: This sub-layer allows the model to attend to different parts of the input sequence while considering dependencies between all tokens.

      ii. Feed-Forward Neural Network: This sub-layer applies a point-wise feed-forward neural network to each token's representation independently.

   (c) Output Layers : : BERT typically uses the representation of the [CLS] token (which is added at the beginning of the input) as the aggregate representation of the entire sequence. This representation is then passed through a final classification layer to predict various downstream tasks.

   BERT has been pretrained on a large corpus of text data using two unsupervised learning tasks: Masked Language Modeling (MLM) and Next Sentence Prediction (NSP). The pretrained BERT model can be fine-tuned on specific downstream tasks, such as text classification, question answering, and named entity recognition.

2. **MARBERT :** MARBert, or Multilingual Arabic BERT, is a variant of BERT that is specifically trained for the Arabic language. It follows a similar architecture to BERT but is pretrained on a large corpus of Arabic text data. MARBERT captures the linguistic patterns and semantics specific to the Arabic language. The key difference between MARBERT and the original BERT lies in the pretraining data. While BERT is pretrained on a mixture of languages, MARBERT focuses exclusively on Arabic. This enables MARBERT to learn Arabic-specific language features, making it more effective for Arabic NLP tasks.

   For this task, we performed two approaches of preprocessing before fine-tuning the model. The first approach is to apply all cleaning steps without normalizing the words. The second approach is to apply all cleaning steps in addition to applying stemming, and we compared the performance between those approaches.

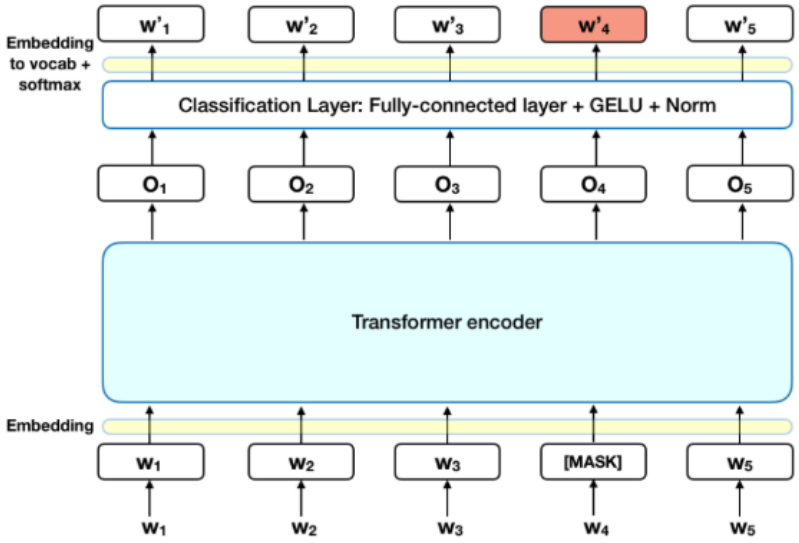

Figure 4: BERT Architecture.

8 Fine-tuning steps

8.1 Encoding labels :

To make this column suitable for the training process, we need to encode the labels to be represented as integers. So, we assigned a unique code for each unique category. The range of the code is [0:6] because we have 7 classes. Here is the map of codes and their classes :

1. Code: 0 - Class: Culture
2. Code: 1 - Class: Finance
3. Code: 2 - Class: Medical
4. Code: 3 - Class: Politics
5. Code: 4 - Class: Religion
6. Code: 5 - Class: Sports
7. Code: 6 - Class: Tech

8.2 Split the dataset :

We splitted the dataset into three datasets : training, validation and testing. In our case : 80% of the data is allocated for training, 10% for validation and 10% for the testing part.

In our project we used SANAD dataset. SANAD Dataset is a large collection of Arabic news articles. The articles were collected using Python scripts written specifically for three popular news websites: AlKhaleej, AlArabiya and Akhbarona. SANAD has seven categories, which are: Culture, Finance, Medical, Politics, Religion, Sports and Technology.For this project, we used a balanced subset of the dataset in which each category contains 6,500 articles, so we have in total 45,500 articles.

8.3 Encode training validation datasets :

Tokenization is a crucial step in natural language processing tasks, where text is divided into individual tokens or subwords. In this task, we used the BERT tokenizer. The tokenizer object takes in the text data as input and performs tokenization based on the specified parameters. The first parameter we used is truncation. We set truncation=True parameter to ensure that any input text longer than the specified **max_length** is truncated to fit the maximum sequence length. The second parameter is The padding. We set it to True to add padding tokens to the sequences that are shorter than the **max_length**, ensuring that all sequences have the same length. The last parameter is The **max_length** parameter which sets the maximum sequence length for the tokenized sequences.

Then we used the tokenized encodings from the **train_encodings** and **val_encodings** variables to create PyTorch TensorDataset objects, which will be used as input for training and validation. The TensorDataset is a PyTorch class that combines tensors along the first dimension to create a dataset. For the training and validation datasets, we combined three tensors: **input_ids** tensor, **attention_mask** tensor and **train_labels** tensor. The **input_ids** tensor contains the tokenized input sequences, the **attention_mask** tensor represents the attention masks indicating which tokens to pay attention to, and the **train_labels** tensor holds the corresponding labels for the training data.

Then we use the DataLoader class from PyTorch which is used to create data loaders for the training and validation datasets. Data loaders are used to efficiently load the data in batches during training and evaluation processes. In our task, we set the batch size to be 8.

Also we used the **AdamW** optimizer is used, which is a variant of the Adam optimizer that includes weight decay regularization. The optimizer is responsible for updating the model's parameters during the training process to minimize the loss. The loss function we used is the **CrossEntropyLoss** function. This loss function is commonly used for multi-class classification tasks. It combines the **softmax** activation function with the negative log-likelihood loss to compute the loss value.

8.4 Fine-tuning loop :

The fine-tuning loop begins, iterating over the specified number of epochs. For each epoch:

1. The model is set to the training mode, and the training data is iterated over in batches.
2. The optimizer is zeroed, and the model is called to compute the output logits and loss.
3. Backpropagation is performed, and the optimizer updates the model's parameters.
4. The average training loss for the epoch is calculated and printed

After the training loop, the model is set to the evaluation mode. The validation data is iterated over in batches, and the same steps as in training are performed to compute the validation loss and obtain predicted labels.

8.5 Evaluation :

The model is set to evaluation mode to disable certain operations during inference. Two empty lists, **true_labels** and **predicted_labels**, are initialized to store the true labels and predicted labels for evaluation. The data loader is iterated over, and for each batch, the input tensors, attention masks, and labels are transferred to the device. The model is called with the input tensors to obtain the output logits. The predicted class is determined by taking the index of the maximum value along the logits' dimension 1. The true labels and predicted labels are extended with the converted list values. After iterating over all batches, the evaluation metrics are calculated using the true labels and predicted labels.

8.5.1 First Approach

This approach is the one **without** applying stemming The **accuracy, precision, recall, and F1-score** are computed using the respective functions from scikit-learn, with the 'macro' average indicating computation for each class and averaging the results. Finally, the function returns the calculated accuracy, precision, recall, and F1-score values, and here are values respectively : **0.9820, 0.9821, 0.9822 and 0.9821** .

8.5.2 Second Approach

This approach is the one **with** applying stemming The **accuracy, precision, recall, and F1-score** are computed using the respective functions from scikit-learn, with the 'macro' average indicating computation for each class and averaging the results. Finally, the function returns the calculated accuracy, precision, recall, and F1-score values, and here are values respectively : **0.2596, 0.1184, 0.2743 and 0.1506** .

9 Results

We decided to show only results from the first approach because the second approach has a low accuracy compared to the first one. Here are some screenshots for arabic tweets and how our model predicts them correct.

First tweet

Second tweet

Third tweet

Fourth tweet

Fifth tweet

Sixth tweet

Seventh subfigure

Eighth subfigure

Figure 7: Tech Predictions.

Figure 8: Politics Predictions.

Figure 9: Sports Predictions.

Figure 10: Finance Predictions.

Figure 11: Medical Predictions.

Figure 12: Religion Predictions.

Figure 13: Culture Predictions.