Topic Detection , Named entity Recognition Diacritization for Arabic Articles

MEKTOUBI Zakaria and FATHALLAH Othmane

Faculty of Sciences Semlalia, Cadi Ayyad University, Marrakech

Introduction

Arabic is the official language of 22 countries, spoken by more than 400 million people. It is recognized as the 4th most used language on the Internet. Over the last decade, Arabic and its dialects have begun to gain ground in the field of Natural Language Processing (NLP) research. Much work has been done on different aspects related to the processing of this language and its dialects, such as morphological analysis, resource creation, machine translation, etc. The NLP research has been carried out in a number of different areas. In order to present the characteristics of this language and to classify the works that deal with it, in our case, our project is based on the detection of the subject, the recognition of named entities and the diacritics of news articles in Arabic. After receiving the project, we did some research in order to better understand the subject and what we had to do, so to deal with it we thought to split the project into mini-tasks as follows:

1. Data Collection.
2. Create a Topic Detection Model.
3. Discritization.
4. Entity name Recognition.
5. Deployment.

1. Data Collection:

In any NLP project, the first issue that faces any Data Scientist is how and where to find a good source of Data, Because the quality of data is a determining factor for the success of the project. In purpose to deal with this issue we thought to scrape the data (Arabic Articles and their Categories) from many Arabic newspapers such as Hespress, BBC Arabic, CNN Arabic to build our Dataset, this latter is a CSV file that contains two columns, Article and Category.

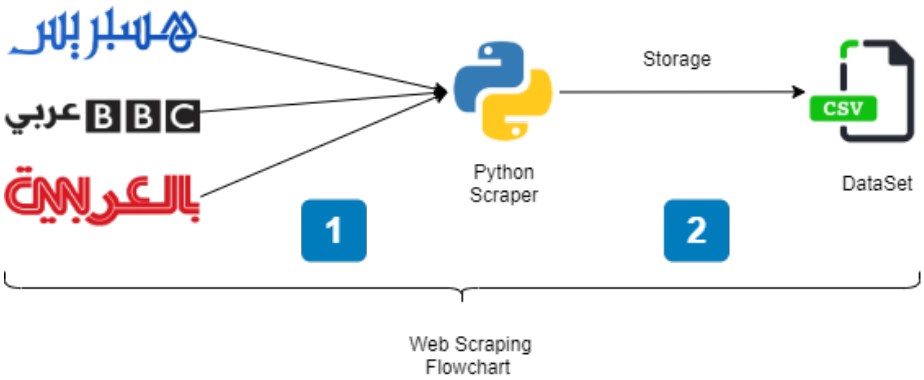


Fig. 1: Scraping Data

Our Data set Contain 4 Categories of Articles as follows

1. Politic
2. Business Economy
3. Art culture
4. Sport

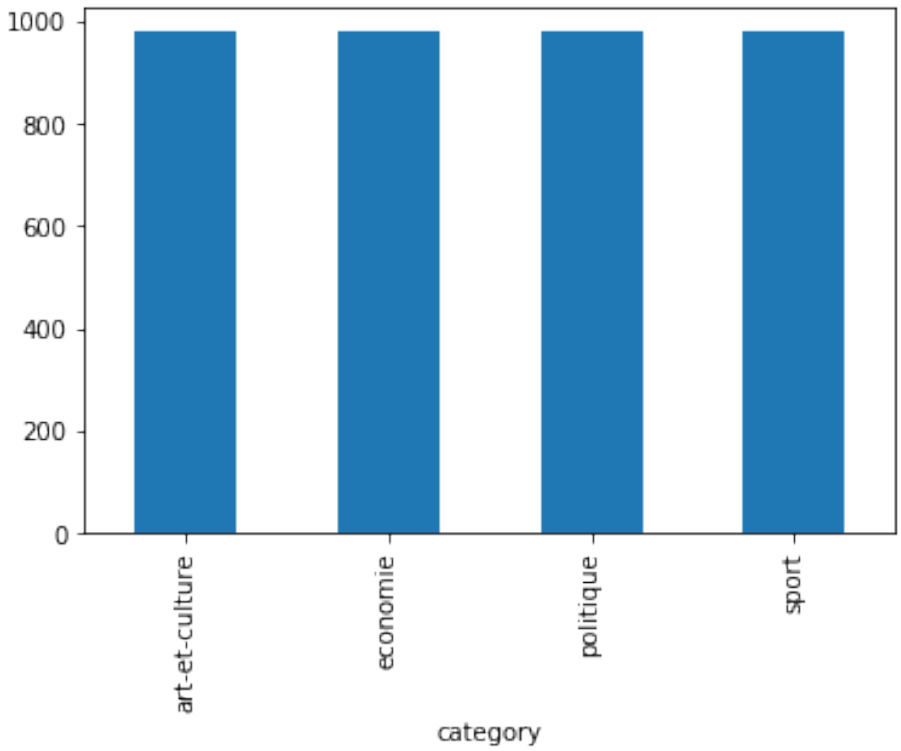


Fig. 2: Data Distribution

As not to fall into the problem of Imbalanced Data Each category contains 980 Article.

2. Data Preprocessing

In this part of the project the aim was to clean our data in purpose to remove noisy data that can hamper the performance of the model, our preprocessing was as follow

1. Remove Noisy Data:
   (a) Tokenization
   (b) Removing Stop words
   (c) Removing punctuation
   (d) Lemmatization: in this step, we used the Farasa lemmatizer, Farasa is a toolkit that provides many modules as lemmatizer, stemmer, etc.
2. Vectorization: in purpose to vectorize our data we used TF-IDF stands for Term Frequency-Inverse Document Frequency which basically tells the importance of the word in the corpus or dataset. TF-IDF contain two concept Term Frequency(TF) and Inverse Document Frequency(IDF).

3. Topic Detection

After Preparing Data it's time to start building our models, the first one is a model that gonna classify Articles into 4 Classes or categories (Politic, Business Economy, Art Culture, Sport), we have trained 3 models Logistic Regression,RandomForestClassifier , MultinomialNB(Naive Bayes).
In purpose to choose the best model that have the better result, we computed some metrics such as accuracy .

Model	Accuracy
Logistic Regression	0.958418
RandomForestClassifier	0.914031
MultinomialNB(Naive Bayes)	0.954592

Fig. 3: Accuracy

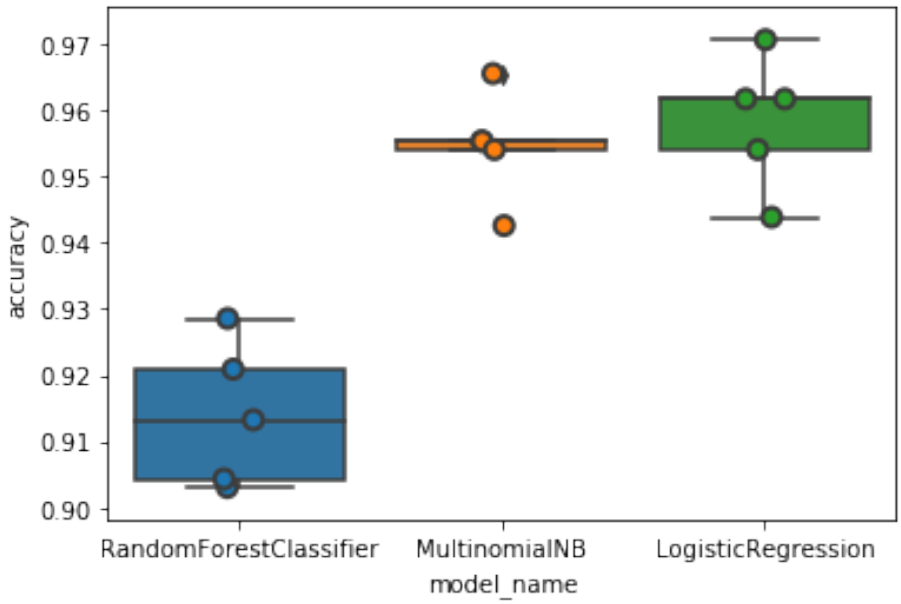
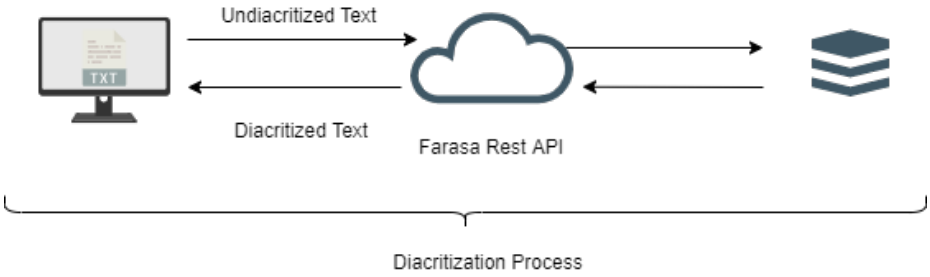


Fig. 4: Box Plot of Accuracy

Based on the previews plot that gives us a big picture about the efficiency and the performance of the models, we have chosen the MultinomialNB(Naive Bayes) classifier.

4. Diacritization



Diacritization of Arabic text is both an interesting and a challenging problem at the same time with various applications ranging from speech synthesis to helping students learning the Arabic language. Like many other tasks or problems in Arabic language processing, the weak efforts invested into this problem and the lack of available (open-source) resources hinder the progress towards

solving this problem.In the current Project we used the Rest API provided by Farasa toolkit which is a text processing toolkit for Arabic text. Farasa consists of the segmentation/tokenization module, POS tagger, Arabic text Diacritizer, and Dependency Parser. According to our use the Diacritization module is very powerful but the disadvantage is that it is written in Java so it is very difficult to implement it under python.

5. Named Entity Recognition

Named Entity Recognition (NER) is the process of identifying and classifying entities such as persons, locations and organisations in the full-text in order to enhance searchability.

Named entity recognition (NER) , also known as entity chunking/extraction, is a popular technique used in information extraction to identify and segment the named entities and classify or categorize them under various predefined classes.

NER is used in many fields in Natural Language Processing (NLP), and it can help answering many real-world questions, such as:

- Which companies were mentioned in the news article?
- Were specified products mentioned in complaints or reviews?
- Does the tweet contain the name of a person? Does the tweet contain this person's location?

5.1. Methods

In order to perform this subject, we have taken on two approahes:

- Statistical approach, based on FARASA Toolkit.
- Deep Learning approch, based on Bidirectional LSTM-CNN.

A. FARASA Toolkit

It is a tool that has the lowest error rate compared to other tools based on Arabic morphology. It uses more than 700K tokens for training the proper model.

As convenient, this toolkit is made with Java language (among 5 JAR files), which requires a lot of ressources in terms of RAM capacity.

B. LSTM-CNN

we used a CRF-LSTM to model the sequence structure of our sentences. We used the LSTM on word level and applied word embeddings.If we haven't seen a word a prediction time, we have to encode it as unknown and have to infer it's meaning by it's surrounding words. Often word postfix or prefix contains a lot of information about the meaning of the word.

To encode the character-level information, we will use character embeddings and a LSTM to encode every word to an vector. We can use basically everything that produces a single vector for a sequence of characters that represent a word.Then we feed the vector to another LSTM together with the learned word embedding.

B.1. Data Preparation

We start as always by loading the data.

```
1 data = pd.read_csv("/content/drive/My Drive/Projects/Named_Entity_Recognition/data.csv", encoding="latin1")
2 data = data.fillna(method="ffill")
3 data.tail(n=10)
```

	Sentence #	Word	POS	Tag
1048565	Sentence: 47958	impact	NN	O
1048566	Sentence: 47958	.	.	O
1048567	Sentence: 47959	Indian	JJ	B-gpe
1048568	Sentence: 47959	forces	NNS	O
1048569	Sentence: 47959	said	VBD	O
1048570	Sentence: 47959	they	PRP	O
1048571	Sentence: 47959	responded	VBD	O
1048572	Sentence: 47959	to	TO	O
1048573	Sentence: 47959	the	DT	O
1048574	Sentence: 47959	attack	NN	O

Fig. 5: Data Preview

We have 35178 different words with 17 different tags. We use the `SentenceGetter` class from last post to retrieve sentences with their labels.

```
1 words = list(set(data["Word"].values))
2 n_words = len(words)
3 tags = list(set(data["Tag"].values))
4 n_tags = len(tags)
5 print(f"Number of Words: {n_words}")
6 print(f"Number of Labels(tags): {n_tags}")
```

```
Number of Words: 35178
Number of Labels(tags): 17
```

Fig. 6: Data Preview

B.2. Tokens Preparation

Now we introduce dictionaries of words and tags.

```
# Vocabulary Key:word -> Value:token_index
# The first 2 entries are reserved for PAD and UNK
word2idx = {w: i + 2 for i, w in enumerate(words)}
word2idx["UNK"] = 1 # Unknown words
word2idx["PAD"] = 0 # Padding

# Vocabulary Key:token_index -> Value:word
idx2word = {i: w for w, i in word2idx.items()}

# Vocabulary Key:Label/Tag -> Value:tag_index
# The first entry is reserved for PAD
tag2idx = {t: i+1 for i, t in enumerate(tags)}
tag2idx["PAD"] = 0

# Vocabulary Key:tag_index -> Value:Label/Tag
idx2tag = {i: w for w, i in tag2idx.items()}
```

Fig. 7: words and tags labeling

B.3. Defining and training the character embedding model

The trick here is, to wrap the parts that should be applied to the characters in a `TimeDistributed` layer to apply the same layers to every character sequence.

```
from keras.models import Model, Input
from keras.layers import LSTM, Embedding, Dense, TimeDistributed, Dropout, Bidirectional
from keras_contrib.layers import CRF

# Model definition
input = Input(shape=(MAX_LEN,))
model = Embedding(input_dim=n_words+2, output_dim=EMBEDDING, # n_words + 2 (PAD & UNK)
                  input_length=MAX_LEN)(input)  # default: 20-dim embedding
model = Bidirectional(LSTM(units=50, return_sequences=True,
                           recurrent_dropout=0.1))(model)  # variational biLSTM
model = TimeDistributed(Dense(50, activation="relu"))(model)  # a dense layer as suggested by neuralNer
crf = CRF(n_tags+1)  # CRF layer, n_tags+1(PAD)
out = crf(model)  # output

model = Model(input, out)
model.compile(optimizer="rmsprop", loss=crf.loss_function, metrics=[crf.accuracy])

model.summary()
```

Fig. 8: LSTM Model

We can also display the model summary:

```
  warnings.warn('CRF.loss_function is deprecated '
/usr/local/lib/python3.6/dist-packages/keras_contrib/layers/crf.py:35
  warnings.warn('CRF.accuracy is deprecated and it '
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 75)                0
_________________________________________________________________
embedding_1 (Embedding)      (None, 75, 20)            703600
_________________________________________________________________
bidirectional_1 (Bidirection (None, 75, 100)           28400
_________________________________________________________________
time_distributed_1 (TimeDist (None, 75, 50)            5050
_________________________________________________________________
crf_1 (CRF)                  (None, 75, 18)            1278
=================================================================
Total params: 738,328
Trainable params: 738,328
Non-trainable params: 0
_________________________________________________________________
```

Fig. 9: LSTM Model Summary

Now look at some predictions.

```
1 # Evaluate the LSTM Model:
2 pred_cat = model.predict(X_te)
3 pred = np.argmax(pred_cat, axis=-1)
4 y_te_true = np.argmax(y_te, -1)
```

Fig. 10: LSTM Model Evaluation

```
/usr/local/lib/python3.6/dist-packages/sklearn/metrics/_classification.py:1272
  _warn_prf(average, modifier, msg_start, len(result))
              precision    recall  f1-score   support

       B-art       0.00      0.00      0.00        32
       B-eve       0.00      0.00      0.00        29
       B-geo       0.81      0.89      0.85      3795
       B-gpe       0.95      0.92      0.93      1610
       B-nat       0.00      0.00      0.00        16
       B-org       0.82      0.65      0.72      2008
       B-per       0.84      0.72      0.77      1677
       B-tim       0.91      0.84      0.88      1998
       I-art       0.00      0.00      0.00        22
       I-eve       0.00      0.00      0.00        29
       I-geo       0.79      0.73      0.76       742
       I-gpe       0.00      0.00      0.00        24
       I-nat       0.00      0.00      0.00         1
       I-org       0.78      0.71      0.75      1663
       I-per       0.85      0.83      0.84      1700
       I-tim       0.78      0.71      0.75       607
           O       0.98      0.99      0.99     88930
         PAD       1.00      1.00      1.00    254817

    accuracy                           0.99    359700
   macro avg       0.53      0.50      0.51    359700
weighted avg       0.99      0.99      0.99    359700
```

Fig. 11: Classification Report

The architecture is straightforward. We notice, that this kind of model performs much better on rare or unknown words.

6. Deployment

To make our project and ideas a reality, we intented to deploy all these models in a web application. We call it *"Al Bayan"*

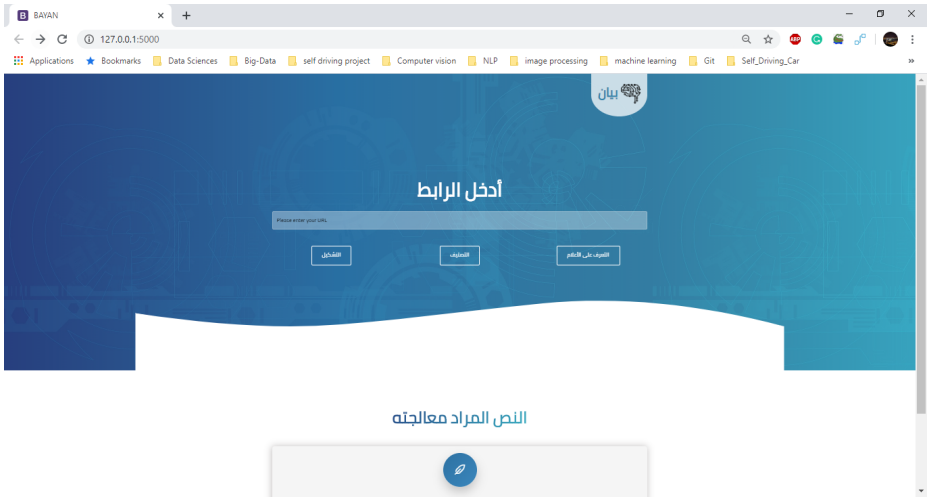


Fig. 12: Al Bayan Application