لا غالب إلا الله

Helwan University

Learning meters of Arabic and English poems with recurrent neural networks

Members, alphabetically:
Abdaullah Ramzy
Ali Abdemoniem
Ali Osama
Taha Magdy
Umar Mohamed

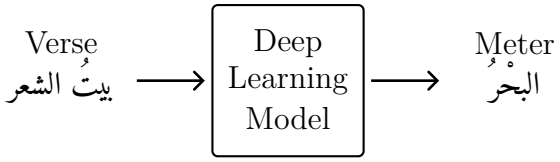
Supervisor:
Prof. Waleed A.Yousuf

May 24, 2018

1 Introduction and Problem Statement

Detecting the meter of poems is not an easy task for ordinary people, but how computers will perform? Our task is to train a model so that it can detect the meter of the input verse/text. We have worked on Arabic and English in parallel, everything thing is applied to Arabic is applied also in English, as possible as we can.

To be clearer, the model's input is a verse/text بيت شعر and the output is a class which is the verse's meter البحر, as shown in the figure below.

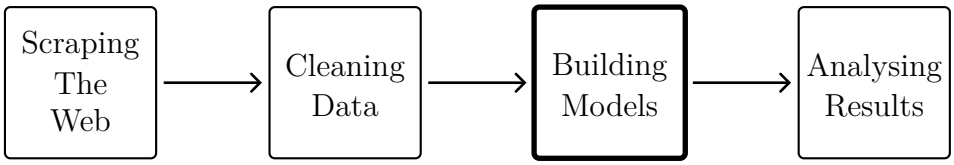


The output variable is a class/categorical, then our problem can be described as *supervised learning classification*. We have trained some deep learning models such as LSTM, Bi-LSTM and GRU. Those models are chosen because of the nature of our problem. We were trying to detect the verse's meter, which is a sequence of characters and *recurrent neural network* are suitable to learn that pattern, thanks to its cell's share-memory and its recursive structure.

2 The Project Road Map

This is are four-stage project. The first one is where we get the raw data and put it is a feature-response form, the second stage is cleaning the data, by that we mean that any non-letter character and unnecessary white-spaces are removed, also, handling any diacritics issue, as it will be demonstrated in the next sections, this stage include encoding the data to the form that is suitable to be fed to our neural networks.

The third stage is where the models are built and are tuned then we have automated the experiments to run all the iterations one after another, much details will be presented. Finally, we gather the results and analyse them, also we then conduct some additional experiments to see the language and encoding effect over the learning curve. The first two steps were much difficult than building the models. The following figure shows the road map.



Gathering Data

3 Tools

Python is pseudo-code like programming language, it is so easy and high-level that we can describe complex structures in a few lines of code, the main second reason is that python recently has been so papular in the Artificial Intelligence community. Its library is so rich with packages for Machine Learning, Deep Learning, data manipulation, even for web-scraping; we don't need to parse HTML by your hands.

We have used: Two columns:

- *Python* 3.6.5
- *Tensorflow* x.x as back-end of Keras.
- *Keras* x.x for deep learning.
- *BeautifulSoup* for web scraping.

4 Scraping The Web *–gathering the data*

To train our models we need dataset of poems. For Arabic, this kind of dataset are not so popular so you can hardly find a dataset for poetry. So, we had to create our own. During our search we have found two big poetry web sites, that contain tons of metrical-classified poems. And similar is happened for English. So we have scrapped those web sites. *Scraping* is to write a script to browse the target web site, and copy the specific content and dump it to our *csv* file. This was the most difficult step in the hole project.

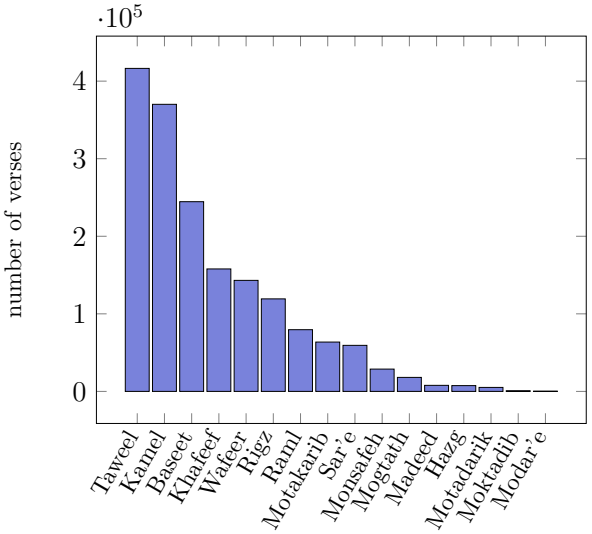
Also, we present a very large Arabic dataset that can be used for further research purposes, next sub-section contains much details about the Arabic dataset. For the English dataset, the situation was worse than scraping big complex web site, because the there were not big web site that contain a large amount of metrical-classified poems, so the English dataset is to small if it is compared to its Arabic counterpart, but this what we have found after

an extensive search. cite(Farren) had been generously granted his data from the Stanford English literature department, we have tried to contact them through their email but none has reposed.

The scraping scripts are under the directory `repository_root/scraping the web`, accompanied by a thorough self-contained `README.md` file, which contains everything you may need to use or re-use the code, even the code is written to be re-used with a lot of *in-line* documentation.

4.1 Arabic dataset

We have scrapped the Arabic dataset from two poetry websites: الديوان[1], الموسوعة الشعرية[2]. Both are merged into one large dataset. The total number of verses is 1,862,046 poetic verses; each verse is labeled by its meter, the poet who wrote it, and the age which it was written in. There are 22 meters, 3701 poets and 11 ages; and they are Pre-Islamic, Islamic, Umayyad, Mamluk, Abbasid, Ayyubid, Ottoman, Andalusian, era between Umayyad and Abbasid, Fatimid and modern. We are only interested in the 16 classic meters which are attributed to *Al-Farahidi*, and they are the majority of the dataset with a total number of 1,722,321 verses[3]. The following chart which clarifies the sizes of the classes.



[1] *aldiwan.net*

[2] *poetry.tcaabudhabi.ae*

[3] https://wwww.github.com/tahamagdy

4.2 English dataset

The English dataset is scraped from many different web resources[4]. It consists of 199,002 verses, each of them is labeled with one of these four meters: *Iambic*, *Trochee*, *Dactyl* and *Anapaestic*. The *Iambic* class dominates the dataset; there are 186,809 *Iambic* verses, 5418 *Trochee* verses, 5378 *Anapaestic* verses, 1397 *Dactyl* verses. We have downsampled the *Iambic* class to 5550 verses.

5 Cleaning Data

Data was not clean enough, there were some non-alphabetical characters and many unnecessary white spaces inside the text, they have been removed. In addition, there were diacritics mistakes, like the existence of two consecutive *harakat*, we have only kept one and have removed the other, or a *haraka* comes after a white space, it has been removed.

6 Data preparing

As a pre-encoding step, we have factored both of *shadda* to two letter the first carries ◌ْ and the second carries حركة, and *tanween* to additional نْ, by this step we shorten the encoding vector, and save more memory.

6.1 Data Encoding

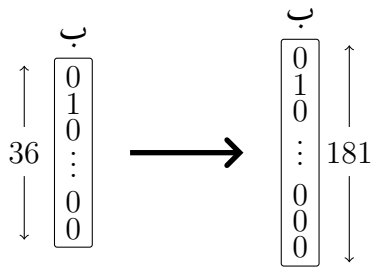
Generally, a character will be represented as an n vector. Consequently, a verse would be an $n \times p$ matrix, where n is the character representation length; n varies from one encoding to another, we have used 3 different techniques to encode the characters, and p is the verse's length.

There is an issue; we will take the *one-hot* encoding as an example to demonstrate it. A character without diacritics is represented as a 37×1 vector, where 37 is the 36 letters in addition to a white-space character, so a phrase like مرحبا having 5 letters is encoded as a 37×5 matrix if it came with diacritics it would be represented as 41×1 vector, where 41 is 37 characters + 4 diacritics, therefore a phrase like مَرْحَبَا which has 5 letters

[4]http://www.eighteenthcenturypoetry.org

and 4 diacritics, would be encoded as a 41×9 matrix. The issue comes from considering diacritics as stand-alone characters while encoding, a phrase with diacritics has more characters than if it was without diacritics, this leads to different number of time steps in the *RNN* cell and different input matrix dimensions which in turn leads to changing the model architecture, we do not want models architectures to differ whether diacritics exist of not in a verse so that we be able to conduct fair performance comparisons between models. As a solution we have encoded both a letter with its diacritic as one character, there are 36 letters and 36×4 combinations between each letter and each diacritic, then we have 181 characters including the white-space. From now forward, we have a 181-character alphabet, according to it, we will encode verses.

So, we used 180×1 vector to represent a character in order to avoid the previous two problems, see the upcoming figure and figure 1a.



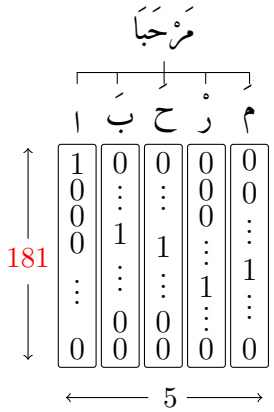
One-Hot Vector: from 36×1 to 181×1

6.2 One-Hot encoding

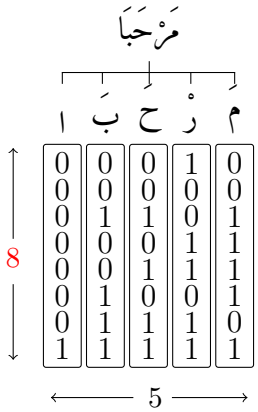
Here, a character is represented by an $n \times 1$ *one-hot* vector, where n is the alphabet length, in *Arabic* it is the 181-character alphabet, and in *English* is the 26-letter alphabet.

6.3 Binary Encoding

The idea is to represent a character with an $n \times 1$ vector which contains a unique combination of ones and zeros. $n = \lceil \log_2 l \rceil$ where l is the alphabet length, and n is the sufficient number of digits to represent l unique binary combinations. For example a phrase like this مَرْحَبا, it has 5 characters, figure



(a) *One-Hot* encoding



(b) *Binary* encoding

Figure 1: 1a Encoding the word مَرْحَبَا using n-letter alphabet *one-hot* encoding; the *one-hot* is applied to encode Arabic and English verses; the only difference is the vector length is 181 in Arabic and is 26 is English. 1b Encoding the same word using *binary* encoding, where the vector length $n = \lceil \log_2 l \rceil$, $l \in \{181, 26\}$ is the alphabet length (Arabic/English).

?? shows how it is encoded as a 8×5 matrix, which saves more memory than the *one-hot* and reduces the model capacity, significantly. But on the other hand, the input characters share some features between them due to the binary representation as it is shown in figure **??**. We will make a comparison between both encoding techniques *binary* and *one-hot*, at the end.

6.4 Two-Hot encoding

This is an intermediate technique which takes the advantages of the previous two encoding techniques. In which we encode the letter and its diacritic separately using *one-hot* encoding, this way the letter is encoded as 37×1 *one-hot* vector and the diacritic is encoded as 4×1 *one-hot* vector, then both vectors are stacked to form one 41×1 vector. By this way, we reduces the vector dimension from 180 to 41 and also minimizes the number of shared features between vectors to maximum 2 ones at each vector.

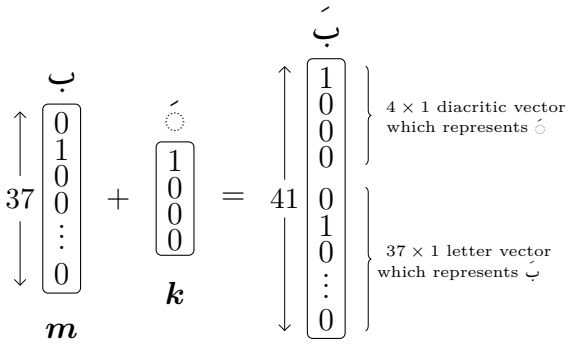


Figure 2: By stacking $\boldsymbol{k}_{4\times1}$ on the top of $\boldsymbol{m}_{37\times1}$ we get the $lolo_{41\times1}[\,{}^{k}_{m}\,]$, which repersents a letter and its diacritic, simultaneously.

7 Building models

8 Analysing Results