Learning meters of Arabic and English poems with recurrent neural networks

Members, alphabetically:
Abdaullah Ramzy
Ali H. El-Kassas
Ali Osama
Taha Magdy
Umar Mohamed

Supervisor:
Assoc. Prof. Waleed A. Yousuf

أَلا اعْطونِي بَيانَاتٍ سئِمْتُم   لَهُنَّ مُصَنَّفا فلقد بنينا

June 20, 2018

Abstract

People can easily determine whether a piece of writing is a poem or prose, but only specialists can determine which meter a poem is belonged to. In the present paper, we build a model that can classify poems according to their meters; a forward step towards machine understanding of Arabic language.

A number of different *deep learning* models are proposed for poem meter classification. As poems are sequence data, then *recurrent neural networks* are suitable for the task. We have trained three variants of them, *LSTM*, *GRU* and *Bi-LSTM* with different architectures. Because meters are nothing but sequences of characters, then we have encoded the input text at the *character-level*, so that we preserve the information provided by the letters succession directly fed to thep models. In addition, we introduce a comparative study on the difference between binary and one-hot encoding in terms of their effect on the learning curve. We also introduce a new encoding technique called *Two-Hot* which merges the advantages of both *Binary* and *One-Hot* techniques.

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.

Verse بيت الشعر ⟶ Deep Learning Model ⟶ Meter البحر

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.

The Project Road Map

This is are four-stage project.

1. The first one is where we get the raw data and put it in a feature-response form.

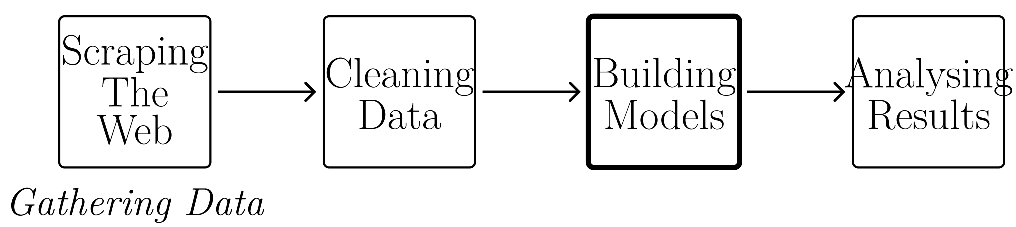
2. The second stage is cleaning the data, by which we mean that any non-letter character and unnecessary white-spaces are removed, also, handling any diacritics issue, as it will be demonstrated in the next sections, this stage include

encoding the data to the form that is suitable to be fed to our neural networks.

3. 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.

4. 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.



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
- *Keras* 2.1.5 for deep learning.
- *Tensorflow* 1.7.0 as back-end of Keras.
- *BeautifulSoup* for web scraping.

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 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. Farren [4] 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.

Introduction

Arabic Language

Arabic is the fifth most widely spoken language[1]. It is written from right to left. Its alphabet consists of 28 primary letters, and there are 8 more derived letters from the basic ones, so the total count of Arabic characters is 36 characters. The writing system is cursive; hence, most letters join to the letter that comes after them, a few letters remain disjoint.

Each Arabic letter represents a consonant, which means that short vowels are not represented by the 36 characters, for this reason, the need of *diacritics* rises. *Diacritics* are symbols that comes after a letter to state the short vowel accompanied by that letter. There are four diacritics ◌َ ◌ُ ◌ِ ◌ْ which represent the following short vowels /*a*/, /*u*/, /*i*/ and *no-vowel* respectively, their names are *fat-ha, dam-ma, kas-ra and sukun* respectively. The first three symbols are called *harakat*. Table 1 shows the 4 diacritics on a letter. There are two more sub-diacritics made up of the basic four to represent two cases: the first case is when a letter is doubled where the first letter is ac-

[1] *according to the 20th edition of Ethnologue, 2017*

companied by *sukun* and the second letter is accompanied by *haraka*, instead of doubling each letter accompanied by their *diacritics* explicitly, both of them are written once accompanied by *shadda* ◌ّ, *shadda* states that its letter is doubled, the first letter is always accompanied by *sukun*, the second letter's *haraka* is followed by *shadda*; for example, دْدَ is written دَّ. The second case, Arabs pronounce the sound /*n*/ accompanied *sukun* at the end the indefinite words, that sound corresponds to this letter نْ, it is called *noon-sakinah*, however, it is just a phone, it is not a part of the indefinite word, if a word comes as a definite word, no additional sound is added. Since it is not an essential sound, it is not written as a letter, but it is written as *tanween* ◌ٍ ◌ً ◌ٌ. *Tanween* states the sound *noon-sakinah*, but as you have noticed, there are 3 *tanween* symbols, this because *tanween* is added as a diacritic over the last letter of the indefinite word, last letter is accompanied by one of the 3 *harakat*, the last letter's *harakah* needs to be stated in addition to the sound *noon-sakinah*, so *tanween* is doubling the last letter's *haraka*, this way the last letter's *haraka* is preserved in addition to stating the sound *noon-sakinah*; for example, رَجُلُ نْ + is written رَجُلٌ and نْ + رَجُلِ is written رَجُلٍ. Those two cases will help us to reduce the dimension of the letter's feature vector as we will see in *preparing data* section.

Diacritics are just to make short vowels clearer, but they are not necessary. Moreover, a phrase without full diacritics or with just some on some letters is right linguistically, so it is allowed to drop them from text.

In Unicode, Arabic diacritics are standalone symbols, each

Diacritics	*without*	*fat-ha*	*kas-ra*	*dam-ma*	*sukun*
Shape	د	دَ	دِ	دُ	دْ

Table 1: *Diacritics on the letter* د

of them has its own unicode. This is in contrast to the Latin diacritics; e.g., in the set *{ê, é, è, ë, ē, ĕ, ě}*, each combination of the letter *e* and a diacritic is represented by one unicode.

Attempts To Define Poetry

Poetry is the other way of using language. Perhaps in some hypothetical beginning of things it was the only way of using language or simply was language tout court, prose being the derivative and younger rival. Both poetry and language are fashionably thought to have belonged to ritual in early agricultural societies; and poetry in particular, it has been claimed, arose at first in the form of magical spells recited to ensure a good harvest. Whatever the truth of this hypothesis, it blurs a useful distinction: by the time there begins to be a separate class of objects called poems, recognizable as such, these objects are no longer much regarded for their possible yam-growing properties, and such magic as they may be thought capable of has retired to do its business upon the human spirit and not directly upon the natural world outside.

Formally, poetry is recognizable by its greater dependence on at least one more parameter, the line, than appears in prose composition. This changes its appearance on the page; and it seems clear that people take their cue from this changed appearance, reading poetry aloud in a very different voice from their habitual voice, possibly because, as Ben Jonson said, poetry “speaketh somewhat above a mortal mouth.” If, as a test of this description, people are shown poems printed as prose, it most often turns out that they will read the result as prose simply because it looks that way; which is to say that they are no longer guided in their reading by the balance and shift of the line in relation to the breath as well as the syntax.

That is a minimal definition but perhaps not altogether uninformative. It may be all that ought to be attempted in the way of a definition: Poetry is the way it is because it looks that way, and it looks that way because it sounds that way and vice versa.

Arabic Poetry الشعر العربى

Arabic poetry is the earliest form of Arabic literature. It dates back to the sixth century. Poets have written poems without knowing exactly what rules which make a collection of words a poem. People recognize poetry by nature, but only talented ones who could write poems. This was the case until *Al-Farahidi* (718 – 786 CE) has analyzed the Arabic poetry, then he came up with that the succession of consonants and vowels produce patterns or *meters*, which make the music of poetry. He has counted them fifteen meters. After that, a student of *Al-Farahidi* has added one more meter to make them sixteen. Arabs call meters بحور which means ”*seas*”.

A poem is a collection of verses, a verse looks like the following:

ألا قاتل الله الحمامة غدوةً على الأيكِ ماذا هيّجتْ حين غنّتِ

تغَنّت بصوت أعجميّ فهيجت من الوَجد ما كانت ضُلوعى أجنّتِ

A verse, known as *bayt* in Arabic, consists of two halves; a half is called a *shatr*[2]. *Al-Farahidi* has introduced *al-'arud* العروض[3]; it is the study of poetic meters, in which he has laid down rigorous rules and measures, with them we can determine whether a meter of a poem is sound or broken. A meter is an ordered sequence of *feet*. Feet are the basic units of meters, there are eight of them. A Foot consists of a sequence of consonant and vowels. Traditionally, feet are represented by mnemonic words called *tafa'il* (تفاعيل). According to *al-Farahidi* and his student, there are sixteen combinations of *tafa'il*. A meter appears in a *verse* twice; each *shatr* carries the same complete meter.

For example, the following *shatr* وَيُسأَلُ في الحَوادِثِ ذو صَوابٍ is equivalent to the *meter* مفاعلتن مفاعلتن فعول, which means it belongs to *Al-Wafeer* meter. We can get the pattern of the *sukun* and *harakat* by replacing each feet by the corresponding code in table 2, which produces the following pattern that should be read from right to left:

0/0// 0///0// 0///0//

[2]it is a singular in arabic, but for simplicity we will use it for both singular and plural.

[3]it is often called the *Knowledge of Poetry*.

Feet	**Scansion**
فَعُولُنْ	0/0//
فَاعِلُنْ	0//0/
مُسْتَفْعِلُنْ	0//0/0/
مَفَاعِيلُنْ	0/0/0//
مَفْعُولَاتَ	0//0///
فَاعِلَاتُنْ	0/0//0/
مُفَاعَلَتُنْ	0///0//
مُتَفَاعِلُنْ	0//0///

Table 2: The eight feet. Every digit represents the corresponding diacritic over each latter in the feet. / If a letter has got *harakat* (◌َ ◌ُ ◌ِ), 0 if a letter has got *sukun* (◌ْ). Any *mad* (و, ا, ى) is equivalent to *sukun*.

This is a very brief introduction to *Arud*, many details are reduced.

Meter Name	**Meter** *feet combination*
al-Wafeer	مُفَاعَلَتُن مُفَاعَلَتُن فَعُولُن
al-Taweel	فَعُولُنْ مَفَاعِيلُنْ فَعُولُنْ مَفَاعِلُنْ
al-Kamel	مُتَفَاعِلُنْ مُتَفَاعِلُنْ مُتَفَاعِلُنْ
al-Baseet	مُسْتَفْعِلُنْ فَاعِلُنْ مُسْتَفْعِلُنْ فَاعِلُنْ
al-Khafeef	فَاعِلَاتُنْ مُسْتَفْعِلُنْ فَاعِلَاتُنْ
al-Rigz	مُسْتَفْعِلُنْ مُسْتَفْعِلُنْ مُسْتَفْعِلُنْ
al-Raml	فَاعِلَاتُنْ فَاعِلَاتُنْ فَاعِلَاتُنْ
al-Motakarib	فَعُولُنْ فَعُولُنْ فَعُولُنْ فَعُولُنْ
al-Sar'e	مُسْتَفْعِلُنْ مُسْتَفْعِلُنْ مَفْعُولَاتُ
al-Monsafeh	مُسْتَفْعِلُنْ مَفْعُولَاتُ مُسْتَفْعِلُنْ
al-Mogtath	مُسْتَفْعِلُنْ فَاعِلَاتُنْ فَاعِلَاتُنْ
al-Madeed	فَاعِلَاتُنْ فَاعِلُنْ فَاعِلَاتُنْ
al-Hazg	مَفَاعِيلُنْ مَفَاعِيلُنْ
al-Motadarik	فَاعِلُنْ فَاعِلُنْ فَاعِلُنْ فَاعِلُنْ
al-Moktadib	مَفْعُولَاتُ مُسْتَفْعِلُنْ مُسْتَفْعِلُنْ
al-Modar'e	مَفَاعِيلُنْ فَاعِلَاتُنْ مَفَاعِيلُنْ

Table 3: *The sixteen Arabic poem meters*

Arabic sound system

The sound system of Arabic is very different from that of English and the other languages of Europe. It includes a number of distinctive guttural sounds (pharyngeal and uvular fricatives) and a series of velarized consonants (pronounced with accompanying constriction of the pharynx and raising of the back of the tongue). There are three short and three long vowels (/a/, /i/, /u/ and /ā/, /ī/, /ū/). Arabic words always start with a single consonant followed by a vowel, and long vowels are rarely followed by more than a single consonant. Clusters containing more than two consonants do not occur in the language.

Arabic shows the fullest development of typical Semitic word structure. An Arabic word is composed of two parts: (1) the root, which generally consists of three consonants and provides the basic lexical meaning of the word, and (2) the pattern, which consists of vowels and gives grammatical meaning to the word. Thus, the root /k-t-b/ combined with the pattern /-i-ā-/ gives kitāb 'book,' whereas the same root combined with the pattern /-ā-i-/ gives kātib 'one who writes' or 'clerk.' The language also makes use of prefixes and suffixes, which act as subject markers, pronouns, prepositions, and the definite article.

Metre and rhyme

The recording of the earliest-known Arabic poetry provided future generations with examples of recitations by bards of 7th- or 8th-century versions of poems whose original composition and performance date back perhaps centuries. The collections reveal an already elaborate prosodic system, the earliest phases in the development of which remain substantially unknown.

The various types of poem are marked by particular patterns of rhyme and syllabic pulse. Each line is divided into two half-lines (called miṣrāᶜ); the second of the two ends with a rhyming syllable that is used throughout the poem. In order that the listening audience may internalize the rhyme that is to be used, the first line (which is often repeated) uses the rhyme at the end of both halves of the line; thereafter the rhyme occurs only at the end of the complete line.

English poetry Introduction

English poetry dates back to the seventh century. At that time, poems were written in *Anglo-Saxon*, also known as *The Old English*. Many political changes have influenced the language until it becomes as it is nowadays. English prosody was not formalized rigorously as a stand-alone knowledge, but many tools of the *Greek* prosody were borrowed to describe the English prosody, tools like the Greek meters types which pre-dates the English language by a long time.

A *syllable* is the unit of pronunciation having one vowel sound, with or without surrounding consonants. English words consist of one or more syllables. For example the word "Water" /ˈwɔːtə/ consists of two phonetic syllables: /ˈwɔː/ and /tə(r)/. As you notice, each syllable have only one vowel sound. Syllables can be either stressed or unstressed which are referred to by / and ×, respectively. In previous "Water" example, the first syllable is stressed, stresses are shown using the primary stress symbol ˈ in phonetics, the second syllable is unstressed, so the word "Water" is a stressed-unstressed word,

Feet	**Stresses Combination**
Iamb	$\times/$
Trochee	$/\times$
Dactyl	$/\times\times$
Anapest	$\times\times/$
Pyrrhic	$\times\times$
Amphibrach	$\times/\times$
Spondee	$//$

Table 4: Every foot is a combination of stressed and unstressed syllables, where stressed syllable is denoted by $/$ and unstressed syllable is denoted by $\times$.

which can be denoted by $/\times$, abstracting the word as stressed and unstressed syllables. There are seven different combinations of stressed and unstressed syllables form make the seven poetic *feet*s. They are shown in table 4. Meters are described as a sequence of feet. English meters are *qualitative* meters; which are stressed syllables coming at regular intervals. A meter is repeating one of the previous seven feet one to eight times, for every verse, then a verse's meter is determined by the repeated foot. If the foot is repeated once, then verse is *monometer*, if it is repeated twice then it is *dimeter* verse, until *octameter* which means a foot is repeated eight times. Here is an example, (stressed syllables are bold).

That **time** of **year** thou **mayst** in **me** be**hold**

The first verse belongs to *Iambic* foot and it is repeated five times; so it is *Iambic pentameter*.

Origins And Basic Characteristics

English belongs to the Indo-European family of languages and is therefore related to most other languages spoken in Europe and western Asia from Iceland to India.

The parent tongue, called Proto-Indo-European, was spoken about 5,000 years ago by nomads believed to have roamed the southeast European plains. Germanic, one of the language groups descended from this ancestral speech, is usually divided by scholars into three regional groups: East (Burgundian, Vandal, and Gothic, all extinct), North (Icelandic, Faroese, Norwegian, Swedish, and Danish), and West (German, Dutch and Flemish, Frisian, and English). Though closely related to English, German remains far more conservative than English in its retention of a fairly elaborate system of inflections. Frisian, spoken by the inhabitants of the Dutch province of Friesland and the islands off the west coast of Schleswig, is the language most nearly related to Modern English. Icelandic, which has changed little over the last thousand years, is the living language most nearly resembling Old English in grammatical structure.

Modern English is analytic (i.e., relatively uninflected), whereas Proto-Indo-European, the ancestral tongue of most of the modern European languages (e.g., German, French, Russian, Greek), was synthetic, or inflected. During the course of thousands of years, English words have been slowly simplified from the inflected variable forms found in Sanskrit, Greek, Latin, Russian, and German, toward invariable forms, as in Chinese and Vietnamese. The German and Chinese words for the noun man are exemplary. German has five forms: Mann, Mannes, Manne, Männer, Männern. Chinese has one form: ren. English stands in between, with four forms: man, man's, men, men's. In English, only nouns, pronouns (as in he, him, his), adjectives (as in big, bigger, biggest), and verbs are inflected. English is the only European language to employ uninflected adjectives; e.g., the tall man, the tall woman, compared to Spanish el hombre alto and la mujer alta. As for verbs, if the Modern English word ride is compared with the corresponding words in Old English and Modern German, it will be found that English now has only 5 forms (ride, rides, rode, riding, ridden), whereas Old English ridan had 13, and Modern German reiten has 16.

In addition to the simplicity of inflections, English has two other basic characteristics: flexibility of function and openness of vocabulary.

Flexibility of function has grown over the last five centuries as a consequence of the loss of inflections. Words formerly distinguished as nouns or verbs by differences in their forms are now often used as both nouns and verbs. One can speak, for example, of planning a table or tabling a plan, booking a place or placing a book, lifting a thumb or thumbing a lift. In the other Indo-European languages, apart from rare exceptions in Scandinavian languages, nouns and verbs are never identical because of the necessity of separate noun and verb endings. In English, forms for traditional pronouns, adjectives, and adverbs can also function as nouns; adjectives and adverbs as verbs; and nouns, pronouns, and adverbs as adjectives. One speaks in English of the Frankfurt Book Fair, but in German one must add the suffix -er to the place-name and put attributive and noun together as a compound, Frankfurter Buchmesse. In French one has no choice but to construct a phrase involving the use of two prepositions: Foire du Livre de Francfort. In English it is now possible to employ a plural noun as adjunct (modifier), as in wages board and sports editor; or even a conjunctional group, as in prices and incomes policy and parks and gardens committee. Any word class may alter its function in this way: the ins and outs (prepositions becoming nouns), no buts (conjunction becoming noun).

Openness of vocabulary implies both free admission of words from other languages and the ready creation of compounds and derivatives. English adopts (without change) or adapts (with slight change) any word really needed to name some new object or to denote some new process. Words from more than 350 languages have entered English in this way. Like French, Spanish, and Russian, English frequently forms scientific terms from Classical Greek word elements. Although a Germanic language in its sounds and grammar, the bulk of English vocabulary is in fact Romance or Classical in origin.

English possesses a system of orthography that does not always accurately reflect the pronunciation of words; see below Orthography.

Literature review

Classifying and detecting poems problem has been addressed and formalized differently across the literature. Moreover, the history is so rich of poetry analysis studies, hundreds of years ago, even before computer appears. However, the topic is still unexplored, computationally.

Abuata and Al-Omari [1] present the most related work to our topic. They classify Arabic poems according to their *meters*. But they have not addressed it as a *learning problem*, they have designed a deterministic five-step *algorithm* for analysing and detecting meters.

1. The first step and the most important is having the input text carrying full diacritics, this means that every single letter must carry a diacritic, explicitly.

2. The next step is converting input text into *Arud writing*[4] using if-else like rules.

3. Then metrical *scansion* rules are applied to the *Arud writing*, which leaves the input text as a series of zeros and ones.

4. After that each group of zeros and ones are defined as a *tafa'il* 2, so now we have a sequence of *tafa'il*.

5. And finally the input text is classified to the closest meter to the *tafa'il* sequence 3.

82.2% is the classification accuracy on a relatively small sample, only 417 verse. There are few well designated algorithms for detecting the meter of Classical Arabic poem. Those algorithms as explained in the literature are either complicated

[4]It is a pronounced version of writing; where only pronounced sounds are written.

and use database or/and not well defined and tested. The proposed algorithm presented above computes the correct meter of verses with high accuracy (82%). That algorithm is implemented to find classical Arabic poetry meter (Buhūr). The algorithm utilizes the complete verse) / bayt) writing styles and characteristics to identify the type of meter that represents the verse. The proposed algorithm consists of five main steps. These steps covert the input poetry into Arud Writing in order to find the correct suitable meter that represents the verse. The algorithm is based on a set of well defined rules used through the algorithm steps. The most important part of the algorithm is the Arud Writing part. Here we have many rules to apply and according to these rules we have to add or remove different letters. The algorithm only needs to rewrite the first part (sadr (of the poem verse and not the whole verse The algorithm was tested with a set of verses from different classical Arabic poems and we tried to choose poems that cover all meters types. The results showed a high level of accuracy with 82meters.

Alnagdawi et al. [3] has taken a similar approach to the previous work, but they formalized the *scansion*, *Arud* and some lingual[5] rules as *context-free grammar* and *regular expression* templates, the result is 75% correctly classified from 128 verses.

Kurt and Kara [6] have worked on detection and analysis of *arud* meter in Ottoman Language. They have depended on Ottoman *aurd* rules to construct an algorithm that analyses

[5]like pronounced and silent rules, which is directly related to *harakat*

Ottoman poems. First Ottoman text must be transliterated to Latin transcription alphabet (LTA) after that text is fed to the algorithm which uses a database containing all Ottoman meters to compare the detected meter extracted from LTA to the closest meter found in the database.

Both Abuata and Al-Omari [1] and Alnagdawi et al. [3] have common problems. The first problem is that the test size cannot give an accurate performance for the algorithms they have constructed, because it is very small. And a 75% total accuracy of 128 verses is even worse. The second problem is that the operation of converting verses into zeros and ones patterns is probabilistic; it also depends on the meaning, which is a source of randomness. Then treating such a problem as a deterministic problem is not going to be satisfying. Moreover, it results in numerous limitations like obligating verses to have full diacritics on every single letter, before conducting the classification.

Here is a different approach to the previous, the algorithmic ones, Almuhareb et al. [2] has used machine learning to recognize modern Arabic poems inside documents. He has built *Naive Bayes* and *Decision Tree* classifiers which detect poems based on the visual features, like line length average, line length standard deviation, average number of block[6], standard deviation of block number, word repetition rate, diacritic rate, punctuation rate. Those features have been extracted from 2067 documents which are divided into 513 modern poems and 1554 prose. Then classifiers have been evaluated using *10-fold cross-validation*. The best accuracy 99.81% has been achieved

[6] *a block: is a group of lines separated by an empty character or more.*

by the *decision tree* classifier which is trained on all features together.

As a reault they, built several classifier using the Decision Tree and Naive Bayes algorithms. The first classifier was a F-Measure decision tree using all of the features. This classifier achieved the best overall accuracy, 99.81%. The remaining classifiers were all Naive Bayes. Table I shows the results of these classifiers using different sets of features. The best accuracy was achieved using only the visual features at 99.71%. The all feature classifier scored a slightly lower performance of 99.61%. On the other hand, the linguistic features achieved only 87.13%. The baseline accuracy is 75.18% based on the majority class in the dataset. The results for the single feature classifiers varied widely. The best single feature classifier is the classifier that was built using the "Block Average Number of Lines" feature, 97.58%. This is a very excellent performance; about 2% less than the best achieved result. The best linguistic feature is "Diacritic Rate", 82.44%. Almost all of the remaining single feature classifiers performed below or equal to the baseline. Even though the "Block Average Number of Lines" feature achieved a very high accuracy, the effect of removing it from the all features set is negligible, at an accuracy of 98.94%.

Table 1. Naive Bayes Classifiers Performance Using Different Sets of Features

Features	Accuracy (%)	Precision(Weight ed Avg.)	Recall(Weight edAvg.)	F-Measure (Weighted Avg.)
(-) Block average number of lines	98.94	0.989	0.989	0.989
All Features	99.61	0.996	0.996	0.996
Average Line length	66.67	0.848	0.667	0.687
Diacritic rate	82.44	0.565	0.824	0.645
Line repetition rate	7518	0.565	0.752	0.645
Linguistic Features	87.13	0.908	0.871	0.878
Block average number of lines	97.58	0.978	0.976	0.976
Punctuation rate	49.49	0.803	0.495	0.5
Rhyme rate	77.70	0.874	0.777	0.76
SD of Line length	68.41	0.857	0.684	0.704
SD of block number of lines	53.12	0.776	0.531	0.548
Visual features	99.71	0.997	0.997	0.997
Word repetition rate	75.18	0.565	0.752	0.645

The F-Measure scores were similar. The ”Block Average Number of Lines” feature achieved the best F score for a single feature (97.6%). The excellent performance of the ”Block Average Number of Lines” visual feature is attributed to that poems are usually arranged in multiple blocks of text (stanzas) with a bunch of short lines (verses). While plain unformatted text articles are normally organized as one or more blocks of less but longer lines of text representing paragraphs. In the dataset, the average value of this feature is 10.7 lines for poems, and 10.2 lines for article documents. For item list documents, this value is 13.4. Fig. 7 shows the class distribution using the ”Block Average Number of Lines” and ”Average Line Length” features. Four other features achieved good precision rates, but with reduced recall: Rhyme Rate, Punctuation Rate, Average and SD of Line Length. This shows that rhymes and punctuations are good indicators for poems, however, in some poems, they may not be emphasized enough. Fig. 8 and 9 shows the poem and non-poem distribution in the rhyme and punctuation feature space. The same applies to the line length features. Poem lines tend to be short and with low standard deviation in most cases but not always. The mean of the average and SD of line length of poems in the dataset is 23.6 and 8.5 characters, respectively, compared to 390.4 and 375.7 characters for non-poems. Fig.10 shows the distribution in the line length space. The precisions of the remaining four features were low, namely, Diacritic Rate, SD of Block Number of Lines, Line and Word Repetition Rates. These features, found to be not very indicatives for poems. On the other hand, the Diacritic Rate feature achieved a good recall score (82.4%) but the low precision score (56.5%) is due to having a similar usage rate of diacritics in poem and non-poem samples in the dataset.

Tizhoosh and Dara [9] has presented similar work to Almuhareb et al. [2], have trained *Naive Bayes* and *Decision Tree* using visual features, they reached accuracy above 90%.

There is a point here, visual features may work when detecting poems inside documents due to poems are written in specific structure which distinguishes them from other text inside documents. In theses approaches models have no clue about the real patterns that create poems, of course the way how words are structured inside text does not produce a poems, at all.

Tanasescu et al. [8] has worked on binary classifying English poems where *metric* and *free-verse* are the categories, he faced an interesting problem with their dataset, it was imbalanced, (871 metrical poems, 4115 free-verse), for this reason they have used *bootstrap aggregating* (also known as *bagging*), which is a meta-algorithm that can greatly improve decision tree accuracy. With *J48* and *bootstrap aggregating*, he was able to achieve a 94.39% correctly classifying poetry as metrical or not.

By the way,*Bootstrap aggregating* , also called bagging, is a machine learning ensemble meta-algorithm designed to improve the stability and accuracy of machine learning algorithms used in statistical classification and regression. It also reduces variance and helps to avoid overfitting. Although it is usually applied to decision tree methods, it can be used with any type of method. Bagging is a special case of the model averaging approach.

As a results, binary classification experiments (metered or not me- tered) were better than the baseline. The unbalanced test experiment resulted in an accuracy of 94.39% (Figure 10), which was significantly better than ZeroR (82.53%) but only marginally better than OneR (92.10%). The two balanced test results (from over and under sampling) were not signif- icantly different from the original test (less than 1% differ- ence between the three). Classification may improve with more advanced classification algorithms and may improve further still as we continue to add more training data to our corpus. The Scandroid was more accurate at determining lexical stress and parsing text into metrical feet than the work done by the Stanford Literary Lab. It may be possible to train a neural network in a way similar to the work done by Hay- ward (but using more modern methods) to further improve accuracy and introduce sensitivity to different levels of stress (not just binary encoding). There are known issues with the Scandroid, including lex- ical ambiguity and phrasal verbs (Hartman 2005). Lexical ambiguity affects the pronunciation of certain words depending on whether they are verbs or nouns. Hartman uses “con VICT” (verb) and “CON vict” (noun) as an example. This ambiguity could be lessened by including contextual information from a POS (Part of Speech) tagger. Phrasal verbs are lexical units that operate as a single word but do not look that way, e.g. “put out”, “put down” or “put away”. Stress tends to be evenly distributed over the verb and gram- matical structure following it. A POS tagger could also be used to identify phrasal verbs and adjust stress appropriately. Finally, our implementation of the Scandroid is missing the ability to identify promoted stress. Source code cleanup should make that easier to implement. Rhyme detection can be improved and extended by adding the ability to detect internal/nonterminal rhymes and rhymes spread out over multiple words (known as mosaic rhymes), e.g. “Poet” and “know it”. If a word is not listed in the cmudict, we used the double metaphone algorithm by Lawrence Philips (Philips 1990) and spelling as crude backups. A more sophisticated approach may involve the use of a computer text to speech system.

Encoding the categorical variables and its impact on neural network performance

Encoding features has an impact on the neural network performance. Potdar et al. [7] has done a comparative study on six encoding techniques. We are interested in the comparison of *one-hot* and *binary*. They have used Artificial Neural Network for evaluating cars based on seven ordered qualitative features. The accuracy of the model was the same in both encodings—*one-hot* and *binary*.

Motivation

Our work in poems classification problem proposed with a lot of new methods and techniques. In this paper we are target to acheive the human experts performance to classify the poems which is not presented in such way or performance before.

- We introduced Poem classifications as a deep learning problem, not an algorithmic problem nor rule-based to utilize the feature in a poem that it is a sequence of character and how can we apply RNN with LSTM to classify it.

  1. The size of the dataset for each encoding.
  2. The learning curve and the time is taken to learn the problem.
  3. Classification performance results: The results we got into this paper is not achieved into any of the previous work in this field before with the variety of options or in machine learning techniques.
  4. Runtime performance.

- We used different hyperparameters for each data encoding and provide comparative study based on our experiments.

- We worked on big datasets which are not represented by such amount of data before to show the effect and provide accurate and reliable results for our models.

- Our data was not balanced so, we introduce how we solve this problem using custom weighted loss function and also, compare it with the normal one. Moreover, we tried to remove the small classes to check the effect on the total accuracy for the RNN model. All the above experiments provided with comparative study and performance recommendation results.

- We applied the same techniques to the Arabic and English and provided the experiments results for both and show how we can apply it to any other language and text in general.

Datasets

English dataset

The English dataset is scraped from many different web resources[7]. 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.

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

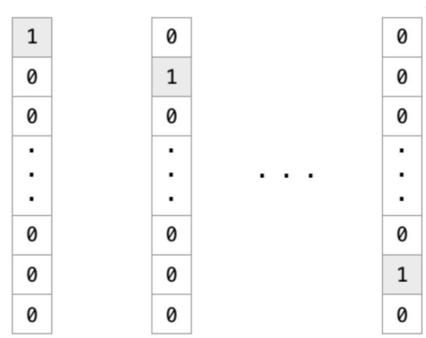
Preparing Data

Data Cleaning

For both Arabic and English data, they were not clean enough, there were some non-alphabetical characters and many unnecessary white spaces inside the text, they have been removed. For Arabic, 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.

As a pre-encoding step, we have factored both of *shadda* and *tanween* to two letters, as it previously presented in Arabic language introduction, by this step we shorten the encoding vector, and save more memory.

Encoding



Ever heard about One-Hot Encoding and its magic? Here you have it: this is a design matrix where you keep the first factor instead of removing it (how simple!).

To put it clear, just check the picture as it talks for itself better than 1,000 words.

In addition to thinking about what One-Hot Encoding does, you will notice something very quickly:

You have as many columns as you have cardinalities (values) in the categorical variable. You have a bunch of zeroes and only few 1s! (one 1 per new feature) Therefore, you have to choose between two representations of One-Hot Encoding:

Dense Representation: 0s are stored in memory, which ballons the RAM usage a LOT if you have many cardinalities. But at least, the support for such representation is typically… worldwide. Sparse Repsentation: 0s are not stored in memory, which makes RAM efficiency a LOT better even if you have millions of cardinalities. However, good luck finding support for sparse matrices for machine learning, because it is not widespread (think: xgboost, LightGBM, etc.).

A one hot encoding allows the representation of categorical data to be more expressive.

Many machine learning algorithms cannot work with categorical data directly. The categories must be converted into numbers. This is required for both input and output variables that are categorical.

We could use an integer encoding directly, rescaled where needed. This may work for problems where there is a natural ordinal relationship between the categories, and in turn the integer values, such as labels for temperature ‘cold’, warm’, and ‘hot’.

There may be problems when there is no ordinal relationship and allowing the representation to lean on any such relationship might be damaging to learning to solve the problem. An example might be the labels 'dog' and 'cat'

In these cases, we would like to give the network more expressive power to learn a probability-like number for each possible label value. This can help in both making the problem easier for the network to model. When a one hot encoding is used for the output variable, it may offer a more nuanced set of predictions than a single label.

Model

Our experiments depend on *LSTM* introduced by Hochreiter and Urgen Schmidhuber [5]; and *Bi-LSTM*, which is two *LSTM*s stacked on top of each other. LSTM is designed to solve the *long-term dependency* problem. In theory *RNN*s are capable of handling long-term dependencies, but in practice they don not, due to *exploding gradient* problem. Where weights are updated by the gradient of the loss function with respect to the current weights in each epoch in training. In some cases the gradient maybe small, vanishingly! this prevent the weights from changing and may stop the neural network from further learning. *LSTM*s overcome that problem.

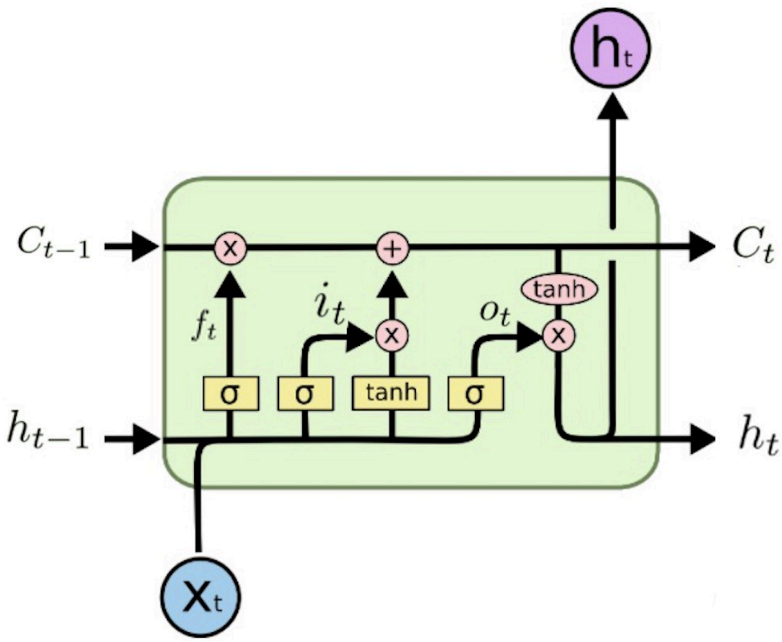


Figure 1: LSTM Cell

Figure 1 [8] shows an LSTM unit. f_t is the forgetting gate, i_t is the input gate, o_t is the output gate, C_t is the memory across cells. W_j,U_j, b_j are the weight matrices and bias vector, $j \in \{f,i,o\}$. The cell's hidden representation h_t of x_t is computed as the following:

$$
\begin{aligned}
f_t &= \sigma(W_f x_t + U_f h_{t-1} + b_t) \\
i_t &= \sigma(W_i x_t + U_i h_{t-1} + b_i) \\
o_t &= \sigma(W_o x_t + U_o h_{t-1} + b_o) \\
C_t &= f_t \circ c_{t-1} + i_t \circ tanh(W_c x_t + U_c h_{t-1} + b_c) \\
h_t &= o_t \circ tanh(c_t)
\end{aligned}
$$

[8] figure is inspired by http://colah.github.io/posts/2015-08-Understanding-LSTMs

We have conducted many experiments using LSTM and BiLSTM with different architectures. As it is shown in the table **??**, the dataset is imbalanced, there is a huge gap between the big and the small classes. Our training set-up is as the following. We built a model then we fed the data. There are operations performed on the data before it is fed to the model. Each operation is a set of options. The first operation is the encoding, we have 3 encoding techniques {One-Hot, Binary, Two-Hot}. The second operation is whether we drop the diacritics or we keep it {With diacritics, Without diacritics}. The third operation is whether we drop the last 5 classes or we keep them {Eliminate data, Full data}, by Eliminated data we mean dropping the last 5 tiny classes so that we have 11 classes, and by Full data we mean keeping the 16 classes. The reason of eliminating the last 5 classes is the gap between them and the rest, in addition this helps us in studying imbalanced data training. This leads to the last operation which is weighting the loss function by

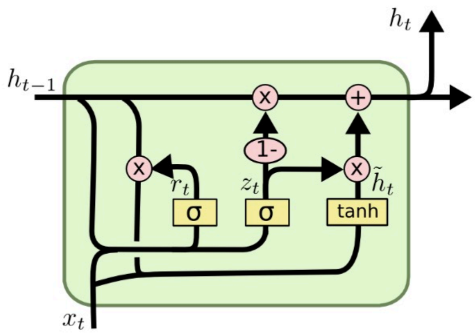
$$w_c = \frac{1/n_c}{\sum_c 1/n_c},$$

where n_c is the sample size of class c, $c = 1, 2, ...C$, where C is the number of classes. Weighting the loss this way keeps the following:

1. the density is constant (for normalization).
2. the smaller the n_c the larger the w_c.
3. $\sum_c w_c = 1$

The total number of the experiments is the Cartesian product of all the previous options sets. The same approch is taken in for the English poems, the operations does not include Eliminated/Full and Weighted/Not-Weighted.

For Enlgish, we have use used another recurrent unit, we have used Gated Recurrent Unit.



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

A recurrent neural network (RNN) is a class of artificial neural network where connections between nodes form a directed graph along a sequence. This allows it to exhibit dynamic temporal behavior for a time sequence. Unlike feedforward neural networks, RNNs can use their internal state (memory) to process sequences of inputs. This makes them applicable to tasks such as unsegmented, connected handwriting recognition or speech recognition.

The term "recurrent neural network" is used indiscriminately to refer to two broad classes of networks with a similar general structure, where one is finite impulse and the other is infinite impulse. Both classes of networks exhibit temporal dynamic behavior. A finite impulse recurrent network is a directed acyclic graph that can be unrolled and replaced with a strictly feedforward neural network, while an infinite impulse recurrent network is a directed cyclic graph that can not be unrolled.

Both finite impulse and infinite impulse recurrent networks can have additional stored state, and the storage can be under direct control by the neural network. The storage can also be replaced by another network or graph, if that incorporates time delays or has feedback loops. Such controlled states are referred to as gated state or gated memory, and are part of long short-term memorys (LSTMs) and gated recurrent units.

As mentioned above, GRUs are improved version of standard recurrent neural network. But what makes them so special and effective? To solve the vanishing gradient problem of a standard RNN, GRU uses, so called, update gate and reset gate. Basically, these are two vectors which decide what information should be passed to the output. The special thing about them is that they can be trained to keep information from long ago, without washing it through time or remove information which is irrelevant to the prediction.

Future Work

In this paper, we introduced several ways and methods to classify Arabic and English Poems. We provide some methods and techniques to work with such problem based on the character level and analyze the text concerning musical way. Also, we have published our datasets to be open source for the community to encourage the community of research into Artifical intelligence to continue from our original works here. We believe the problem and our datasets can be used in the below future works.

- Enhance the classification results to be same as the human expert.
- Use the current datasets to classify the poem meaning as this paper did not work for this idea.
- Analyze the historical impact based on the Poem and the Poetry for example for a specific period Is the Poem affected by this period, or there are patterns of writing between the Poetry or not.
- Can we generate from each class some poem similar to the poetry poems?

Bibliography

[1] Belal Abuata and A Al-Omari. A Rule-Based Algorithm for the Detection of Arud Meter in Classical Arabic Poetry. *researchgate.net*. URL https://www.researchgate.net/profile/Belal_Abuata/publication/298199192_A_Rule-Based_Algorithm_for_the_Detection_of_Arud_Meter_in_Classical_Arabic_Poetry/links/5790817108ae64311c0ff2fd.pdf.

[2] Abdulrahman Almuhareb, Waleed A Almutairi, Haya Altuwaijri, Abdulelah Almubarak, and Marwa Khan. Recognition of Modern Arabic Poems. 10(4):454–464, 2015. ISSN 1796217X. doi: 10.17706/jsw.10.4.454-464.

[3] Mohammad a Alnagdawi, Hasan Rashideh, and Ala Fahed. Finding Arabic Poem Meter using Context Free Grammar. *J. of Commun. & Comput. Eng.*, 3(1):52–59, 2013.

[4] Derek Farren. Deep Causal Inference for Average Treatment Effect Estimation of Poems Popularity. pages 1–6.

[5] Sepp Hochreiter and J Urgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780,

1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL http://www7.informatik.tu-muenchen.de/{~}hochreit{%}5Cnhttp://www.idsia.ch/{~}juergen.

[6] Atakan Kurt and Mehmet Kara. An algorithm for the detection and analysis of arud meter in Diwan poetry. *Turkish Journal of Electrical Engineering and Computer Sciences*, 20(6):948–963, 2012. ISSN 13000632. doi: 10.3906/elk-1010-899.

[7] Kedar Potdar, Taher S., and Chinmay D. A Comparative Study of Categorical Variable Encoding Techniques for Neural Network Classifiers. *International Journal of Computer Applications*, 175(4):7–9, 2017. ISSN 09758887. doi: 10.5120/ijca2017915495. URL http://www.ijcaonline.org/archives/volume175/number4/potdar-2017-ijca-915495.pdf.

[8] Margento Chris Tanasescu, Bryan Paget, and Diana Inkpen. Automatic Classification of Poetry by Meter and Rhyme. *Florida Artificial Intelligence Research Society Conference*, 5, 2016. URL https://www.aaai.org/ocs/index.php/FLAIRS/FLAIRS16/paper/view/12923.

[9] H. R. Tizhoosh and R. A. Dara. On poem recognition. *Pattern Analysis and Applications*, 9(4):325–338, 2006. ISSN 14337541. doi: 10.1007/s10044-006-0044-8.