A Simple Approach to Unify Ambiguously Encoded Kurdish Characters

Sardar Jaf
The University of Durham
sardar.jaf@durham.ac.uk

Abstract

In this study we outline a potential problem in the normalisation stage of processing texts that are based on a modified version of the Arabic alphabet. The main source of resources available for processing resource-scarce languages is raw text. We have identified an interesting challenge that must be addressed when normalising certain natural language texts. Many less-resourced languages, such as Kurdish, Farsi, Urdu, Pashtu, etc., use a modified version of the Arabic writing system. Many characters in harvested data from the Internet may have exactly the same form but encoded with different Unicode values (ambiguous characters). It is important to identify ambiguous characters during the normalisation stage of most text processing tasks. We will demonstrate cases related to ambiguous Kurdish and Farsi characters and propose a semi-automatic approach to identifying and unifying ambiguously encoded characters.

1. Introduction

One of the main challenges in processing less-resourced languages is the lack of natural language processing (NLP) tools and resources.

Large numbers of languages use a modified version of the Arabic writing system, such as Kurdish, Farsi, Urdu, Pasthtu, etc. We have observed a situation where characters of these languages have exactly the same form but encoded differently. The problem with the inconsistent encoding of some characters (ambiguous characters) is they are treated as different characters. This makes large numbers of similar words, which are similar in meaning and form, to be treated as completely different words. This situation is evident in most languages that use a modified version of Arabic script. In this paper, we attempt to shed light on ambiguous characters, which results in generating multiple instances of words of similar forms but different encodings. Moreover, we will show an approach for identifying ambiguous characters and an approach for correcting them by appropriately unifying their Unicode values. We will mainly focus on Kurdish but we show the applicability of our work to other related languages such as Farsi.

This paper is organised as follow: in section 2 we present a brief overview of Kurdish and in Section 3 we highlight some of the general challenges in processing Kurdish. While in Section 4 we outline situations where inconsistencies in character encoding could generate multiple words unnecessarily, in section 5 we describe our approach to identifying and unifying Unicode values of characters of equal forms. In section 6 we briefly describe the applicability of our approach to processing other related languages and we conclude our paper in Section 7.

2. A Brief Overview of Kurdish

Kurdish belongs to the Indo-European language family and it is closely related to Farsi. Kurdish is spoken by approximately 25 to 30 million people, but the exact number varies depending on the source. Despite the fact that large numbers of people speak Kurdish, the language is considered as a resource-scarce language.

There are several dialects in Kurdish, such as Sorani, Kurmanji, Zazaki, Hawrami, Gorani, etc. However, the main two dialects are Sorani and Kurmanji. These two dialects differ in many ways, one of the main differences is the writing style. Sorani uses a modified version of Arabic scripts while Kurmanji uses a modified version of Latin scripts. The use of a modified version of Arabic alphabet poses an interesting challenge in processing Sorani text, where this challenge is applicable to other related languages that use Arabic alphabet. Although our focus is to unify the encoding of Kurdish Sorani we will show that our approach could be applied to other related languages, such as Farsi. In the following section we will highlight some of the major challenges in processing Kurdish text.

3. The Challenges of Processing Kurdish

Two of the challenges in processing Kurdish are the dialect diversity and script diversity. However, the main challenge that we address in this paper is about unifying the Unicode values of some characters that are similar in form (ambiguous characters). Before describing our approach it is worth highlighting some aspects of the dialect and script diversity of Kurdish so that we can demonstrate the character ambiguity with examples.

3.1. Dialect Diversity

In Kurdish, there are several dialects. Two of the main dialects are Sorani and Kurmanji. These dialects differ in a number of ways. The effect of the differences is that developing an NLP tool for one dialect is not easily applicable to another dialect, hence the tasks for developing any NLP applications for Kurdish is twice more compared to working on other languages, therefore we state that our solution has been applied to Kurdish Sorani dialect only. Below is a short list of some of the main differences between Sorani and Kurmanji dialects (Esmaili, 2012; MacKenzie, 1961; McCarus, 1958):

- Gender distinction: both gender (feminine:masculine) is retained in Kurmanji while it is ignored in Sorani.
- Case Opposition: Kurmanji uses case opposition (absolute:oblique) for nouns and pronouns while Sorani ignores them but uses the pronominal suffixes to take over the function of the case.
- For past tense transitive verbs, Sorani uses pronominal enclitics, because of the absence of oblique pronoun, while Kurmanji uses the full ergative alignment.
- Sorani verb morphology is used for constructing passive and causative while in Kurmanji the verb هاتن (*hatin* “to come”) and دان (*dan* “to give”) are used respectively.
- The definite suffix ەکە- (*-eke*, “the”) is used only in Sorani.

3.2. Script Diversity

One of the major differences between Sorani and Kurmanji is the writing system. Kurmanji uses a modified version of Latin alphabet while Sorani uses a modified version of Arabic alphabet. The script diversity makes it difficult to bijectively construct a mapping between Sorani and Kurmanji in many cases (Gautier, 1998). Some of the one-to-many mappings between the two writing systems are demonstrated in Table 1.

As can be noted from Table 1 (a) multiple Latin letters could be mapped to one Arabic letter. Similarity in Table 1 (b) the mapping from the Arabic-based letter {ە} to the Latin-based letters is a one-to-many mapping, where mapping to any of H, h, E, or e, is not a trivial process. The same situation applies to mapping letter {و } to U, u, W, or w and letter {ی} to Î, î, Y, or y. The mapping shows that multiple Latin letters may potentially be mapped to an Arabic letter. This paper is to identify multiple Unicode values that are assigned to Arabic-based letters that have the same form and identify a way to unify them. Table 1 shows a list of identified characters with different Unicode values, however we anticipate that there may be other characters that did not appear in our dataset.

Unicode Value	Latin-based letters	Arabic-based letters	Unicode Value
u0048	H	ه	u06BE u06D5 or u0647
u0068	h		
u0049	I	-	-
u0069	i		
u0055	U	و	u0648
u0075	u		
u0057	W	و	u0648
u0077	w		
u0059	Y	ی	u06CC
u0079	y		

a) Mapping from Latin-based to Arabic-based

Unicode Value	Arabic-based letters	Latin-based letters	Unicode Value
u06BE u06D5 or u0647	ه	H	u0048
		h	u0068
		E	u0057
		e	u0077
u0648	و	U	u0055
		u	u0075
		W	u0045
		w	u00EA
u06CC	ی	Î	u00CE
		î	u00EE
		Y	u0059
		y	u0079

b) Mapping from Arabic-based to Latin-based

Table 1. Mapping between Arabic-based and Latin-based of Kurdish Alphabets (Esmaili, 2012).

From Table 1 we can see that the letter {ه} constitutes one letter (H, h, E, e) which is pronounced as either */ha/* or */a/* depending on its location in the word. If it appears at the start of a word it forms {هـ} or in some cases if it appears in the middle it forms {ـهـ}, and in both cases it is pronounced as */ha/*, however it may be assigned different Unicode values. If it appears at the end of a word it forms { ـه}or in isolation it forms {ه} and in both cases it constitutes /E/ or /e/. In addition to these two cases, in most electronic texts, it may appear as a *zero-width non-joiner (zwnj)* character, which prevents joining a character from its follower (Esmaili, 2012). For example, in the word بارهەڵگرەکە (*barHelgreke,* "The goods carrier") it constitutes */H/* in the fourth position, it constitutes */e/* in the fifth position, and it constitutes a *zwnj* character in position nine in the word. For the same letter (i.e., the letter { ه}) different Unicode values are often used. For example when it appeared in position four in the word its Unicode value was 06BE but in some cases it is assigned 0647, when it appeared in position five and nine its Unicode value was either 0647, 06BE, or 06D5. This inconsistent encoding makes large numbers of words lose their unique form. Table 2 contains examples of different words that have the same forms but different Unicode values. This kind of ambiguity has also been observed in Urdu (Bajwa et al, 2011).

The problem that we are going to address is related to identifying ambiguous characters (i.e., characters that have the same form but different Unicode values) and unify their Unicode values. The solution to this problem is essential during the normalisation process of Sorani text because ignoring this problem will lead to incorrectly treating large numbers of words as unique words.

3.2.1. Other challenges of Processing Kurdish

Another challenge in processing Kurdish stems from the lack of NLP tools and resources. The unavailability of data, resources and tools for Kurdish text processing restrict developers in their approaches to processing the language. Another noticeable challenge is in segmentation and tokenisation is the identification of sentence, phrase, and word boundary. It is not possible to use spaces as a boundary sign because they may appear within a word or between words, or they may be absent between some sequential words (Esmaili, 2012; Shamsfard, 2011; Bajwa et al, 2011). Additionally, because of the absence of capitalisation the task of segmentation, tokenisation, and recognising Named Entities are further complicated.

The difficulty in processing Kurdish is further aggravated by the lack of gold-standard dataset. Although there are several dictionaries available for Kurdish annotated corpus and large datasets are still unavailable (Walther and Sagot, 2010). It is possible to use the large data available on the Internet for developing a corpus of raw text, which could be used in Information Retrieval application for example. However the harvested data from the Internet may pose a number of problems and solving the ambiguity of characters must be performed during the normalisation stage of raw text processing.

4. Dataset Collection

In this section we will describe the database that we have used for validating the appropriateness of our approach to identify ambiguous characters. Fortunately, there is a large number of Kurdish news websites, where we can harvest the required data. We have collected various data from several website.[1]

The collected dataset contains about 2,000,000 words which constitute just over 21,000 articles, which is large enough to capture a large variety of word forms. The dataset is also diverse, which includes topics covering sport, economy, politics, art, culture, health, multimedia and lifestyle, science and technology.

5. Collecting, Identifying and unifying ambiguously encoded characters

In this section we will describe the steps that we have used in identifying ambiguous words and characters in terms of their forms.

5.1. Collecting and parsing web pages

The first step of the process involved collecting over 21,000 news articles from a large number of websites. Then, we parsed each web page and removed various unwanted data, such as mark-up text, numbers, punctuations, foreign words, etc. A small challenge in this step is that although it is easy to identify Latin-based scripts in the pages, detecting Arabic or Farsi words is hard because they share the same writing system as Kurdish Sorani. A simple way to tackle this issue is to extract all unique words from the data with a specific frequency threshold. We have intentionally removed words that have occurred less than 0.001% in the data. These words were either Arabic or Farsi names, which are occasionally used in Kurdish news articles; words with incorrect spelling; and words that are accidentally merged with some other words during the parsing process of the web pages.

5.2. Identifying unique characters

Once a set of clean text is retrieved we have processed all the data and generated a lexicon, which contained unique words, and manually inspected the most frequently occurring words. At this stage, a

[1]The data are collected from the following websites: www.asoyroj.com, www.chawigal.com, www.aweza.co, www.dengekan.info, www.gulan-media.com, www.hawlati.co,, www.hawpshti.com, www.helwist.com, www.lvinpress.com, www.malmokurd.com, www.nnsroj.com, www.radionawxo.org, www.regaykurdistan.com, www.rojnews.net,www.rozhnamawany.com, www.serbexo.com, www.shaqam.net, www.shrova.org, and www.xendan.org

large number of words were treated as unique words even though they had similar forms with some other words. For example, as we have mentioned in Section 3.2. some Arabic-based characters are ambiguous. These ambiguous characters may appear in many words that are exactly the same in terms of meaning and form. Table 2 contains examples of some of the most ambiguously occurring words in the lexicon. Also, we can note from Table 2 the frequency of most ambiguous words is high.

Total words	**Frequency**	**Unicode Value**
له (*le*, “on”)	135059	u0644 u0647
له (*le*, “on”)	122063	u0644 u06D5
كه (*ke*, “as”)	125881	u0643 u06D5
كه (*ke*, “as”)	92812	u0643 u0647
كه (*ke*, “as”)	39747	u06A9 u0647
كه (*ke*, “as”)	11312	u0643 u06D5
كوردستان (*kurdistan*, “Kurdistan”)	16081	u0643 u0648 u0631 u062F u0633 u062A u0627 0646
كوردستان (*kurdistan*, “Kurdistan”)	13196	u06A9 u0648 u0631 u062F u0633 u062A u0627 0646
ئێمه (*aeme*, “us”)	4252	u0626 u06CE u0645 u0647
ئێمه (*aeme*, “us”)	4050	u0626 u06CE u0645 u06D5
هێزی (*hyz*, “its power”)	2472	u0647 u06CE u0632 u06CC
هێزی (*hyz*, “its power”)	2042	u06BE u06CE u0632 u06CC
هێزی (*hyz*, “its power”)	1674	u06BE u06CE u0632 u0649

Table 2. Ambiguous words with their frequency and Unicode values (letter 'u' is used in front of each Unicode value to distinguish different character's encoding value)

The identification of ambiguous characters in words is performed by manually inspecting the encoding values of characters in many frequently occurring words. Using the identification of unique words is time consuming and does not give an accurate account of the level of character ambiguity in the data, and it is neither efficient nor easy to locate ambiguous characters in large numbers of words.

An efficiency improvement is achieved by processing every character in every word in the lexicon and record all the unique characters along with their Unicode values. Then manually inspect the encoding of the recorded characters. However, the inefficiency aspect of this approach is it requires processing very large number of characters. For example, our dataset contained 2,983,579 words and the average word length was 6 characters, which yielded approximately 18 million characters to process. The time taken to process all the words was 56 seconds.

This approach can be improved using a very simple technique. That is, recording all the unique words in a second lexicon which does not contain duplicate words[2]. Then process the characters of the recorded unique words. The total number of unique words was dramatically reduced to 52,987 words and the processing time was reduced to 29 seconds.

Once we have identified all the unique characters and their Unicode values, which gave us a total of 37 unique characters (excluding punctuations), we have then identified three ambiguous characters, i.e., characters with the same form but different Unicode values. Those characters are shown in Table 3.

[2]Ambiguous words are treated as duplicated but are treated as unique because their Unicode values are unique even thought their forms are similar.

Characters	Unicode Values	Frequency
ه (pronounced as /ha/, /a/ and used as zero-width non-joiner character)	u06D5	2361391
	u0647	1961352
	u06BE	51442
ی (pronounced as /ye/)	u06CC	81987
	u0649	69363
	u06BE	61961
ک (pronounced as /k/)	u06A9	585728
	u0643	537621

Table 3. Ambiguous Characters

5.3. Unifying Different Unicode Values of Similar Characters

Once we identified all the unique words and manually identified the ambiguous characters, we generated a mapping dictionary that mapped each Unicode value to different Unicode value, which is shown in Table 4. The content of the mapping dictionary is simple and can be formatted in any styles.

Unicode Value	Mapped Unicode Value
u0647	u06BE or u06D5
u0643	u06A9
u0649	u06CC
u06BE	u06CC

Table 4. Mapping from one Unicode Value to Another Unicode Value

Generally, if we find a specific character with a specific Unicode value in a word then we replace it with a given Unicode value. However, as it can be noted from Table 4 the Unicode value u0647, which represents ه (a, "a"), (h, "ha"), or *zwnj* should remain as it is or be mapped to u06BE or u06D5. The location in which the character appears dictates its form. If the character was followed by a character with the same Unicode value then it's changed to u06BE. Otherwise, there are exceptional cases for correctly mapping u0647 to u06BE or u06D5: (i) if the character is final then we replace it with u06D5. (ii) if a specific vowel (with the Unicode value u06CE, u06CC, u0627, or u06c6) follows the character then it should be mapped to u06D5. (iii) If the previous two cases do not apply then it should be mapped to u06BE. The steps for finding the Unicode values of ambiguous characters are given in Figure 1.

The mapping dictionary that we have compiled contains one entry per line. In each entry there are two comma separated Unicode values (parameters), where the first parameter represents an ambiguous character in a word and the second parameter represents the Unicode value that is used for replacing the ambiguous character. In order to deal with the exceptional cases for handling u0647 Unicode value, the format of the dictionary entry for characters with u0647 Unicode value is in the following: the first parameter is u0647 Unicode value; the second parameter is the value that replaces u0647 if the character with u0647 Unicode is a final character; the third parameter is a list of *n* number of Unicode values, where n is a positive number; the last parameter is the value that is used for replacing the character with u0647 Unicode value if and only if the immediate following character is similar to Unicode values in the list of *n* Unicode values.

```
1. read words from a file and add them to a lexicon (L).
2. count the frequency of each word in L and create a create a new lexicon containing words and their frequency (LF).
  2.1. optional: sort content of LF in ascending order.
  2.2. for each word in LF, write the word and its Unicode values to a file for manual inspection.
  2.3. retrieve the characters of each word in LF.
    2.3.1. add the characters to a list (CL) if it is not in CL.
    2.3.2. write the character and their Unicode values to file if it is not in CL.
3. Inspect the characters that were written to the file in step 2.3.2 and identify n duplicate characters, where n is a predetermine number with the same form but with different Unicode values.
```

Figure 1. Finding the Unicode values of ambiguous characters

From the list of characters that we have identified by following the steps in Figure 1. we have manually inspected the characters that were of the same form but with different Unicode values. This way we have identified the characters that had the same form but different Unicode values, which resulted in duplicating a large number of words; some examples are shown in Table 2. Once we have identified all the ambiguous characters, we have compiled a mapping dictionary for replacing the Unicode values, which is shown in Table 4. The algorithm for mapping/unifying ambiguous characters is given in Figure 2. The evaluation of the solution is conducted by extracting all the unique characters and their Unicode values from the lexicon and manually inspecting them to identify a character that is similar in form to one or more character(s) but with different Unicode value. The absence of an ambiguous character indicated that all characters in the lexicon were encoded correctly.

```
For each word w is in the lexicon L do:
  For each entry uv in the mapping Unicode value dictionary do:
    Find uv in w
    If uv is in w:
      If there are more characters in w after the identified uv do:
        If the character that follows uv is the same as uv do:
          Replace uv with the second parameter in the entry
        Else do:
          If the uv is at the start of w do:
            If the length of the entry is more than 2 parameters
              If the character that immediately follows uv is in the list of special characters:
                Replace uv with the second parameter in the entry
              Else:
                Replace uv with last parameter in the entry
            Else:
              Replace uv with second parameter in the entry
          Else:
            If the length of the entry is more than 2 parameters
              If the character that immediately follows uv is in the list of special characters:
                Replace uv with the second parameter in the entry
              Else:
                Replace uv with last parameter in the entry
            Else:
              Replace uv with second parameter in the entry
```

Figure 2. Unifying ambiguous characters

6. Applying Our Approach to Related Languages

We also used the same approach on Farsi, which is closely related to Kurdish. From our experiment on Farsi we identified that in Farsi the number of ambiguous characters is fewer than those in Kurdish. For example, from Table 5 we can see that the final and medial character ي (*y,* "y") appears with different Unicode values. It is noticeable that the final ي (*y,* "y") has u06CC assignment more frequently than u06BE while a medial ـيـ (*y,* "y") is assigned u06CC Unicode value more than u06BE. Unlike in Kurdish, the character ه (a, "a") have not been assigned the Unicode value u06BE. The u0647 Unicode value is assigned to the initial, medial and final character ه (a, "a") more than u06D5 Unicode value. The third ambiguous character in Farsi was the character {ک} (*k,* "K") which was often assigned the Unicode values u06A9 instead of u0643 Unicode value. In conclusion, after applying the same technique to related languages we can identify ambiguous characters and semi-automatically correct them.

Total words	Frequency	Unicode Value
آئین	118	u0622 u0626 u06CC u0646
آئین	10	u0622 u0626 u06BE u0646
آزادی	112	u0622 u0632 u0627 u062F u06CC
آزادی	11	u0622 u0632 u0627 u062F u0649
جامعه	197	u062C u0627 u0645 u0639 u0647
جامعه	22	u062C u0627 u0645 u0639 u06D5
حاکم	183	u062D u0627 u06A9 u0645
حاکم	14	u062D u0627 u0643 u0645

Table 5. Farsi ambiguous words

7. Conclusion

The normalisation of text often involves removing unwanted texts (noise) such as foreign words, numbers, punctuations, etc. This stage of text processing is one of the main stages in processing less-resourced languages because in most cases raw data is collected from the Internet, which may contain various noise. In addition to noise removal process of online text we have identified an interesting case in processing Kurdish, and other related languages such as Farsi, where some characters of similar forms are assigned different Unicode values (ambiguous characters). We anticipate that the reason is that for languages that use a modified version of Arabic script for writing may interchangeably use different Unicode values, which could be the Unicode value of the original Arabic character or a specific code for the modified character. Another possibility is it may be due to the type of Operating Systems or the data entry devices that are used in compiling web pages, where they have different Unicode values for characters with similar forms.

Unifying ambiguous characters is an important step in the text normalisation stage because ambiguous characters, which are used for constructing words, lead to ambiguous words. In many inductive NLP processing tasks it is not plausible to induce information from noisy data. Therefore, unifying Unicode values of ambiguous characters is an essential step towards removing noise.

In this paper, we have presented a semi-automatic approach to unifying Unicode values of Kurdish text. Furthermore, we have used the same approach on Farsi and we have identified the same issue. Our experiment on Farsi shows that our approach could be applicable to other related languages, such as Urdu and Pashtu, which we aim to apply it to them in the near future.

References

Bajwa, U. I., Rehman , Z., and Anwar, W. (2011). Challenges in Urdu Text Tokenization and Sentence Boundary Disambiguation. *In Proceedings of the 2nd Workshop on South Southeast Asian Natural Language Processing (WSSANLP 2011).*

Esmaili, K. Shykh. (2012). Challenges in Kurdish Processing. *In Proceedings of the 9th Information Retreival Society Conference (AIRS 2013), Singapore.*

Gautier, G. (1998). Building a Kurdish Language Corpus: An Overview of the Technical Problems. *In Proceedings of ICEMCO 1998.*

MacKenzie, D. N. 1961. *Kurdish dialect studies, volume 1 of London Oriental Series*. Oxford University Press.

McCarus, E. N. 1958. *A Kurdish Grammar descriptive analysis of the Kurdish of Sulaimaniya Iraq*. PhD thesis. New York, USA: American Council of Learned Societies.

Shamsfard M. (2011). Challenges and Open Problems in Persian Text Processing. *In Proceedings of the 5th Language and Technology Conference (LTC 2011), Poland.*

Thackston. W. M. (1960). *Sorani Kurdish: A Reference Grammar with Selected Reading*s. Oxford University Press, UK.

Walther, G., B. and Sagot, B. (2010). Developing a large-scale lexicon for a less-resourced language: general methodology and preliminary experiments on Sorani Kurdish. *In Proceedings of the 7th SaLTMiL Workshop on Creation and use of basic lexical resources for less-resourced languages (LREC 2010 Workshop), Malta*.