YZV 405E NLP Term Project Report

Bora Boyacıoğlu
Istanbul Technical University
Artificial Intelligence and Data Engineering
150200310
boyacioglu20@itu.edu.tr

Abstract

This document is for an NLP project, aiming to fix character dialetics happening in a variety of similar languages. The process, called diacritization, will be done using an NLP model. I am, the author, Bora Boyacıoğlu, a student in Istanbul Technincal University. This is my project proposal for the specified task.

1 Introduction

With the effects of globalisation, languages began to surrender their unique features, especially their alternative letters, to English. English is considered to not have accented characters, as it is used as a base language. Also considering computer era raised in English-speaking world (USA, UK, Canada), the main character set used in the informatics (ASCII, American Standard Code for Information Interchance) is English based. Therefore Turkish and many other European languages being Latin Alphabet based, there are still very common characters which not been represented in the ASCII character set.

ASCII is used widely. From the internet addresses to SMS's (Short Message Service), a lot of systems used by billions of people used daily are only compatible with ASCII charset, with some like SMS only supporting a limited extra set in its essence (ISO 8859-1). This means people actually use non-decorative characters to write their more complex languages. This creates an issue. While a lot of words being understood easily even when using similar but different letters, some get confused easily. To give an example, Turkish has "oldum" (I have been) and "öldüm" (I was died), which are both represented as "oldum" in the ASCII set.

With the new context-based understanding applications, this is tried to be overcome using NLP techniques. In this project, my goal is too, do a similar task. I will explain the problem in detail. Though before, it is necessary to list a few example works have been done in other projects.

2 Dataset

In this task, I am using the project dataset, which is available on this Kaggle Competition. There are two data available initially: a train and a test data in CSV (Comma-Separated Values) format. These are essentially two column data where the first column reserved for the entry ID. The second, and the main column, contains Turkish sentences written in non-Turkish specific ASCII character set, in the test data. To train the model, training data provides me with the real full-charset Turkish sentences. There are 1145 sentences available in the test data. To give an example of the non-Turkish charset, in Table 1 a couple of sentences from the test data are shown.

18	diyalog icin de kapi acildi aclik grevle...
24	Israil cocuklara acimiyor Erdogan Kah...
34	kadini cahil tutmak suretiyle somurme...
55	isim secimimin otobiyografik nedenle...

Table 1: Example Sentences from Test Data

On the other side, the train data is completely normal structured, with proper letters and fully meaningful words. This will be a necessity as my model will be trained over this data. There are 40986 sentences to train the model in the train data, which is hopefully more than enough. To give an example from the training data, here are a few sentences in Table 2.

25	isterseniz hızlı arama veya detaylı ara...
46	bu iki yemek tarifimiz girit yemekleri...
66	krep hamuru bitene kadar işleme deva...
99	ilk önce hayatsal fonksiyonları neden...

Table 2: Example Sentences from Train Data

There are a few characters different betweeen the plain-English ASCII and ISO 8859-9 Turkish sets: native Turkish doesn't have *q/Q, w/W, x/X* char-

acters while ASCII not having *ç/Ç, ğ/Ğ, ı, İ, ö/Ö, ş/Ş, ü/Ü*. Of course, absence of some characters in the native Turkish isn't much of a problem since this data is already Turkish and all the charsets supporting English characters. However, there are actually 12 letters including the capitals which not have been represented in the given test dataset.

Also, I am using the Turkish Vocabulary itself, the Güncel Türkçe Sözlük (Isik, 2021) dataset from GitHub. This dataset contains the words in the Turkish Vocabulary, which is last published by Türk Dil Kurumu in 2021. It contains 99.236 words alongside their meanings. I only utilise the words themselves. Some examples are as follows in the Table 3.

1084	ünlü
3408	dağ
4958	doğululaşma
11505	seyrekleştirmek

Table 3: Example Words from Turkish Vocabulary

My main goal is trying to overcome this issue by utilizing an NLP model, which learns from the train set and will be hopefully able to work on the test set. This is, *diacritize* the words back to their original form with their original letters.

3 Measures

To evaluate the performance of both the Neural Network model and the combined rule-based approach, I used the following metrics:

1. **Word Score (Word-Level Accuracy):** This metric measures the percentage of words in the test dataset that are correctly diacritized. It is calculated as the ratio of the number of correct words to the total number of words.

$$\frac{1}{n_{\text{words}}} \sum_{i}^{\text{sents}} \sum_{j}^{\text{sents}_i} \text{preds}_{\text{words}_{i,j}} \in \text{sents}_i$$

2. **Sentence Score (Sentence-Level Subset Accuracy):** And this one measures the percentage of sentences where every word is correctly diacritized. It shows the model's ability to perfectly diacritize entire sentences (at least part of them).

$$\frac{1}{n_{\text{sents}}} \sum_{i}^{\text{sents}} \text{preds}_i \in \text{sents}_i$$

I have choosen these metrics because they provide a clear indication of how well the model performs on both word and sentence level, which is important for understanding the effectiveness of the diacritization process.

4 Related Work

There are a number of studies conducted for this task. Interestingly, most of them are specifically done for Arabic. Though, the fact that their alphabet is different than both Turkish and English, I would not prefer to base my project on an Arabic study. Instead, there are also less popular but important studies done in European languages.

One of the studies I found really interesting is "Diacritics Restoration using BERT with Analysis on Czech language" Náplava et al. (2021). Although it is based on Czech and conducted in Czech Republic, it actually includes many European languages, some more extensively than Czech. And the fact that their instruction set is the same in both Turkish and Czech data highlights its importance for this project. With 1005 as their Turkish set size and 10884 in total, they get around 98.98% accuracy for Turkish text. This is really useful for me. And even though they are utilizing a pre-trained BERT model, there are still lots of important points I will be inspiring from.

There is an other article I want to point out. It is "Diacritics correction in turkish with context-aware sequence to sequence modeling" Köksal et al. (2022). The beauty is, it is done by Turkish people and focused on the problem Turkish specific case. Also, their data includes more than 50 million words in total. This shows that the results will be more satisfying. They rely on a couple of models, one of them being ITU. It is a **deasciifier** model develop by Eryiğit (2014). Their different models combined, have more than 90% F1 scores.

Lastly, because its variety, it is crucial to mention one of the Arabic diacritization studies. I want to focus on the study "Arabic Diacritization with Recurrent Neural Networks" (Belinkov and Glass, 2015). Their approach invlolves an RNN to solve more complex problem, as they are focusing on non-Latin languages like Arabic and Hebrew. Their involvement of more than 600k words in total, and utilization of a strong model, will be very beneficial for me.

5 Methodology

5.1 Neural Network Model

In the implementation of the model, I utilised an Encoder-Decoder Seq2Seq model. The architecture starts by encoding the inputs, and finishes by decoding the outputs. In the forward pass of the Seq2Seq model, encoder and decoder are utilised.

The encoder part consists of an embedding, an LSTM (Long Short-Term Memory) nework, and a dropout layer. Then, the decoder also adds a linear layer. The LSTM memorises the indiced representations in the form of dense vector representations. In the encoder, the contextual information is saved into a state vector.

The decoder part takes the encoder's output state as its initial state and starts generating diacritised sequences. Also, a linear layer is included after the LSTM in the decoder.

For the training, after the model initialised, I used the Adam optimiser with the CrossEntropyLoss being the loss function. I also implemented early stopping to prevent overfitting, which was not used in my tiny 50 epoch training process. And the model was saved periodically when the losses decreased, to prevent epoch losses.

5.2 Rule Based Algorithms

After the model predictions, I turned to the rule based improvements, which turned out to be much more helpful in my case. I listed three cases for analysing the words.

Case 1: **The word does not need to be changed.** In this case, the word already only consists of ASCII characters. Which means it was never transformed from its original form at the beginning. These words stay as they are, and directly put into the output sentence.

$$\text{bilgisayar} \Longrightarrow \text{bilgisayar}$$

Case 2: **There is only one acronym for that word.** For the words only corresponding to one acronym, that acronym is used. This correspondance comes from the vocabulary I defined by using both the train data and the Turkish Vocabulary.

$$\text{sinif} \Longrightarrow \{\text{sınıf}\}$$

Case 3: **There are more than one acronyms.** The last case is where things happen. For words with multiple potential ways to be diacritised, the model's prediction for that sequence is called. If any of the possible acronyms is in the prediction, that one is used. Otherwise, always the one with the highest occurrence in the train data is selected.

$$\text{aci} \Longrightarrow \{\text{acı, açı}\}$$

This combined approach results in a supported score, which is much higher than the model itself. To use these cases, I created a vocabulary using both the train data and the Turkish Vocabulary data. I made a set of all the words I have, which is 150286. Then, I listed the ones that has non-ASCII chars (Case 2), which is 85486. This already eliminated the 43.12% of the words to stay in the Case 1. Later, I have listed the acronyms that correspond to words with more than one acronym, which is 1789. And I found out that at the end, only 1.19% of the whole vocabulary need to go into Case 3.

6 Experimental Results

Using my Word Score and Sentence Score measuers, I get the following results in Table 4 for the pure model and the combined approach:

Type	Word Score	Sentence Score
N.N. Model	2.60%	0.52%
Combined	81.88%	14.01%

Table 4: Pure Model and Combined Results

My results are low, especially the pure model being too low. This is due to my limitations and the largeness of the model itself. However, these are solvable things. The rule based way of solving the problem seems to have helped a lot as well.

7 Conclusions and Future Work

While acknowledging the lowness of the results, I can say that my model seems a bit overdone for the task. And actually the biggest issue was the way I was processing the data. Instead of complete sentences, working with words, and even letters could have gradually changed the accuracy. The current implementation almost completely relies on the rule based algorithms. The model itself, may have succeeded if there were more time and resources to train. However, for a task like sub-latin character diacritisation, this should not have been

necessary. Also, a couple of adjustments could be done to boost the ruled algorithms.

References

Yonatan Belinkov and James Glass. 2015. Arabic diacritization with recurrent neural networks. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 2281–2285.

Gülşen Eryiğit. 2014. ITU Turkish NLP web service. In *Proceedings of the Demonstrations at the 14th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, Gothenburg, Sweden. Association for Computational Linguistics.

Kemal Ogun Isik. 2021. guncel-turkce-sozluk: A Turkish Dictionary. `https://github.com/ogun/guncel-turkce-sozluk`. Accessed: May 13, 2024.

Asiye Tuba Köksal, Özge Bozal, and Umut Özge. 2022. Diacritics correction in turkish with context-aware sequence to sequence modeling. *Turkish Journal of Electrical Engineering and Computer Sciences*, 30(6):Article 28.

Jakub Náplava, Milan Straka, and Jana Straková. 2021. Diacritics restoration using BERT with analysis on czech language. *CoRR*, abs/2105.11408.