Exploring Spelling Correction Approaches for Turkish

Dilara Torunoğlu-Selamet, Eren Bekar, Tugay İlbay, Gülşen Eryiğit
Department of Computer Engineering
Istanbul Technical University
Istanbul, 34469, Turkey
[torunoglud, erenbekar, ilbay, gulsen.cebiroglu]@itu.edu.tr

***Abstract*—The spelling correction of morphologically rich languages is hard to be solved with traditional approaches since in these languages, words may have hundreds of different surface forms which do not occur in a dictionary. Turkish is an agglutinative language with a very complex morphology and lacks annotated language resources. In this study, we explore the impact of different spelling correction approaches for Turkish and ways to eliminate the training data scarcity. We test with seven different spelling correction approaches, four of which are introduced in this study. As the result of this preliminary work, we propose a new automatic training data collection process where existing spelling correctors help to develop an error model for a better system. Our best performing model uses a unigram language model and this error model, and improves the performance scores by almost 20 percentage points over the widely used baselines. As a result, our study reveals the achievable top performance with the proposed approach and gives directions for a better future implementation plan.**

***Keywords*—Spelling Corrector, Spell Checker, Turkish**

I. Introduction

In morphologically rich languages (MRLs) and especially the agglutinative ones like Turkish, Finnish or Hungarian, a word may occur in hundreds of different surface forms by the addition of multiple suffixes the end of a word stem. The creation of a lexicon/dictionary consisting of all possible surface forms is impractical and most of the time not efficient due to memory space and search speed constraints. As a result, the usage of a lexicon to check if the newly constructed candidate of a misspelled word is valid or not, as is the case in traditional approaches tailored for morphologically poor languages, becomes unusable for MRLs.

Finite state transducers (FSTs) [1], [2] are proven to be very well suited for this kind of languages and perform very fast lookup over possible word generations. One of the early implementations of spelling correction for MRLs is the error tolerant finite state recognition (ETFSR) approach of Oflazer [3]. Although it is very fast to create the possible candidates up to the specified edit distance limit, the deficiency of this approach is that it does not produce an ordered list of possible corrections which prevents its usage as an automated spelling corrector. Recent approaches [4]–[6] which focus on weighted finite-state spell-checking using language models and error models are very efficient for the spelling correction of MRLs. Pirinen and Lindén [6] who experiment also with some agglutinative and polysynthetic languages as well as English, use the Wikipedia articles of the related languages in order to create the corresponding language models. On the other hand, the same error models which are used for English are also used for MRLs only by adding language-specific characters.

Wang et al. [7] propose a fast and accurate approximate string search algorithm (ASS) which keeps track of the frequent mistakes (error model) extracted from training data (consisting of spelling mistakes and their corrections) and generates the most probable correction candidates. The method uses a vocabulary trie for validating the generated candidates. It is very straightforward to collect the training data for the error model from the user queries of a search engine (the suggested and selected corrections) as it is conducted in the mentioned study.

In this paper, we explore the way of creating a Turkish-specific error model for lack of manually annotated training data and the different combinations of the error model, the language model and the minimum edit distance candidate generation for spelling correction. We compare our results with 3 existing spelling correction systems for Turkish: 1. error tolerant finite state recognition (ETFSR) approach of Oflazer [3], 2. MsWord and 3. Zemberek [8].[1] The paper is structured as follows: Section 2 introduces the error model and Section 3 discusses the proposed spelling correctors, Section 4 presents the used datasets and evaluation metrics, Section 5 gives the experimental results and discussions and Section 6 the conclusion and future work.

II. The Error Model

Obtaining the error model is a challenging task considering the lack of manually annotated training data for the Turkish language. Wang et al. [7] proposed a probabilistic approach for spelling correction. This approach was novel in that it was using a log-linear candidate generation utilizing a special data structure that can find top candidates efficiently. The proposed method works effectively for languages which have a limited dictionary for lookup. They derived all the possible

[1]To the best of our knowledge, at the time of writing this paper, the only three spelling correction systems that we can compare with were these three systems.

rules from the training data using a similar approach to Brill and Moore [9]. In their study, they collected the training data for the error model from the user queries of a search engine. Despite not having this opportunity, we propose a new automatic training data collection process where the existing spelling correctors help to develop an error model. We collected a training data set from the Twitter domain. We then passed all the ill-formed words (which are not accepted by our morphological analyzer) from one online (Google[2]) and one offline spelling correctors [8] and accepted the corrections which are proposed identically by both of these correctors as the corrected form of the ill-formed words in our training set. At the end of this process, we obtained a training set of 5775 word pairs (ill-formed and corrected words) which have a character length within a range of 2 to 23. After obtaining the training set for the error model, we used the same approach with Wang et al. [7] to store the extracted error rules.

We used the Aho-Corasick tree structure for storing and applying the correction rules. During the generation of the error model, the rules are extracted from the misspelled and corrected forms of words by using the Levenshtein edit distance algorithm. The output of this part is a set of rules which includes addition, deletion and substitutions of letters. This rule set also contains the likelihood of each derived rule. The extracted rules and their estimated likelihoods are stored in an Aho-Corasick search tree which is a very efficient string matching trie-based data structure. All leaf nodes in this search tree have an output link which associates the node itself with the likelihood of the rule in the node. This lets fetching rules and their likelihoods effectively. It also stores failure links that redirect the search to the best applicable node when there is no way to continue for the queried string. This prevents us from starting from the beginning each time the search query fails and results in a significant time gain during the search.

III. Spelling Correctors

ETFSR and Zemberek use edit-distance based candidate generation approaches. The following subsections introduces our new approaches that we experiment with, which are basically the different combinations of the language and error models as well as ETFSR.

Spelling Corrector #1 (SC1)

Our first approach is an adaptation of Wang et al. [7]. Since creating a lexicon which will cover all possible surface forms in an MRL is not practical and efficient in that the required memory allocation for the data structure is very big even with the most compact data structures[3], instead of the vocabulary trie for candidate validation, SC1 uses an FST (a finite-state transducer which is built from a stem lexicon for the MRL in focus and the morphotactic and phonetic rules to generate the inflected forms of these stems) as the language validator. Figure 1 draws the main components of SC1. The training phase is the process of creating the error model which is explained in Section II. In the candidate generation phase, the previously constructed Aho-Corasick tree is looked-up for all applicable rules for a given misspelled word. Since not all the rules generate a valid surface form, the generated results should be validated by the FST. If the constructed word is validated by the FST, then all applied rule likelihoods are summed up and this forms the likelihood of the candidate word. As a pruning technique, before applying a rule, it is always checked that the rule likelihood is able to generate a more probable candidate. If not, the rule is not applied to the misspelled word.

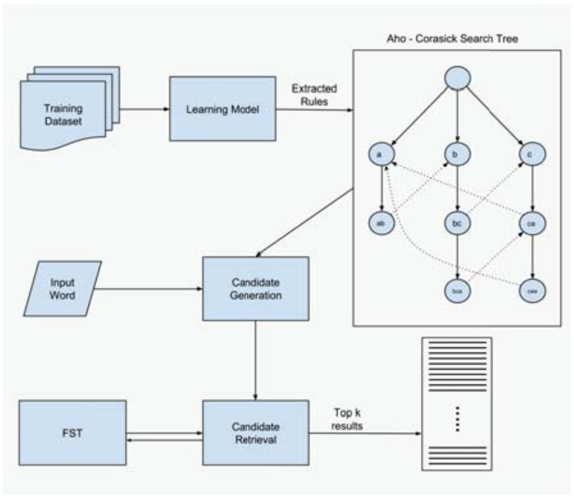

Fig. 1: Spelling Corrector #1

As a result, our approach differs from the original ASS model [7] in two main points: 1) the usage of FST for validation, 2) the calculation of the rule set probabilities in the training phase. In the original work, they employ a log-linear model for calculating the probabilities of rule sets whereas in our work, we simply use likelihoods for preliminary investigation.

Spelling Corrector #2 (SC2)

As mentioned in the introductory section, the output of ETFSR is a set of unsorted candidates and the size of the candidate list is unpredictable. SC2 is an approach to deal with this deficiency by re-ranking ETFSR outputs using the probabilities calculated from the error model as explained previously. Figure 2 shows the structure of SC2, where the misspelled inputs firstly enter the ETFSR. We then retrieve the rules (and their scores) from our rule tree that should be applied to the misspelled word to generate each candidate in the output list from the ETFSR. In other words, we get

[2]At the time of this collection process, Google spelling correction service was still available.

[3]In the early stages of our implementation, we tried to just place the most frequently occurring surface forms extracted from a corpus into the lexicon and even this approach took more than 500M of memory by using a suffix tree, which we believe is not acceptable for a spelling corrector application to be in practical usage.

the list of applied rules (which can be addition, deletion and substitution of a letter) according to the Levenshtein edit distance between the misspelled word and the corresponding candidate. When we have the rules for a candidate, we sum up the costs of the applied rules, and then simply sort the candidates by their costs. The candidate with the minimum cost is accepted as the most probable correction.

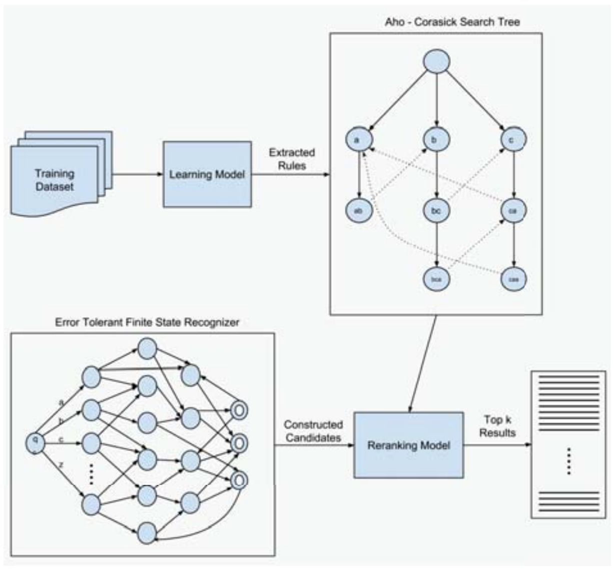

Fig. 2: Spelling Corrector #2

Spelling Corrector #3 (SC3)

Inspired by previous works by Linden and Pirinen [4]–[6], SC3 aims to make use of unigram language models for candidate sorting. To this end, a unigram language model is trained from word surface forms from a Turkish corpus. The ETFSR outputs are then re-ranked similarly to SC2 but this time using the unigram probabilities. The candidate having the highest probability and the smallest edit distance from the input misspelled word is then accepted as the produced correction. The structure of SC3 is shown in Figure 3.

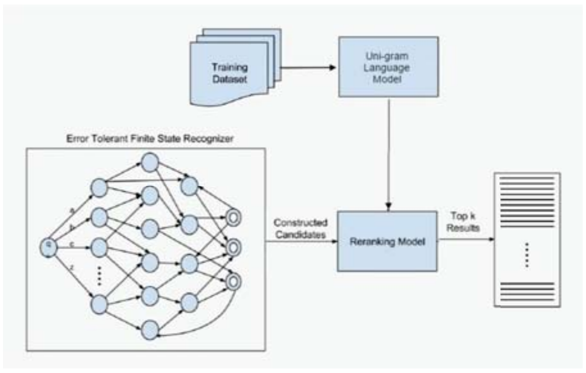

Fig. 3: Spelling Corrector #3

Spelling Corrector #4 (SC4)

SC4 is inspired from Linden and Pirinen [6], in that it uses a language and an error model together in order to generate candidates. SC4 uses the same unigram language model from SC3 and the same error model introduced in Section II. SC4 differs from SC1 in that, the generated candidates by the error model are validated by using the language model instead of the FST and the best proposal is selected as the candidate with minimum rule cost and maximum unigram probability:

$$\underset{c\epsilon Gen}{\operatorname{argmax}} \quad p(c)\frac{1}{rulecost(c)}$$

Laplace Smoothing [10] is used in order to compensate for the absence of a candidate word in the language model. SC4 is depicted in Figure 4.

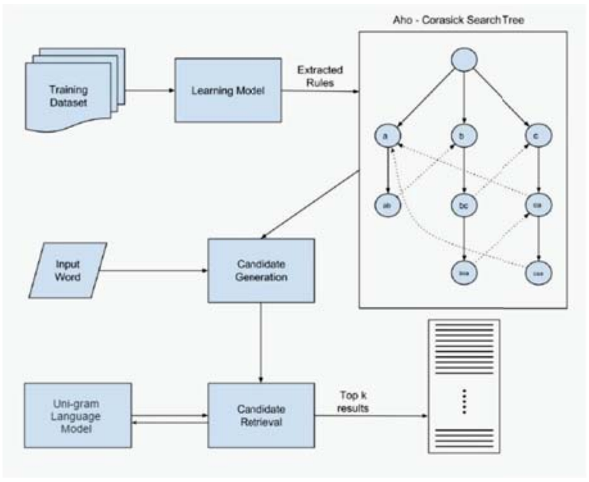

Fig. 4: Spelling Corrector #4

Table I displays the usage and combination of language and error models as well as the candidate generation method in the introduced spelling correctors. As can be noticed from the table, the difference between SC2 and SC1 is that in SC2, which uses ETFSR in its candidate generation stage, all the produced candidates are already valid words, whereas in SC1 the candidates are validated after being produced by the use of the error model. The last two spelling correctors (SC3 and SC4) using language models are the highest memory-consuming systems as expected and explained in the introductory section. They are tested both with ETFSR candidate generation (SC3) and Aho-Corasick candidate generation (SC4). SC4 also uses the error model in its probability calculation. Another possible system (discussed in the following sections) which could provide a slight increase in the scores would be a combination of ETFSR, the language model and the error model, though it was not tested as part of this study.

Models	Error Model	Language Model	Candidate Generation
SC1	✓	✗	Aho-Corasick
SC2	✓	✗	ETFSR
SC3	✗	✓	ETFSR
SC4	✓	✓	Aho-Corasick

TABLE I: Models Used in Different Approaches

IV. Experimental Setup

We tested our system on Turkish which is a highly agglutinative language carrying all the characteristics of a morphologically rich language. We used an available two-level morphological analyzer of Oflazer [11] as the FST language validator of our system in SC1 and again the ETFSR from Oflazer [3] in SC2 and SC3.

To obtain a unigram language model we used the corpus introduced by Sak et al. [12]. The text corpus compiled from the web contains about 500M tokens. Due to the composition of the data found on the web, the corpus include noisy data. We extracted only the valid Turkish words which constitute 842 MB of the corpus (almost 43M valid tokens).

During the collection of the test data, for the sake of fairness we do not include errors made on purpose due to social media writing trends such as emoticons and words that are typed out without vowels or the proper diacritics, which would be corrected in a normalization stage [13] rather than spelling correction.

The creation of the training data to train the error model is explained in Section II. Since this automatic approach is only applied during the creation of the training data used in rule extraction, this does not hamper the evaluation on our test data which is manually annotated with corrected forms (1016 word pairs).

V. Experimental Results & Discussions

In our experiments, we first test with ETFSR and the spelling correctors introduced in Section III and evaluate their results. We then compare our models with other available spelling correctors for Turkish.

Table II introduces some statistics about ETFSR and with the other models (SC1, SC2, SC3 and SC4); namely the average operation time the spelling correction approach on our test set described in previous section and the average index of the correct candidate within all possible generated candidates. The index number starts from 0 indicating that the first candidate in the output is the correct one given the manually annotated test set. One may notice from this table that the ETFSR approach produces very fast results but the correct answer generally occurs in lower positions in the produced candidate list. On the other hand SC1 is almost 10 times slower when compared to ETFSR but produces more accurate results. SC2 is much faster when compared to SC1 and has almost similar success ranges. SC3 and SC4 similar to SC2 in terms of average duration but give better average index results. The added cost due to re-ranking is smaller than a single millisecond[4] over ETFSR.

Approach	Average Duration (ms)	Average Index
ETFSR	388	1.46
SC1	3385	0.9
SC2	389	0.6
SC3	333	0.35
SC4	363	0.145

TABLE II: Output Statistics

Table III gives the comparison of the spelling correction accuracies of our models with the mentioned tools. Although, the Google spelling suggestion API was used during the creation of our training data, it could not be compared with the other spelling correctors in this section since it is no longer available. In this experiment, for all of the systems, we took the first suggestion given by that system and compared it with the gold-standard correction in our test set. Our best model outperforms the widely used Zemberek spelling corrector by almost 20 percentage points. Despite the modest size of our training data set that we were not able to continue to collect due to the unavailability of one of the services (Google spelling suggestion API) that we have used, we see that the proposed error model on its own (SC2) outperforms MsWord by more than 2 percentage points. We believe that, with the addition of extra training data, the system performance may be improved even further. As a future work, self-training approaches may be tested for the learning of the error-rule probabilities. But we observe that the used language model has a much higher impact by almost 10 percentage points. One should notice that the used language model is just a unigram surface model and better results may be obtained with more sophisticated language models.

	Accuracy
ETFSR	49.0%
Zemberek	61.4%
MsWord	66.3%
SC1	68.6%
SC2	67.8%
SC3	78.7%
SC4	80.7%

TABLE III: Comparison with previous studies

In order to investigate the results and the behavior of the algorithms more closely, we also made a different evaluation based on promoting the correct candidate appearing in the top n list of the algorithm's output. Table IV presents these scores for n=1, 3, 5 and 10., e.g. SC4 positioned the correct candidate in its top 3 list in 92.7% of the cases.

We can observe that the success rates of all the models become similar as n increases, meaning that ETFSR is also successful in generating the correct candidate in its top 10 list. But SC3 and SC4 are certainly more suited to be used

[4]The training time (629 ms with our available training data) is not added to this cost since it occurs only once in the preparation stage and the pre-trained model is only loaded at the beginning of testing stage.

Candidate List Size	ETFSR	SC1	SC2	SC3	SC4
1	49.0%	68.6%	67.8%	78.7%	80.7%
3	76.7%	88.6%	89.1%	92.7%	92.7%
5	86.2%	93.5%	92.9%	94.5%	97.0%
10	93.8%	95.7%	95.4%	95.5%	98.9%

TABLE IV: Candidate List Evaluation

as automated spelling correctors. In top 1, the difference is as high as 31,7 percentage points between ETFSR and SC4.

Although SC3 and SC4 both yield very high scores, they are both memory-inefficient due to the used surface language models. A better possible system which would be the combination of both will actually be a kind of the system proposed by Linden and Pirinen [6] combined with our automatically created error model which we aim to develop in our future work. Although the difference between candidate generation using FSTs and Aho-Corasick tree is not statistically significant[5], we expect that memory consumption will be alleviated with a better implementation, even though there may not be an increase in performance.

VI. Conclusion & Future Work

In this study, we explored ways to eliminate the scarcity of training data for spelling correction, as well as the impact of different spelling correction approaches for Turkish. We proposed a new automatic training data collection process where existing spelling correctors contribute to the development of an error model, paving the way for better systems. We explained four spelling correction approaches adapted for Turkish alternatively using language models, error models and combination of candidate generation approaches, and reported their performances for Turkish in comparison with three established spelling correctors. Our work has been a preliminary investigation of better spelling correction approaches for MRLs, and there is still much that could be further investigated and improved, such as 1) Automatically increasing training set size, 2) Integrating self-training approaches in learning error rule probabilities, and 3) Using weighted finite-state language and error models. Although we used a simple unigram language model in our best-performing systems, we observed that the systems making use of the language model outperform those without the model by about 10 percentage points. Furthermore, we believe that using weighted finite-state language and error models would produce slightly better results than the ones represented in this paper as well as eliminating the memory consumption problem of our best corrector.

Acknowledgment

This work is part of our ongoing research project "Parsing Turkish Web 2.0 Sentences" supported by ICT COST Action IC1207 TUBITAK 1001 (grant no: 112E276).

[5]We used McNemar's paired t-test to evaluate the difference between SC1(68.6%) and SC2(67.8%) and found that the difference between these two models is not statistically significant, with a two-tailed p value of 0.7.

References

[1] *Finite-state morphology: Xerox tools and techniques*, 2003.
[2] K. Lindén, M. Silfverberg, and T. Pirinen, *HFST tools for morphology–an efficient open-source package for construction of morphological analyzers*, Std., 2009.
[3] *Error-tolerant finite-state recognition with applications to morphological analysis and spelling correction*, vol. 22, no. 1, 1996.
[4] K. Lindén, T. Pirinen *et al.*, *Weighting finite-state morphological analyzers using hfst tools*, Std., 2009.
[5] T. Pirinen, K. Lindén *et al.*, *Finite-state spell-checking with weighted language and error models*, Std., 2010.
[6] T. A. Pirinen and K. Lindén, *State-of-the-Art in Weighted Finite-State Spell-Checking*, Std., 2014.
[7] Z. Wang, G. Xu, H. Li, and M. Zhang, *A fast and accurate method for approximate string search*, Association for Computational Linguistics Std., 2011.
[8] *Zemberek, an open source NLP framework for Turkic Languages*, vol. 10, 2007.
[9] E. Brill and R. C. Moore, *An improved error model for noisy channel spelling correction*, Association for Computational Linguistics Std., 2000.
[10] *Laplacian smoothing and Delaunay triangulations*, vol. 4, no. 6, 1988.
[11] *Two-level description of Turkish morphology*, vol. 9, no. 2, 1994.
[12] *Resources for Turkish morphological processing*, vol. 45, no. 2, 2011.
[13] D. Torunoğlu and G. Eryiğit, *A Cascaded Approach for Social Media Text Normalization of Turkish*, Std., April 2014.