A set of open-source tools for Turkish natural language processing

Çağrı Çöltekin
Universitiy of Groningen
c.coltekin@rug.nl

Abstract

This paper introduces a set of freely available, open-source tools for Turkish that are built around TRmorph, a morphological analyzer introduced earlier in Çöltekin (2010a). The article first provides an update on the analyzer, which includes a complete rewrite using a different finite-state description language and tool set as well as major tagset changes to comply better with the state-of-the-art computational processing of Turkish and the user requests received so far. Besides these major changes to the analyzer, this paper introduces tools for *morphological segmentation*, *stemming* and *lemmatization*, *guessing unknown words*, *grapheme to phoneme conversion*, *hyphenation* and a *morphological disambiguation*.

Keywords: Turkish Morphology, finite-state tools, morphological disambiguation

1. Introduction

This paper introduces a set of tools for Turkish natural language processing (NLP) that are built around a free, open-source morphological analyzer, TRmorph (Çöltekin, 2010a). Namely, we introduce tools for *morphological segmentation*, *stemming* and *lemmatization*, *guessing unknown words*, *grapheme to phoneme conversion*, *syllabification/hyphenation* and a *morphological disambiguation*. Similar to the current version of the morphological analyzer, all tools are freely available with the terms of GNU Lesser General Public License (LGPL).[1] The source code of the analyzer and the other tools reported in this paper can be obtained from https://github.com/coltekin/TRmorph.

The most important feature of the tools presented here is the fact that they are developed and maintained following the *open-source* software development practices. As a result, they can be customized for a particular purpose, and equally importantly, they are constantly improved by problem reports, suggestions and contributions from the users.

The rest of this article is organized as follows. The next section introduces some aspects of Turkish morphology that are relevant to the rest of the discussion in this paper. Section 3 provides an update on TRmorph, outlining important changes since it was first announced. Section 4 describes the new finite-state tools, and Section 5 discusses a simple but effective morphological disambiguation tool. Lastly, the paper is concluded with a brief summary and notes on the future directions.

2. Turkish morphology and NLP

Turkish is a morphologically complex language. The complex morphology is often demonstrated by the following popular example.

(1) İstanbul-lu-laş-tır-ama-dık-lar-ımız-dan-mış-sınız
'You are (supposedly) one of those who we could not convert to an İstanbulite'

The example above, consisting of 11 morphemes, is intended to demonstrate the complexity. It definitely is not representative of a typical word. However, it is perfectly understandable by the native speakers, and, albeit rare, words of similar size can be observed in large corpora. More importantly, the rough translation indicates that a relatively complex sentence can be expressed by a single word.

Although Turkish words can be long, the word formation is rather regular. This makes it relatively easy to implement the morphology in a finite-state formalism. However the complex word structure poses a number of other problems for natural language processing applications. First, the traditional notion of the word as a stem and a set of suffixes belonging to a paradigm does not hold. For example, in (1) above, the word class changes multiple times by suffixation. This process is quite productive, and some of these 'derivational' suffixes can be attached to already inflected words. Furthermore, parts of a word, as opposed to the complete word, can participate in syntactic relations. This requires sub-word units to be used in syntactic processing. Another practical reflection of the complex word structure is the problem of data sparseness that affects the machine learning methods applied to Turkish NLP. Since one can produce a large number of possible words forms from a single stem, most of these will never appear even in a large corpus.

The solution to these problems offered in the state-of-the-art Turkish NLP literature is to define sub-word units called *inflectional groups* (IGs, see e.g., Hakkani-Tür et al., 2002; Eryiğit et al., 2008).

In this paper, we view an analysis string as a sequence of inflectional groups. Each IG contains a *root* or a *derivational suffix*, a *category* marker, and a set of *inflections* that belong to the respective category. For example, the word *evdekininkilerde* 'in the ones (e.g., books) that belong to the one (e.g., person) at home' has the following IGs (represented using TRmorph analysis symbols, slightly simplified for readability):[2]

[1] https://www.gnu.org/licenses/lgpl.html.

[2] The definition presented here is slightly different than other definitions in the literature. We treat the derivational morphemes specially, not as part of the inflection sequence belonging to the newly formed IG.

1. ev⟨N⟩⟨loc⟩: the root *ev* 'house' with category noun ⟨N⟩ and locative inflection ⟨loc⟩
2. ⟨ki⟩⟨N⟩⟨gen⟩: the derivation ⟨ki⟩ deriving a (pro)noun followed by genitive inflection
3. ⟨ki⟩⟨N⟩⟨pl⟩⟨loc⟩: the derivation ⟨ki⟩ deriving another (pro)noun followed by plural and locative inflections.

The inflectional groups allow a natural way to express syntactic relations between sub-word units, as well as reducing the data sparseness problem in machine learning methods used in NLP.

Another problem worth noting due to complex morphology is the large number of ambiguous analyses. The problem of ambiguity will be discussed at length in Section 5.

Given the importance of the morphological analysis, it is not surprising that the development and improvement of morphological analyzers has been an active area of research for Turkish and related languages (e.g., Hankamer, 1986; Oflazer, 1994; Washington et al., 2012; Şahin et al., 2013; Kairakbay, 2013).

3. TRmorph: an update

TRmorph is an open source finite state morphological analyzer introduced in Çöltekin (2010a). Since its introduction, TRmorph has been used by many NLP researchers and practitioners. For example, it was used in information retrieval (Özdikiş et al., 2012; Hadımlı and Yöndem, 2012); for testing unsupervised morphology learners (Çöltekin, 2010b; Kılıç and Bozşahin, 2012; Janicki, 2013); in corpora annotation (Ruhi et al., 2012);[3] for preparation of experiment materials (Güneş and Çöltekin, 2014); testing finite state toolkits (Lindén, Axelson, et al., 2013); in machine translation systems (Forcada et al., 2009); and utilized at varying levels in development of analyzers for other languages, e.g., for Azeri,[4] Kyrgyz (Washington et al., 2012) and Hungarian.[5] Besides the academic use cited above, TRmorph has been in use by a number of commercial organizations, and the demo web page at present receives about 30 queries per day.

During the last four years that it has been in public use, TRmorph has been in constant change. Besides minor improvements or bug fixes in a software's life cycle, there have also been a few major changes that are worth reporting here. The rest of the present section describes these major changes.

3.1. Tagset change

The most important change visible to the users is a completely revised morphological tagset. The earlier morphological analyses consisted of the part-of-speech tag of the root and sequence of tags that correspond to the surface affixes. Although this works fine for many applications, state-of-the art (dependency) parsing methods for Turkish are based on so-called *inflectional groups*. The new tagset allows explicit identification of IGs within the analysis strings, as well as fixing some inconsistencies in the initial tagset. All morphological tags are described in the extensive manual included in the distribution.

3.2. The lexicon

Another substantial change is a new lexicon. The earlier version of TRmorph used a modified version of lexicon from the Zemberek project (A. A. Akın and M. D. Akın, 2007). The current lexicon of TRmorph is generated semi-automatically from a large web-based corpus of about 800M tokens collected by crawling the Web with a procedure similar to one described in Baroni et al. (2009). The lexicon is built by using the unknown word guesser and disambiguator described below, as well as checking the automatically extracted root forms against online dictionaries, and by hand-editing the results.

The initial motivation of the lexicon change was the concerns raised by some users from the industry about incompatibilities of different open-source licenses used by Zemberek and TRmorph. However, extracting a root lexicon from the Web and keeping it up to date is an attractive goal by itself. The refinement of the procedure outlined above and the lexicon itself is still in progress.

3.3. Rewrite with a different FST formalism

Under the hood, the morphological description has been completely rewritten. TRmorph was initially written in SFST (Schmid, 2005). The new version has been rewritten using lexc and xfst languages from Xerox (Beesley and Karttunen, 2003), using the free implementation Foma, (Hulden, 2009). TRmorph can also be used with HFST (Lindén, Silfverberg, et al., 2009).

The main reason behind rewriting the finite-state specification in another language was maintainability. Although SFST provides a familiar and complete FST description language, during development of TRmorph there were quite a few cases where the same automata needed to be repeatedly defined. Furthermore, the Xerox languages lexc and xfst have been around for a longer time, and are better known in the computational linguistics community. This also helps users who want to change the source code to fit their needs. These being said, similar problems arose during the rewrite of the system with the Xerox languages. To prevent some of the repetition and redundancy, TRmorph source code now includes C preprocessor directives which are preprocessed before compiling the code with lexc and xfst. The use of a preprocessor also enables a straightforward method for conditional inclusion or exclusion of code fragments, providing a natural way to enable or disable some behavior in compile time.

3.4. Options

TRmorph came into existence because of the need for customization. Although there had been other analyzers, most notably Oflazer (1994), reported in the literature, their availability did not allow customization necessary for some purposes. The initial motivation for developing TRmorph was to be able to analyze child and child-directed speech transcriptions (Çöltekin and Bozşahin, 2007). However, similar modifications are often necessary in other practical applications, for example one may want to allow analysis

[3] Also in preperation of http://tscorpus.com/.

[4] http://wiki.apertium.org/wiki/Apertium-aze.

[5] https://gitorious.org/hunmorph-foma/pages/Home.

of forms common in colloquial language found in social media, or on the Internet at large. The open source nature of TRmorph allows users to customize the analyzer to their needs by changing the source code of the analyzer. However, the FST formalisms used in describing the morphology are not necessarily easy to grasp even for experienced programmers. A significant amount of learning time is required before one can even do simple modifications. Mostly based on the requests from the users, new release of TRmorph allows some common options to be specified during the compile time. Example options include whether to accept common misspellings, analyze words written in all capital letters or to produce '+' separated analysis strings similar to Oflazer (1994). This allow users to specify these common preferences without needing to learn the finite state language(s) the analyzer was implemented with.

4. New finite-state tools

Since Turkish is a morphologically complex language, some common NLP tasks can only be done properly with the help of a morphological analyzer. Most of these tools share a common code base provided by the analyzer, and their usage is similar, since the ideal input for these tools is the analysis string rather than the surface form of a word. We also describe a number of tools that do not make use of the analyzer, but are included in the TRmorph distribution since they are relatively easy to implement within a finite-state framework.

4.1. Morphological segmentation

Morphological segmentation is the task of finding the surface morphemes of a word. The result is interesting for various research purposes, such as in evaluating unsupervised morphological analyzers (e.g., Çöltekin, 2010b; Kılıç and Bozşahin, 2012; Janicki, 2013).

For an agglutinating language like Turkish, one can find almost a one-to-one correspondence with the morphemes and their surface forms. However, analysis looses the link between the underlying morpheme and its surface form. Furthermore in presence of zero morphemes, morphemes without a surface realization, one cannot even determine the number of surface morphemes.

The main TRmorph FST includes root and morpheme boundary symbols used internally for morphophonological rules that are sensitive to these boundaries. Since the surface forms of written words do not contain these boundaries, they are deleted from the surface words in the regular analyzer. The segmentation tool, alongside a few others described below, shares a common strategy in its implementation. The crucial part in the segmentation tool is a slightly modified finite state transducer that keeps the intermediate boundary symbols on the surface string. In its simplest form, the segmentation FST takes an analysis string, and generates a surface form with boundary markers. If the aim is to produce the segmented form of a surface word, the automata for segmentation and analysis can be composed to convert the surface form to a segmented surface form directly. This strategy will produce ambiguous results in some cases. For example, the word *evleri* will be segmented as both *ev-ler-i* 'his/her houses', and *ev-leri* 'their house'. Disambiguating the analysis first, and then generating the segmented surface form would be desirable in most cases.

4.2. A grapheme-to-phoneme converter

Turkish orthography is rather transparent with respect to standard pronunciation. One can build a straightforward mapping from orthographic form to a phonetic representation with a number of alternations sensitive to immediate phonetic or orthographic context. However, there are certain aspects of pronunciation, particularly the placement of lexical stress in a word, that change depending on the morphological process. As a result, the morphological analyzer is also necessary for proper grapheme-to-phoneme (g2p) conversion. The functionality of the g2p tool is similar to the earlier work described by Oflazer and Inkelas (2006). However, as well as being open source, the g2p tool described here also differs significantly from the architecture explained in Oflazer and Inkelas (2006).

As with the morphological analyzer, the g2p description follows Göksel and Kerslake (2005). The default lexical stress in Turkish is on the final syllable. And when most suffixes are added to a stem, the stress moves to the final syllable of the word. Some roots, especially of place names and some borrowings, exhibit exceptional stress, and some morphemes contain non-stressable syllables. In a simplified form:

- If a stressable suffix is added to a root form with final stress, the stress moves to the final syllable (of the added suffix).
- If a suffix with a non-stressable syllable is added to a stem, stress falls before the non-stressable syllable. The suffixes after that (stressable or not) do not change the position of the stress.
- The location of stress does not change with suffixation of root forms with exceptional stress. In the presence of the negative marker (-ma/-me), however, the stress falls on the syllable before the negative marker irrespective or the stress placement on the root.

As in the segmentation tool, the main implementation is based on a modified transducer that is typically used in generation mode. To accommodate the stress calculations, the morphotactics description in TRmorph is modified to include internal symbols that mark non-stressable syllables in the suffixes. Similarly, the lexicon syntax is extended to allow stress marking on the root forms. In regular analyzer/generator, the stress related symbols are discarded at the beginning of the morphophonological processing. When compiling the g2p transducer, these markers, as well as the morpheme boundary markers are left on the orthographic surface form. The orthographic form with these markers are, in turn, syllabified, converted to a phonetic form in International Phonetic Alphabet (IPA) symbols and the location of the stress is determined based on the internal stress and boundary markers, and a surface stress marker is inserted before the stressed vowel (or optionally before the stressed syllable).

The description of the g2p system is fairly complete, and has been useful in some preliminary experiments in unsupervised learning of segmentation (Çöltekin and Nerbonne, 2014). However, the grapheme-to-phoneme converter described here is relatively new and experimental. Particularly, the exceptional stress marking in lexicon is still in progress.

4.3. Stemming lemmatization

Stemming is one of the common tasks in information retrieval and many other NLP tasks. Since possible suffix sequence inventory for Turkish is rather large (and theoretically infinite), the only way to achieve reasonable stemming is through morphological analysis. The present TRmorph distribution includes a stemmer that first analyzes the given word, and strips away all analysis symbols, leaving only the stem. One can optionally include the part of speech tag, and/or the derivational suffixes that immediately follow the root in the stemmer output.

4.4. Unknown word guesser

Updating the lexicon of a morphological analyzer is one of the constant maintenance tasks. However, it is simply impossible to add every possible root form to the lexicon. An unknown word guesser is useful for providing alternative analyses of unknown words. TRmorph now includes an unknown guesser which replaces the lexicon of the standard analyzer with a relatively permissive finite state automaton. The lexicon of the guesser allows arbitrary character strings (with some configurable restrictions) to be used as the root of an open-class word. If the guesser is combined with the analyzer using the priority union operation (Beesley and Karttunen, 2003), the guesser will only be used for the words that cannot be analyzed by the analyzer. This produces analyses that stem from the lexicon when possible, but falls back to the guesser when analyzer fails.

Another possible use of the unknown word guesser is to identify unknown roots. A variation of guesser in combination with a disambiguator was used during the construction of the current TRmorph lexicon.

4.5. Hyphenation and tag set conversion

Another simple but useful tool newly included in TRmorph is a FST for hyphenation. Hyphenation in Turkish can be done completely automatically, since the hyphenation is allowed at any syllable boundary, and the syllable structure can unambiguously be inferred form the orthographic form of a word.

Lastly, we introduce two experimental converters that convert between different morphological tagsets. First, a converter that converts from standard TRmorph tagset to a tagset compatible with the conventions used in the CHILDES database (MacWhinney and Snow, 1990). Second, a converter from the common tagset (based on Oflazer, 1994) used in corpora from earlier studies to TRmorph tagset. The second converter is used for converting a well known disambiguated data set to TRmorph format for training the disambiguator described in Section 5.

5. Morphological disambiguation

Morphological disambiguation is the task of selecting the most likely morphological analysis of a given word. It is similar to the part of speech tagging. However, the number of classes (size of the tagset) and the sparsity of the data due to large number of missing word forms complicate the problem.

Even in languages with relatively simpler morphology, a word may have multiple analyses. The most straightforward case of morphological ambiguity is when a particular stem belongs to multiple word classes, e.g., English word 'book' having a noun and verb sense. Similarly, ambiguous affixes may also contribute to the number of possible analyses of a word. For morphologically complex languages like Turkish, the number of ambiguous analyses is typically much higher. Depending on the semantic distinctions one wants to make in the tagset, some words may be analyzed in hundreds of different ways. As a result, morphological disambiguation is an important step in Turkish NLP, and has been a popular topic in Turkish NLP literature.

TRmorph produces highly ambiguous results compared to the numbers reported in the literature using Oflazer's analyzer. With default options, TRmorph produces on average about 12 analyses per word. On the other hand, for example, Hakkani-Tür et al. (2002) reports an average ambiguity rate of less than two. The differences are, at least in part, due to the choices made during the design of these analyzers. During the analysis TRmorph produces all possible analyses of a word, including very unlikely ones. For example, for any adjective (without any suffixes, or any other root ambiguity), TRmorph produces eight analyses. The reasons for this ambiguity include the fact that any adjective in Turkish can be used as a (pro)noun referring to an object with relevant property; any noun or adjective may be used a predicate with present copula and 'null' person-number agreement; the null agreement on predicates may agree with third person singular and plural subjects; and the person-number agreement of a predicate may be attached the following question particle, or another predicate (in case of coordination). The cases of ambiguity are explained in detail in the TRmorph manual.

Earlier disambiguators presented in the literature have been based on the analyses (tagset) of Oflazer (1994). Unfortunately, none of the earlier systems could be used with the output of TRmorph, either due to the lack of availability or due to the fact that the tagsets used in the analyzers differ substantially. The rest of this section reports a simple morphological disambiguator that works well with TRmorph output.

5.1. Related work

The earliest work on Turkish morphological disambiguation is a rule-based system introduced by Oflazer and Kuruöz (1994). Oflazer and Tür (1997) present a follow-up work on the rule-based disambiguator. The motivation is to relax the need for determining the rule application order using a majority voting method. The accuracy values reported in these models are rather high (about 98–99%). However, this success is likely to be due to overfitting to the small corpora used in these studies, since these earlier

models have not been in use despite substantially lower performance of later models.

The first statistical disambiguation work was reported by Hakkani-Tür et al. (2002). The method used in this study is essentially the same as a standard trigram HMM tagger POS tagger. However, to alleviate the data sparseness problem, they decompose the morphological analyses into IGs. The best model presented in Hakkani-Tür et al. (2002) achieves around 94% accuracy. Yüret and Türe (2006) introduce a method that learns morphological analysis of a given word from the features extracted only from surface form of the target word and its left and right neighbors. Yüret and Türe (2006) report an accuracy of 95.82%. In a more recent study, Sak et al. (2007) presents a disambiguation system which takes the 50-best analyses of a sentences using the model of Hakkani-Tür et al. (2002), and re-ranks these analyses using a variation of the perceptron algorithm. The resulting disambiguation system achieves 96.28% accuracy.

Except the first two rule-based disambiguation systems, all recent studies make use of the same data set. However, the scores may not be directly comparable due to different testing settings.

5.2. Data and preprocessing

The largest data set used in this study is the data set from Yüret and Türe (2006).[6] This corpus seems to have originated from study by Hakkani-Tür et al. (2002), and used by almost all subsequent studies. It contains newspaper text collected from online newspapers. The data consists of two parts. The first part contains approximately 1M semi-automatically disambiguated tokens. We will call this data set '1M', The second part consists of approximately 1K hand-annotated tokens. We will call this set as '1K' in this paper. As observed by others earlier, despite being hand-annotated, 1K also contains errors, which were not corrected in this study.

Since both 1M and 1K are analyzed using Oflazer's (1994) morphological analyzer, the data had to be converted to TRmorph format. A large part of the data can be converted using the conversion utility included in the TRmorph distribution. However, since there is no one-to-one mapping between the tagsets used by two analyzers, this is a best-effort conversion. Some of the tags of TRmorph never appear on the resulting data set, and some information is lost during the conversion. Although the conversion utility produces a single result for each analysis found in 1M and 1K, in about 5% of the cases, TRmorph does not generate the same surface string for the given analysis. Most of these cases are due to out-of-vocabulary words (mainly proper nouns), disagreements about POS tag of the root words in TRmorph lexicon and the corpus, different choices for including derived forms in the lexicon or analyzing them, and lastly, liberal use of word guesser to mark typos and non-word strings as words.

These two data sets allow rough comparisons of the results obtained in this study with the earlier studies in the literature. To test the success of the disambiguation on the target system, the native output of TRmorph, we also use two small data sets that are analyzed by TRmorph and disambiguated manually. One of these data sets is a set of 357 sentences (approximately 5K tokens excluding punctuation) randomly selected from a Web corpus of approximately 800M tokens. As an attempt to test effects of a more balanced corpus, we also use a small data set of 164 sentences (approximately 2K tokens excluding punctuation) randomly selected from the METU corpus (Say et al., 2002). Similar to the web data, these sentences were analyzed by TRmorph and disambiguated manually. Table 1 presents some statistics for each data set used in this study.

Data set	sent.	tokens	types
1M	50519	834924	111127
1K	41	861	524
web5K	302	5025	3079
metu1K	76	1142	773

Table 1: Number of sentences, tokens and types in the data sets used in this study.

5.3. Models of disambiguation without context

Despite the fact that there has been a relatively large number of studies on morphological disambiguation of Turkish, it is surprising that none of the earlier studies considered a simple disambiguator that does not make use of the word context, but relies only on the features available in the word and its analyses.

Indisputably, the neighboring words are important for morphological disambiguation. Contextual differences are also the main motivation behind the disambiguation effort. Nevertheless, all words will have a 'most likely' interpretation, and finding the most common analysis of a word (or preferably, scoring all the analyses of a word) is useful for a number of reasons.

First, the use of a zero-context disambiguation model serves as a baseline in evaluation of more complex models. Second, such a disambiguation system is useful when context is not available. And yet another possibility to assign scores or ranks using such a disambiguator, but let the ambiguity resolution be done by a higher level process, e.g., a parser, which would be in a better position to determine the correct contextual dependencies. This approach also has the advantages of not doing double-work, and not biasing the decision of the higher level process with the way the disambiguation system relates the words in context.

Here, we present three methods to assign a probability to a given analysis string. In essence, what we are looking for is $\arg\max_{\mathsf{T}} \mathsf{P}(\mathsf{T}|W)$ for all morphological analyses T (root + morphological tags) offered by the analyzer. If we rewrite the term that we want to maximize, $\mathsf{P}(\mathsf{T}|W) = \frac{\mathsf{P}(W|\mathsf{T})\mathsf{P}(\mathsf{T})}{\mathsf{P}(W)}$, we observe that $\mathsf{P}(W)$ does not change for the analyses we compare, and aside from a few exceptions, $\mathsf{P}(W|\mathsf{T})$ is 1 for Turkish. That is, given an analysis, there is only one surface form that corresponds to it. With these assumptions, the quantity to maximize is $\mathsf{P}(\mathsf{T})$.

[6] Obtained from http://www.denizyuret.com/2006/11/turkish-resources.html on 2013-09-26.

5.3.1. Model 1: a rote-learning disambiguator

First model we define here is a 'rote-learning' model that simply counts the analyses observed in the training data. During testing, it picks the analysis that was observed in the training with the highest frequency.

The model takes the analysis string T as a whole, and estimates the probability of an analysis string using its relative frequency in the training corpus with add-one smoothing. During the test time we simply pick the analysis with the highest $P(T)$. The ties are broken randomly.

5.3.2. Model 2: making use of analysis without root

Model 1 described in Section 5.3.1 counts complete analysis strings in the training data. This model will have difficulties estimating probabilities of a large number of analyses. Even for very frequent roots, it is unlikely (and theoretically impossible) that all possible suffix sequences will be observed in the training data. However, if a particular sequence of suffixes is frequent, regardless of the root it follows, we expect the same sequence to be attached to unobserved roots with high probability as well.

The model we describe in this section decomposes an analysis string T into a root r and the part of the analysis string excluding the root a. For example for analysis string ev⟨N⟩⟨loc⟩⟨ki⟩⟨Adj⟩, r is ev and a is ⟨N⟩⟨loc⟩⟨ki⟩⟨Adj⟩. We are interested in the joint probability of the r and a, which can be factored as, $P(r, a) = P(r|a)P(a)$. As before, we estimate the probabilities from relative frequencies in the training corpus with add-one smoothing. Ties are again broken randomly, but ties are less of an issue for this model in comparison to Model 1.

5.3.3. Model 3: making use of IGs

Model 2 splits an analysis string into two. However, we can exploit the structure of the analyses for better estimation of the probabilities for rare or unknown words.

The model defined here, Model 3, makes use of the sequence of IGs, and decomposes an IG into a root r or a derivation d and a sequences of inflection(s) i including the category marker. An analysis string is represented as at least one (r, i) tuple followed by a possibly unbounded number sequence of (d, i) tuples. For example the word *evdeki* 'the one at the house' is represented as two IGs with the following components.

$$\underbrace{\text{ev}}_{r}\ \underbrace{\langle N\rangle\langle loc\rangle}_{i_0}\ \underbrace{\langle ki\rangle}_{d_1}\ \underbrace{\langle Adj\rangle}_{i_1}\ \ldots$$

Having this decomposition at hand, we make two independence assumptions. Namely, (1) an i is independent of other parts of the analysis given present r or d and (2) a d is independent of other parts of the analysis given the i preceding it.

We estimate the probability of an analysis string with n derivations by

$$P(T) = P(i_0)P(r|i_0)\prod_{k=1}^{n} P(d_k|i_{k-1})P(i_k|d_k)$$

Note that we invert the conditional probability $P(i_0|r)$ as before.

Test set	Model 1	Model 2	Model 3
10fold	0.90±0.001	0.94±0.001	0.93±0.001
1K	0.92±0.010	0.96±0.007	0.93±0.009
web	0.78±0.006	0.87±0.005	0.86±0.005
metu	0.76±0.013	0.84±0.011	0.84±0.011

Table 2: Accuracy of the disambiguation models trained on the 1M corpus. '10fold' indicates 10-fold cross validation results, and otherwise, the corpus used for testing. The intervals are approximate standard errors assuming accuracy values follow the binary distribution.

5.4. Experiments and Results

In all experiments reported below, we train all three models on the indicated training data. During testing, we analyze the surface words using TRmorph, pick the best analysis according to the model, and compare it with the gold-standard analysis. We present 10-fold cross validation results on the same corpus, and tests that are done on corpora with different properties.

The first set of result we present in Table 2 are accuracy of models trained on the 1M corpus. The most interesting finding at first sight is that Model 2 performs surprisingly well in 10-fold cross validation test and while testing on the 1K corpus. It rivals earlier more complex context-aware models tested on the same corpora, scoring about 2% below the best results obtained so far.

With respect to the different test sets used, the results are as expected. All models perform better on familiar corpora, and perform worse as the territory becomes more unknown. The web5K and metu1K corpora are different from the training data since they do not contain newspaper text only, and they are annotated directly using the TRmroph tagset. The differences between models are also as expected. The baseline model, Model 1, performs the worst. So far, Model 2 seems to perform the best in all test scenarios, since it does not make the simplifying independence assumptions of Model 3.

The accuracy scores presented in Table 2 credit a model only if the gold-standard analysis receives the highest score. Figure 1 shows the ranks assigned to gold-standard analyses on the 1K corpus by each model trained on the 1M corpus. The figure demonstrates that misses by models 2 and 3 are mostly clustered in rank two or three. Both models include the gold-standard analyses within the first two ranks in 99% of the cases.

It is clear from Table 2 that the performance of the models on web5K and metu1K is far behind the results obtained on the newspaper corpus used for training. The next set of experiments investigate the question of whether we can achieve better disambiguation using a small but more similar training set.

The accuracy of the models trained on web5K corpus is presented in Table 3. The results clearly indicate that 5K tokens are not enough for achieving the level of accuracy obtained on larger but less familiar corpus. We also note that Model 3 performs well above others in this small training set. The complexity introduced by model 3 seems to

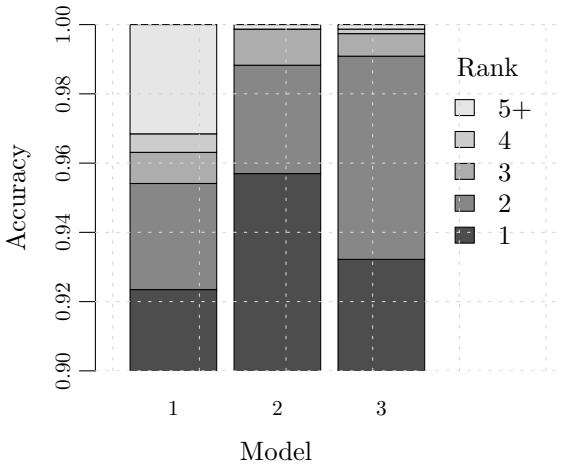

Figure 1: Rank assigned to the gold-standard analyses on the 1K corpus by each model trained on the 1M corpus. Note that the y-axis is cropped at 90% to show the differences between the models clearly.

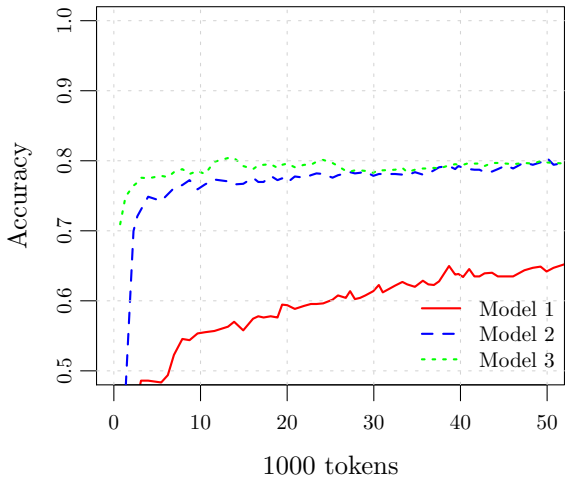

Figure 2: The learning curve of the models trained incrementally on the 1M corpus, tested on metu1K. The x-axis is cropped below 0.5, and the y-axis do not show all 800K tokens to emphasize the differences with small training sets.

Test set	Model 1	Model 2	Model 3
10fold	0.52±0.023	0.50±0.023	0.79±0.018
metu	0.52±0.015	0.53±0.015	0.77±0.012
1K	0.47±0.018	0.49±0.018	0.80±0.014

Table 3: Accuracy of the zero-context disambiguation models trained on web5K corpus. '10fold' indicates 10-fold cross validation results, and otherwise, the corpus used for testing. The intervals are approximate standard errors assuming accuracy values follow the binary distribution.

pay off if one has to work with a small training set. To show the differences in learning, Figure 2 presents the learning curves for each model when trained on the 1M corpus incrementally, and tested on metu1K. As expected, Model 1 is the slowest learner. Unlike when trained on web5K corpus, Model 2 performs better, but it stays behind Model 3 until about 30K tokens. The graph in Figure 2 presents results only until 50K tokens to emphasize the changes at the beginning. Although the models 2 and 3 reach to a certain level quickly at about 10K tokens, the small but steady increase in accuracy continues until the end the 1M corpus.

6. Summary

In summary, this paper provides an update on major changes on TRmorph over the last four years; introduces a set of (finite-state) tools that are useful for speech and language processing; and describes a new morphological disambiguation model suitable to be used with TRmorph. It should also be stressed that most of the new and additional tools are motivated by the users other than the author. The changes introduced and additional tools implemented over the last four years make TRmorph a more usable system for Turkish NLP. TRmorph and the associated tools are actively being developed. Besides constant improvements, the tool set is constantly being expanded with new tools and functionality.

Even though the disambiguator presented here does not make use of any context, the performance of it is only about 2% behind the best performing disambiguators in the literature. The first impression out of a preliminary error analysis suggests that the context is not utilized enough by the earlier models. Furthermore, the data set used by all models in the literature (including the present one) is highly specific, and contains many errors. A better disambiguation system is one of the clear directions for future studies.

7. References

Akın, Ahmet Afşin and Akın, Mehmet Dündar (2007). "Zemberek, an open source NLP framework for Turkic Languages."

Baroni, Marco, Bernardini, Silvia, Ferraresi, Adriano, and Zanchetta, Eros (2009). "The WaCky wide web: a collection of very large linguistically processed web-crawled corpora." In: *Language Resources and Evaluation* 43.3, pp. 209–226.

Beesley, Kenneth R. and Karttunen, Lauri (2003). "Finite-state morphology: Xerox tools and techniques." In: *CSLI, Stanford*.

Çöltekin, Çağrı (2010a). "A Freely Available Morphological Analyzer for Turkish." In: *Proceedings of the 7th International Conference on Language Resources and Evaluation (LREC 2010)*. Valetta, Malta, pp. 820–827.

Çöltekin, Çağrı (2010b). "Improving Successor Variety for Morphological Segmentation." In: *Proceedings of the 20th Meeting of Computational Linguistics in the Netherlands*. Utrecht, The Netherlands, pp. 13–28.

Çöltekin, Çağrı and Bozşahin, Cem (2007). "Syllables, Morphemes and Bayesian Computational Models of Acquiring a Word Grammar." In: *Proceedings of 29th Annual Meeting of Cognitive Science Society*. Nashville, pp. 887–892.

Çöltekin, Çağrı and Nerbonne, John (2014). "An explicit statistical model of learning lexical segmentation using multiple cues." In: *Proceedings of EACL 2014 Workshop*

on Cognitive Aspects of Computational Language Learning.

Eryiğit, Gülşen, Nivre, Joakim, and Oflazer, Kemal (2008). "Dependency Parsing of Turkish." In: *Computational Linguistics* 34.3, pp. 357–389.

Forcada, Mikel L., Tyers, Francis M., and Ramírez-Sánchez, Gema (2009). "The Apertium machine translation platform: five years on." In: *Proceedings of the First International Workshop on Free/Open-Source Rule-Based Machine Translation*, pp. 3–10.

Göksel, Aslı and Kerslake, Celia (2005). *Turkish: A Comprehensive Grammar*. London: Routledge.

Güneş, Güliz and Çöltekin, Çağrı (2014). "Mapping to prosody: not all parentheticals are alike." In: *Parenthetical Verbs*. Ed. by S. Schneider, J. Glikman, and M. Avanzi. Berlin: De Gruyter, (to appear).

Hadımlı, Kerem and Yöndem, MeltemTurhan (2012). "Two Alternate Methods for Information Retrieval from Turkish Radiology Reports." In: *Computer and Information Sciences II*. Ed. by Erol Gelenbe, Ricardo Lent, and Georgia Sakellari. Springer London, pp. 527–532.

Hakkani-Tür, Dilek Z., Oflazer, Kemal, and Tür, Gökhan (2002). "Statistical Morphological Disambiguation for Agglutinative Languages." In: *Computers and the Humanities* 36.4, pp. 381–410.

Hankamer, Jorge (1986). "Finite state morphology and left to right phonology." In: *Proceedings of the West Coast Conference on Formal Linguistics*. Vol. 5. Stanford Linguistic Association.

Hulden, Mans (2009). "Foma: a finite-state compiler and library." In: *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics: Demonstrations Session*. Association for Computational Linguistics, pp. 29–32.

Janicki, Maciej (2013). "Unsupervised Learning of A-Morphous Inflection with Graph Clustering." In: *Proceedings of the Student Research Workshop associated with RANLP*, pp. 93–99.

Kairakbay, Bakyt M. (2013). "Finite State Approach to the Kazakh Nominal Paradigm." In: *Finite State Methods and Natural Language Processing*, pp. 108–112.

Kılıç, Özkan and Bozşahin, Cem (2012). "Semi-supervised morpheme segmentation without morphological analysis." In: *First Workshop on Language Resources and Technologies for Turkic Languages, LREC*, pp. 52–56.

Lindén, Krister, Axelson, Erik, Drobac, Senka, Hardwick, Sam, Silfverberg, Miikka, and Pirinen, Tommi A. (2013). "Using HFST for creating computational linguistic applications." In: *Computational Linguistics*. Springer, pp. 3–25.

Lindén, Krister, Silfverberg, Miikka, and Pirinen, Tommi (2009). "HFST Tools for Morphology–An Efficient Open-Source Package for Construction of Morphological Analyzers." In: *State of the Art in Computational Morphology*. Ed. by Cerstin Mahlow and Michael Piotrowski. Communications in Computer and Information Science. Springer, pp. 28–47.

MacWhinney, Brian and Snow, Catherine (1990). "The child language data exchange system: An update." In: *Journal of Child Language* 17.2, pp. 457–472.

Oflazer, Kemal (1994). "Two-level description of Turkish morphology." In: *Literary and Linguistic Computing* 9 (2).

Oflazer, Kemal and Inkelas, Sharon (2006). "The architecture and the implementation of a finite state pronunciation lexicon for Turkish." In: *Computer Speech and Language* 20.1, pp. 80–106.

Oflazer, Kemal and Kuruöz, İlker (1994). "Tagging and morphological disambiguation of Turkish text." In: *Proceedings of the fourth conference on Applied natural language processing*. Association for Computational Linguistics, pp. 144–149.

Oflazer, Kemal and Tür, Gökhan (1997). "Morphological Disambiguation by Voting Constraints." In: *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics*, pp. 222–229.

Özdikiş, Özer, Şenkul, Pınar, and Oğuztüzün, Halit (2012). "Semantic expansion of tweet contents for enhanced event detection in Twitter." In: *IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*. IEEE, pp. 20–24.

Ruhi, Şükriye, Eryılmaz, Kerem, and Acar, M. Güneş C. (2012). "A platform for creating multimodal and multilingual spoken corpora for Turkic languages: Insights from the Spoken Turkish Corpus." In: *First Workshop on Language Resources and Technologies for Turkic Languages, LREC*, pp. 57–63.

Şahin, Muhammet, Sulubacak, Umut, and Eryiğit, Gülşen (2013). "Redefinition Of Turkish Morphology Using Flag Diacritics." In: *Proceedings of The Tenth Symposium on Natural Language Processing (SNLP-2013)*.

Sak, Haşim, Güngör, Tunga, and Saraçlar, Murat (2007). "Morphological Disambiguation of Turkish Text with Perceptron Algorithm." In: *CICLing 2007*. Vol. LNCS 4394, pp. 107–118.

Say, Bilge, Zeyrek, Deniz, Oflazer, Kemal, and Özge, Umut (2002). "Development of a Corpus and a TreeBank for Present-day Written Turkish." In: *Proceedings of the Eleventh International Conference of Turkish Linguistics*. Eastern Mediterranean University, Cyprus.

Schmid, Helmut (2005). "A programming language for finite state transducers." In: *Proceedings of the 5th International Workshop on Finite State Methods in Natural Language Processing (FSMNLP 2005)*. Helsinki, pp. 308–309.

Washington, Jonathan, Ipasov, Mirlan, and Tyers, Francis M. (2012). "A finite-state morphological transducer for Kyrgyz." In: *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12)*. Ed. by N. Calzolari, Choukri K., Declerck T., M. U. Doğan, Maegaard B., J. Mariani, Odijk J., and S. Piperidis. Istanbul, Turkey.

Yüret, Deniz and Türe, Ferhan (2006). "Learning morphological disambiguation rules for Turkish." In: *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*. HLT-NAACL '06. New York, pp. 328–334.

10 Natural Language Processing Tools â€“ SaaS & Open-Source. We spend a lot of time having conversations and engaging with others via chat, email, websites, social mediaâ€¦ But we donâ€™t always stop to think about the massive amounts of text data we generate every second.Â Natural Language Processing (NLP) is a discipline within artificial intelligence that leverages linguistics and computer science to make human language intelligible to machines. By allowing computers to automatically analyze massive sets of data, NLP can help you find meaningful information in just seconds. Letâ€™s say you work at a SaaS and want to perform data analysis on customer support tickets to identify the most common issues raised by your clients. We built an open-source software platform intended to serve as a common infrastructure that can be of use in the development of new applica-tions involving the processing of Turkish.Â Although there are several high precision morphological analysis tools for Turkish language (Eryigit & Adali, 2004;Oflazer, GÃ¶Ã§men, & BozÅŸahin, 1994;Say et al., 2004;Ã‡Ã¶ltekin, 2010), Preparation of a document-based corpus and construction of the lexicon are explained in the following sections. ...Â The Arabic data set, consisting of transcripts of telephone conversations 9 , is considerably smaller than the data sets of the other languages. ...Â An intelligent natural language interface based on Turkish ,Language is designed for creating Java class skeleton, listing the class and its members. A curated list of awesome Turkish language processing libraries, models, resources and datasets. The main focus is on open source tools, downloadable data and research papers with code. t.me/turkishnlp. 6 stars.Â Awesome Turkish NLP. A curated list of awesome Turkish language processing libraries, models, resources and datasets. The main focus is on open source tools, downloadable data and research papers with code. Contents. Libraries. Natural language processing (NLP) is a field of computer science, artificial intelligence, and computational linguistics concerned with the interactions between computers and human (natural) languages. It includes word and sentence tokenization, text classification and sentiment analysis, spelling correction, information extraction, parsing, meaning extraction, and question answering. In our formative years, we master the basics of spoken and written language.Â To provide an insight into the quality of software that is available, we have compiled a list of 14 excellent open source NLP tools. Hopefully, there will be something of interest here for anyone who wants to use these tools to solve practical problems. Hereâ€™s our fine-tuned recommendations. IceNLP is an open source Natural Language Processing (NLP) toolkit for analyzing and processing Icelandic text. The toolkit is implemented in Java. 1 Review.Â The project supports the Welsh Language Technology domain with a set of NLP tools that drive innovation and advance the development of sophisticated textual analysis solutions. The WNLT project delivers four core NLP modules; a) Word Segmentation for separating text into words b) Sentence Boundary Disambiguation for finding sentence boundaries c) Part of Speech Tagger for determining the part of speech of each word d) Morphological Analyser for identifying the root form (lemma) of words