Improving Turkish Spelling Correction with Wikipedia Edit History Data

Theresa Guinard

A thesis
submitted in partial fulfillment of the
requirements for the degree of

Master of Science

University of Washington

2021

Committee:

Fei Xia

Meliha Yetişgen

Program Authorized to Offer Degree:
Department of Linguistics

University of Washington

Abstract

Improving Turkish Spelling Correction with Wikipedia Edit History Data

Theresa Guinard

Chair of the Supervisory Committee:
Professor Fei Xia
Department of Linguistics

Spelling correction is a well-established NLP application, but the quality for English spelling correction tends to be significantly higher than for other languages. One significant issue for minority languages in NLP is the availability of specialized corpora for various tasks. In this thesis, I develop and make available[1] a corpus of about 780,000 Turkish spelling errors and their corrections. I present a fairly language-independent system[2] for identifying small edits from Wikipedia's edit history and a decision forest classifier with Turkish-specific features for distinguishing spelling corrections from other types of small edits. When analyzing the corpus, I find the major categories of error types to be changes in diacritics, changes in capitalization, changes in spacing, and character insertions/deletions/substitutions/swaps. I present a baseline cascaded system, where each error type category is handled by separate modules. When trained with the Wikipedia data, this system handles cross-domain spelling errors more effectively than existing systems for Turkish normalization and spelling correction. Additionally, I investigate the possibility of using a machine translation model in a spelling correction system, and I find that an SMT model trained on the error model for character insertions/deletions/substitutions/swaps is a viable option for handling that category of errors.

---

[1] `https://github.com/tguinard/TurkishWikiSpellingMistakes`

[2] `https://github.com/tguinard/WikiSmallEdits`

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION

1.1 Background and Motivation

Spelling correction is one of the oldest and most widely-used NLP applications. Not only is spelling correction useful for user-facing suggestions in word processing and auto-complete programs, it is also a useful normalization step for many text-based NLP applications (for example, users of search engines or translation programs are often aided by spelling corrections in the form of "*did you mean...*?" query suggestions).

As with almost all NLP applications, techniques are generally developed with English as a priority, resulting in the technology for other languages lagging behind. Spelling correction in Turkish clearly suffers from this phenomenon. First, largely due to phonological and orthographic differences between Turkish and English, the nature of typical spelling mistakes is quite different. Additionally, some fundamental sub-tasks, such as identifying whether or not a word is in the language, require significantly modified approaches (for the case of word identification, this is due to Turkish's rich morphology).

An important problem when dealing with minority languages is the lack of corpora. In spelling correction, many advanced approaches require parallel corpora: one version with spelling errors, and the other with these errors corrected. For most minority languages including Turkish, there isn't a widely-accepted high-quality corpus of this kind, so many studies use workarounds to solve this problem, most often using corpora with synthetic errors or small, manually annotated corpora.

1.2 Approach

A major focus of this study is the curation of a large data set of spelling error and correction pairs. A common struggle when working with less commonly used languages is that resources for training data are more scarce, and manually labelled data sets may be too small to capture general phenomena. Therefore, automatically or semi-automatically labelling training data can often be a more practical and effective solution. Wikimedia offers the entire edit history of Wikipedia in all of its languages, available freely for download[1]. A natural idea is to build a corpus of spelling correction data from instances where typos are corrected, but this comes with challenges both in extracting the small edits and then determining whether the small edit is a spelling error. I release software[2] that efficiently extracts small edits, which can be applied to any language that is available on Wikipedia and whose words are separated by spaces. I then present a Turkish-specific method of extracting the spelling errors from the candidate edits, and I make the resulting dataset available for future researchers.[3]

In the second half of this study, I use this dataset to analyze what kinds of spelling errors are common in Turkish, and I investigate how to use the data and these insights to build baseline spelling correction systems. When taking into account the frequency of different types of errors, I find that the traditional approach to treating Turkish spelling correction as a noisy-channel problem (Oflazer, 1996; Torunoğlu-Selamet et al., 2016; Büyük, 2020) can handle a subset of the data well, but is fundamentally limited in the types of errors it can correct. As a general approach, it is more appropriate to treat spelling correction as a type of normalization problem, often studied in the context of social media texts (Torunoğlu-Selamet and Eryiğit, 2014; Eryiğit and Torunoğlu-Selamet, 2017; Çolakoğlu et al., 2019). I investigate using two different general paradigms that have been previously proposed for normalization problems: a cascaded system with a noisy-channel model, and an SMT-based

---

[1] `https://en.wikipedia.org/wiki/Wikipedia:Database_download`

[2] `https://github.com/tguinard/WikiSmallEdits`

[3] `https://github.com/tguinard/TurkishWikiSpellingMistakes`

system.

1.3 Organization

The rest of this thesis is organized as follows. First I describe the challenge of extracting the spelling error correction corpus from Wikipedia. This involves designing a scheme to efficiently extract small edits from the history dump (Chapter 2), and then a classification system to determine whether a small edit is a spelling correction (Chapter 3). I then present an analysis of this corpus, investigating the nature of Turkish spelling mistakes, investigating assumptions by previous work, and giving insight into important design considerations for a spelling correction system (Chapter 4). I briefly describe test datasets I use to evaluate spelling correction systems: both my own and those developed by previous studies (Chapter 5). I evaluate how previous proposed methods of Turkish spelling correction can benefit from this corpus: both cascaded approach (Chapter 6) and a machine translation-based approach (Chapter 7). Finally, I present a conclusion (Chapter 8).

Chapter 2
SMALL EDIT EXTRACTION FROM WIKIPEDIA

This chapter describes the first component of my system for generating the spelling correction data set: extracting small edits from Wikipedia's edit history. The purpose of this step is to convert raw data into candidates with a structured format that can be easily managed by downstream components.

This software, which I make publicly available[1], makes minimal language-specific assumptions: the only criteria used to detect small edits is the number of space-separated tokens in an individual edit and its surrounding context. Given this design and Wikipedia's extensive coverage on many languages, this software can be used for creating rich data sets for quite a large number of languages.

2.1 Problem Description

People may edit Wikipedia for a variety of reasons, ranging from adding and deleting large chunks of text, to small changes to a word, phrase, or markup. Daxenberger and Gurevych (2013) studied edit types in detail, building a categorizer for edit types including information addition, information modification, spelling correction, reverting a previous change, vandalism, and paraphrasing.

Miłkowski (2007) suggested that the majority of Wikipedia edits are small changes. A reasonable hypothesis (both in terms of Miłkowski's suggestion and convenience of data processing) is that a spelling correction edit involves a small number of consecutive words (modifications in spacing may be present: e.g. "in to" vs "into"). Thus, I have designed the first step of the process as extracting edits from the history, where a small number of

[1] `https://github.com/tguinard/WikiSmallEdits`

consecutive words are changed. I define a small edit as a string pair (a, b) aligned from some (original, revision) article version pair, where the contexts of a and b are the same (at least one surrounding word on either side is identical) and both a and b contain no more than $n = 3$ words.

2.2 Previous Work

2.2.1 Wikipedia Edit History for NLP Applications

Several studies have used Wikipedia edit history to aid various error correction and paraphrasing tasks: preposition error correction (Cahill et al., 2013), general grammatical error correction (Grundkiewicz and Junczys-Dowmunt, 2014; Boyd, 2018), spelling correction (Zesch, 2012; Nelken and Yamangil, 2008; Max and Wisniewski, 2010), paraphrasing (Max and Wisniewski, 2010; Nelken and Yamangil, 2008), lexical simplification (Yatskar et al., 2010), and text summarization (Nelken and Yamangil, 2008). Additionally, large-scale corpora covering various error types, extracted from Wikipedia edit history, have been released for English (Grundkiewicz and Junczys-Dowmunt, 2014), French (Max and Wisniewski, 2010), and Polish (Grundkiewicz, 2013).

2.2.2 Software for Processing Wikipedia Edit History

There are also some existing open source software packages that extract corpora from Wikipedia's edit history. Wikipedia Revision Toolkit (Ferschke et al., 2011) is a system that converts the revision history xml dump into a more efficient database storage format. However, it does not have direct support for extracting small edits, and nontrivial application code must be implemented on top of their system. WikiEdits[2], which was used to create the WikEd corpus (Grundkiewicz and Junczys-Dowmunt, 2014), does the same task of extracting small edits, but unfortunately there are performance inefficiencies, which are accounted for in this thesis.

[2] `https://github.com/snukky/wikiedits`

2.2.3 Common Extraction Issues

The studies that apply Wikipedia's edit history to NLP tasks have several subtasks in common:

- How to convert the list of full article versions to a list of edits
- How to convert Wikitext (Wikipedia's markup) into plaintext
- How to filter out junk edits: according to a crowdsourcing identification of vandalism by Potthast (2010), about 7% of Wikipedia's edits are vandalism

Aligning Two Versions of the Same Article

A Wikipedia history dump file stores the edit history of articles as a list of versions. Each version's full text is provided (as opposed to a list of edits or diffs). To find which portions of the article were edited, the most common solution is to model the problem as a longest common subsequence (LCS) problem, which is the same strategy as implemented by Unix's diff command.

Many of the previous studies introduced in Section 2.2.1 extracted minor edits (i.e. only a few words changed). For example, Cahill et al. (2013) extract sentences with only one changed word. Zesch (2012), Max and Wisniewski (2010), and Grundkiewicz and Junczys-Dowmunt (2014) extract sentences with a percent change under some threshold.

Converting Wikitext into Plaintext

Studies that deal with pure language-based edits often convert the Wikitext (markup) into plaintext as a preprocessing step. There are several libraries available to parse Wikitext. Cahill et al. (2013) used Java Wikipedia Library (Ferschke et al., 2011). Zesch (2012) used JWPL Wikipedia API (Zesch et al., 2008). Boyd (2018) used WikiExtractor[3].

[3] `https://github.com/attardi/wikiextractor`

Vandalism Cleanup

A significant proportion (about 7%) of Wikipedia edits are vandalism (Potthast, 2010), so several studies implement some kind of filtering mechanism to remove these junk edits from their datasets. Zesch (2012) excludes an edit if it matches a list of vulgar stopwords. Cahill et al. (2013) ignores "circular chains" of edits: if the same word in a sentence is changed multiple times, and in two versions, the word has the same value x, ignore any edits between these two versions. Grundkiewicz (2013) uses a similar list of stopwords as Zesch (2012), ignores circular edits similar to Cahill et al. (2013), and also excludes edits that involve modifying punctuation to repeat (e.g. "?" $\rightarrow$ "??") and edits that involve abnormal modifications in case ("Wikipedia" $\rightarrow$ "WiKipEdia"). Grundkiewicz and Junczys-Dowmunt (2014) detect vandalism reverts from comments associated with each edit.

2.3 Source Dataset

The Turkish Wikipedia edit history dump[4] was retrieved on July 1, 2020, which has a total unpacked size of about 200 GB.

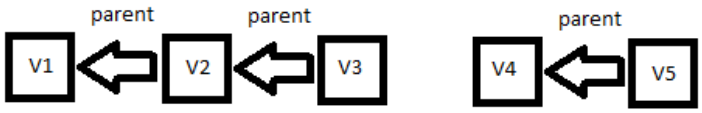
The edit history consists of the complete markup text of each article version, with the parent-child relationships between versions. In the vast majority of cases, the version graph is strictly linear (as in V1-V3 in Figure 2.1).

In a minority of cases (0.003% of edits), a new version may be considered a rewrite, rendering a graph similar to Figure 2.1, where the change from V3 to V4 is considered a rewrite.

Another minority case (0.007% of edits) is that there are two (or more) versions with the same parent. In this case, the new current version of the page is chosen to be only one of those versions, and the alternates are abandoned (see Figure 2.2). A case like this is plausible if two people save edits at (nearly) the same time. After manually checking a few examples like this, both child versions seem to be valid versions of the page, but with different edits,

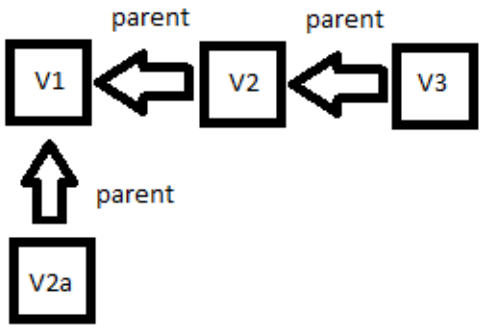
[4] `https://dumps.wikimedia.org/trwiki/latest/trwiki-latest-pages-meta-history.xml.bz2`

Figure 2.1: A version graph with one rewrite



so all links like this are kept and processed as normal.

Figure 2.2: A version graph with competing successors of V1. In this case V2a is abandoned



2.4 Approach

2.4.1 Data Partitioning

The data is kept in compressed (.bz2) format, where the total packed file size is about 8 GB and the total size of the unpacked file is about 200 GB. I process the .bz2 data directly; the unpacking is done on-the-fly, one article at a time. In order to make a fault tolerant system, I pre-split the data into individual files (by design, if the process does not exit cleanly, the whole file must be reanalyzed). In each file, only complete (<page>...</page>) xml elements are stored, and a new file is created if, upon starting to read from a new <page> starting tag, the number of lines written exceeds a threshold n.

2.4.2 Small Edit Extraction

Architecture

```
for page in EditHistory do
    for v1, v2 in page.AdjacentVersions do
        chunksBefore = ChunkOnMatchingParentheses(v1.markup);
        chunksAfter = ChunkOnMatchingParentheses(v2.markup);
        for beforeMarkup, afterMarkup in PrefixAlign(chunksBefore, chunksAfter) do
            if beforeMarkup ≠ afterMarkup then
                beforeToks = Tokenize(MarkupToPlaintext(beforeMarkup));
                afterToks = Tokenize(MarkupToPlaintext(afterMarkup));
                beforeToks, afterToks = RemoveCommonPrefixAndSuffix(beforeToks,
                 afterToks);
                for edit in Diff(beforeToks, afterToks) do
                    if edit.Before.TokenCount ≤ 3 and edit.After.TokenCount ≤ 3
                     then
                        SmallEdits.Append(edit);
                    end
                end
            end
        end
    end
end
```

Algorithm 1: Extracting small edits

Algorithm 1 is a summary of the approach taken for extracting small edits. Essentially, I compare two versions of a page at a time and find all of the edits using an LCS algorithm[5]. I implement various optimizations in order to reduce the cost of the relatively expensive operations of converting markup to plaintext and running the diff algorithm.

[5]I used python's difflib.SequenceMatcher: `https://docs.python.org/3/library/difflib.html`

Chunking Each Version

In order to convert markup into natural language text, I used the library mwparserfromhell[6]. Running this parser on the entire Wikitext of each version is costly, so when comparing two versions, this parser was only run on sections that changed between the versions.

In order to implement this optimization, for a Wikitext string T we must find a partition

$$T = T_1\ T_2 \ldots . T_n$$

Where

$$parse(T) = parse(T_1)\ parse(T_2) \ldots . parse(T_n)$$

The most straightforward solution is to find a partition that does not split on any "parenthetical" elements. For example, for a given starting HTML tag, the corresponding ending tag must occur in the same partition T_i. For Wikitext, relevant parenthetical elements are HTML tags, curly braces *{}*, and square braces *[]*. One additional (Wikitext specific) constraint is to always split on a blank line.

To align chunks of two versions, I use the diff algorithm on the prefixes (100 characters) of each chunk. If a prefix is unique in both versions and it is aligned by the diff algorithm, this chunk is marked as a valid start chunk. If a chunk is not a valid start chunk, it is appended to the previous chunk.

Word-Based Diff

Only if the two markup chunks differ do we perform the relatively expensive operation of looking for edits. This involves extracting the plaintext from the markups and performing a diff, where the unit being compared is a word. Using the unit of comparison as a word gives a significant performance boost, as the time complexity of a common solution to the longest substring algorithm implemented by python's difflib.SequenceMatcher is quadratic in

[6] `https://github.com/earwig/mwparserfromhell`

the worst case. For this problem, since we are interested in word-level changes anyways, this simple optimization beats Google's diff-match-patch[7] (used by Cahill et al. (2013)), which is heavily optimized but ultimately uses a character-level diff.

An additional optimization, which is also implemented by Google's diff-match-patch library, is to strip the common prefix and suffix between the two sequences. This effectively reduces the time complexity for the LCS algorithm, as the input is shorter.

2.4.3 Output

For each article, a json object with following information is outputted:

- Article Title
- Namespace
- ArticleId
- List of edits

Each edit in the list of edits has the following properties

- Original text
- Edited text
- Original text left context
- Original text right context
- Edited text left context
- Edited text right context

The context is the text surrounding the edit, so (left context) + (text) + (right context) would be a substring of the specific version.

Each context is collected by scanning the string left or right, until any one of these three conditions is met

- A paragraph break is encountered (defined as "\n\n")
- Two sentence breaks are encountered (defined as ". ")

[7] `https://github.com/google/diff-match-patch`

- 100 words are collected (defined as space separated tokens)

2.4.4 Redundant Edit Filtering

As a post-processing step, I filter circular and non-final edits: if a word with the same context in a specific page is edited multiple times, only keep the last edit. The last edit should also be discarded if it is part of a circular edit: the same word was seen in the same context at an earlier version of the page.

The purpose of this step is to filter out edits that are likely wrong: if someone edited the same word later, we shouldn't assume that the result of the earlier edited version is a valid correction. Often editors mistakenly "correct" a mistake, but inadvertently revert a correct form or introduce some other kind of mistake. This filtering catches instances where the bad edit was eventually detected and fixed by someone else.

Also, as mentioned in Section 2.2.3, vandalism is a common occurrence on Wikipedia, but fortunately, many vandalism instances are reverted by the community. This step removes several instances of such vandalism (often the insertion of vulgar or very informal words).

Additionally, there are many instances where information is regularly updated, for example numerical figures such as population count. Several instances of these edits are filtered out.

2.5 Results

2.5.1 Data Size for Turkish

Before performing redundant edit filtering (Section 2.4.4), I extracted 20,081,283 small edits. After filtering, this number was sizably reduced to 5,380,708. There were 1,826,378 pages represented in the history dump, including 619,614 article pages[8]. However, most of the small edits extracted (94%) belong to the article pages. A mean of 8 small edits were extracted

[8]pages with namespace 0; in the history dump, special pages such as "talk" and user profiles are also included. For a list of all types of pages see `https://en.wikipedia.org/wiki/Wikipedia:Namespace`

Package	Method name	CPU time (%)
mwparserfromhell	parse_anything	30.7%
(my code)	chunk	25.8%
bz2	readline	14.1%
mwparserfromhell	strip_code	9.2%
difflib	find_longest_match	4.4%

Table 2.1: 84.2% of the inclusive CPU time distribution (time attributed to the listed functions is non-overlapping)

from each article page after filtering, though this figure varied widely per page due to some pages being much more popular than others. At most 7,129 edits were extracted from a single article.

2.5.2 Performance

While developing this software, I used the CPU profiler snakeviz[9] to locate inefficiencies in terms of CPU processing time. This led to the development of optimizations as described in Section 2.4, especially in managing the input fed to difflib and minimizing the input fed to mwparserfromhell. In the final solution, we can attribute 84.2% of the processing power to five broad functions (Table 2.1).

The runtime attributed to bz2 is not easily reducible, as we are reading each line from the input .bz2 file exactly once. The amount of input fed to difflib and mwparserfromhell libraries has been optimized as described in Section 2.4. This means that at least 58.4% of the CPU time has been fully optimized, given the presented assumptions about how the text must be parsed and assuming that the libraries used are efficient.

2.6 Conclusion

This chapter presented a rather language-independent way of extracting small edits from Wikipedia's edit history, that is suitable even for researchers with limited computing re-

[9] `https://jiffyclub.github.io/snakeviz/`

sources: the edit history is never fully uncompressed, and there are several optimizations to deal with the problem of comparing two full text versions of a page.

When developing this software, I began with a simple approach of extracting the small edits from one page given two versions of the page's markup as the input. After manually verifying that it correctly extracts small edits as defined in Section 2.1, I would introduce optimizations (described in Section 2.4), verifying that the simple program and the optimized program produced the same output. Through this method, I was able to develop a system that efficiently extracts small edits from an entire history dump.

Chapter 3

FILTERING TURKISH SPELLING MISTAKES FROM WIKIPEDIA'S SMALL EDITS

As evidenced by the edit-type taxonomies created in several studies (Daxenberger and Gurevych, 2013; Grundkiewicz and Junczys-Dowmunt, 2014; Bryant et al., 2017), Wikipedia edit types are quite numerous, and many types of edits may fit into the category of "small edits" (as introduced in the previous chapter). Thus, it is not sufficient to directly learn from the extracted small edits; further filtering should be performed. This chapter introduces data considerations for designing such a system: what kinds of edits should be considered spelling mistakes? An active learning approach using a random decision forest is presented, and the resulting data set is made publicly available.[1] Although the final classifier is specific to Turkish, such a process can be used to design spelling mistake filters for other languages.

3.1 Classification Task

In this chapter I build a binary classifier, which filters the extracted small edits, as described in Chapter 2. Each edit is labeled as "spelling correction" or "not a spelling correction".

When training the model, I don't explicitly build a taxonomy of what types of "spelling errors" are allowed, but guidance for what can be considered a spelling error is provided in Section 3.3.

Although I extract small edits with context, input to the model is context-free: I consider the original form of the word (or group of words) and the edited form (for example: ("teh", "the") would be a common input for an English version of this task).

---

[1] `https://github.com/tguinard/TurkishWikiSpellingMistakes`

3.1.1 Context-Free Model and its Limitations

Although the input is context-free, the model also aims to detect errors that would be context-dependent: for example in English "affect" and "effect" are often confused, and determining what the correct form is relies on context since by themselves, both forms are valid. Therefore, the goal of this classification model is not to guarantee that the edited form is correct, it just determines whether the original-replacement pair *looks like* the correction of a spelling error. That is, the model does not evaluate the fitness of either form, but detects the edit-type intention based on characteristics of the edit itself. The key assumptions taken by this approach are: (1) spelling correction-type edits can be detected without the context, and (2) the edited form is correct.

Assumption (1) may be broken in cases of morphological ambiguity, which comes into play in many suffix change-based edits. See Section 3.3 for an example of how "arkadaş" → "arkadaşı" may or may not be a spelling correction, depending on the morphological analysis of "arkadaşı", which can only be determined from context. I think this is an important class of corrections that warrants future study, but due to the complexities of this issue, I leave it as out of scope for this study.

The main issue that breaks assumption (2) deals with capitalization-based edits. My final model marks essentially all capitalization edits as spelling corrections, but I have seen several instances where the edited form is not the most common form, which I also consider as being "not correct". For example, some Wikipedia editors capitalize an entire person's name or other named entity (e.g. "Atatürk" → "ATATÜRK"). Another example of editing to a less common form (at least in the Wikipedia medium) is changing the capitalization of a month to be lowercase ("Ağustos" → "ağustos"). In both of these instances, the edited form is recognized as a valid word by the morphological analyzer, but it is not in the standard form according to Wikipedia's conventions. I estimate about 5% of capitalization edits, or about 2% of the final corpus, breaks assumption (2) due to this phenomenon.

3.2 Previous Work

3.2.1 Rule-Based Systems

Previous studies that extract data from Wikipedia's edit history to aid NLP tasks had various nontrivial filtering requirements, and most of the time, filtering was performed with rule-based systems.

Grundkiewicz and Junczys-Dowmunt (2014) developed a rule-based method to categorize small edits: spelling correction, rewording, paraphrasing, casing change, information addition, broken markup fix, etc. Similarly, Bryant et al. (2017) created a rule-based categorizer for grammatical error correction (about 50 rules total), but did not include many rules specific to spelling correction.

Max and Wisniewski (2010) developed a simple heuristic for creating a spelling correction corpus for French. A single token must be changed, which must have certain characteristics, such as not including special punctuation symbols or numbers, and the corrected word must be in the dictionary. If the original token was a real word, keep the pair if they are within edit distance 3 of each other, and otherwise, keep if within edit distance 5. Edits that were purely changes in punctuation or capitalization were discarded.

3.2.2 Machine Learning Classifiers

Daxenberger and Gurevych (2013) created a categorizer for general Wikipedia edits, with many categories including information addition, information modification, spelling correction, reverting a previous change, vandalism, and paraphrasing. They extracted features based on metadata and textual properties, and experimented with several multi-label classifiers, observing the best performance with the Random k-Labelsets (RAKEL) algorithm. When developing the edit type taxonomy, one important distinction they pointed out was the difference between meaning and non-meaning changes. In my system, spelling correction should always be considered a non-meaning change.

Adler et al. (2011) and Javanmardi et al. (2011) both implemented a system for vandalism detection in Wikipedia edits, and the best model of both studies was a random decision forest, with various features about edit metadata and textual content. Although the application is quite different from my own, the design of these studies is most similar to my approach. The challenge with detecting vandalism edits is that vandals have a variety of motivations, and thus different instances of vandalism can look quite different. While some forms of vandalism may be more obvious, such as inserting vulgar words into text, more subtle forms exist: for example, commercial entities advertising their products, and the deletion or distortion of information (Javanmardi et al., 2011). Similarly, I assume that Turkish spelling errors may be made for a variety of reasons, so I model the problem of detecting spelling mistakes as a random decision forest.

3.3 Definition of a Spelling Error

The most important criterion I consider when identifying a spelling correction is that there is no information change. As a general rule of thumb, a human should be able to guess (with reasonable accuracy) what the author meant. I also assume that spelling errors have low-dependency fixes, generally not relying on complex grammar rules. Style-based choices (where both the original and edited version are linguistically acceptable, but one version is preferred based on an individual's taste), are also generally not considered spelling corrections.

A common classification of spelling errors across the literature is cognitive and typing mistakes. A cognitive mistake could be purely phonetic in nature ("recent" vs "recint"), or the author may mistake one word for another ("affect" vs "effect"; "recent" vs "resent"). A typing mistake is more mechanical in nature ("the" vs "teh"). For languages including Turkish, "asciification" is another common issue ("şaşırmak" vs "sasirmak").

There are many classes of errors that are more ambiguous as to whether they should be considered "spelling errors"

- *Informal writing:* Should an informal writing, such as that seen on social media, be

corrected to the formal version: should "cuz" be corrected to "'cause" or even "because"? In Turkish, should "dakka" be corrected to "dakika"?

- *Following different conventions:* For example, is it wrong to write "USA" as "U.S.A.", "Usa", and "usa"?
- *Change in language:* This type of edit is especially common with named entities: Should the European captain who came to the Americas in 1492 be called "Christopher Columbus", "Cristoforo Colombo", "Kristof Kolomb"?
- *Information change:* Can a factual change of information be considered a spelling mistake: "Christopher Columbus arrived in the Americas in 1492/2492/1493"?
- *Suffix change:* Should wrong or missing morphological information be considered a spelling mistake or a more complex grammar mistake? Consider "I have two cat" vs "I have two cats"

To account for these cases, I decided to largely follow the conventions of the Wikipedia medium (see Table 3.1 for a summary):

- *Informal writing:* Since Wikipedia contains mostly formal writing, formality issues are treated as genuine spelling corrections, so long as the lexical forms are sufficiently reminiscent of each other ("gonna" vs "going to" is more likely to be an intentional use of informal language, while "goin to" vs "going to" may be due to a typing mistake). For formal texts in Turkish, it is reasonable to correct "dakka" to "dakika". In the Wikipedia corpus, it is actually quite rare to see an informal form of a word, so this isn't a major issue for formal texts.
- *Following different conventions:* If the convention is implemented relatively consistently across Wikipedia, the most common form is accepted as correct. For example, in the Turkish Wikipedia, one of the most common small edits is changing "ptt" or "Ptt" to "PTT" (PTT (Posta ve Telgraf Teşkilatı) is the national postal service in Turkey). This is a valid spelling correction, as PTT is consistently used as the correct form in this corpus.
- *Change in language:* I interpret the decision of which language to use for named

entities as a style choice, thus these are not considered spelling errors. Furthermore, for ordinary language (non named entities), changing the language is considered a complex process out of the scope of spelling correction.

- *Information change:* Factual information changes may be because of a typo, but since we cannot generally suggest a correction without general world knowledge, this is out of scope of automatic spelling correction.
- *Suffix change:* Changing the morphological suffix of a word is one of the most common types of small edits in Turkish, but a lot of the time, information is being modified. In a minority of cases, we may consider that a change in ending is a spelling correction, generally whenever the correct form is orthographically close to the original form, but fits much better in the context. Different examples of these errors and ways of interpreting how to classify them are summarized in Table 3.2. I would consider Example 1 to be information change. Example 2 is information change, but may be interpreted by some as a spelling correction since the change to "annesi" would be the most common/logical form of that sentence. Examples 3 and 4 would be spelling mistakes because the original is erroneous, and the correction is a reasonable choice given that it is orthographically similar and fits the context.

For Turkish, it seems that suffix changes are among the hardest edits to classify as a spelling correction or not. One issue is that when identifying the edit type, we may need to look at the context in which it's used. For example, when ignoring the context, it is impossible to tell whether ("arkadaş" → "arkadaşı") is a spelling correction or not (considering Table 3.2). On the one hand, it could be the same type of correction as Example 3, where the author originally forgot to use the accusative case correctly. Alternatively, it could be the same type of correction as Example 2, as the final "ı" in "arkadaşı" may also be the 3rd person possessive suffix (i.e. the change is "(the) friend" → "her friend"). Some changes can be more easily classified, such as ("güçtü" → "güçtür"), as even without looking at context, we can infer this is likely a spelling correction since the semantic roles of the "tü" and "tür" suffixes are significantly different. For this thesis, when building the classifier, I mark ambiguous

Edit Type	Judgment
Informal writing	Not a spelling correction
Following different conventions	Spelling correction if edited form is more common
Change in language	Not a spelling correction
Information change	Not a spelling correction
Suffix change	Spelling correction if it's not an information change

Table 3.1: Summary of different spelling correction issues

examples such as ("arkadaş" → "arkadaşı") as non-spelling corrections, but mark examples such as ("güçtü" → "güçtür"), where we can reasonably guess the edit intention without context, as spelling corrections.

3.4 Candidate Pre-Filtering

A reasonable fundamental assumption to make about spell correction is that the corrected form will be a valid word in the language. For this reason, I filter all edits to only include those that only contain words recognized by the TRmorph morphological analyzer (Cöltekin, 2010; Çöltekin, 2014).

Additionally, the following types of edits were pre-filtered

- Pure "insert" operations: an entire word was inserted
- Pure "delete" operations: an entire word was deleted
- Any change including a token greater than 100 characters (most likely junk)
- Any edit where only non-apostrophe punctuation was changed
- Any edit where only numbers were changed
- Any edit where only the diacritics "â", "î", "û" were changed. These are often found in some loanwords such as "hikâye", but they are not universally used

#	Original	Correction	Analysis
1	Defterim var *I have a notebook*	Defterin var *You have a notebook*	There is a change in information. This is out of scope of a spelling correction.
2	Anne onu çağırdı *The mother called him*	Annesi onu çağırdı *His mother called him*	Both forms are grammatically valid, but the correction is pragmatically more likely. However, it is difficult to predict the correct ending that is missing, as the ending adds new information that may not be discernible from context
3	Arkadaşım gördüm *(Incorrect accusative case usage)*	Arkadaşımı gördüm *I saw my friend*	In order to generally solve errors of this sort, we need knowledge on how Turkish syntax works. This error may be a typo (failing to hit the last key in "Arkadaşımı"), or it could be a grammar mistake (failing to use the accusative case). I think this should ideally be considered a spelling mistake because there is truly a quick fix which most people can agree on.
4	Bilgi güçtü *Knowledge was hard*	Bilgi güçtür *Knowledge is power*	Both sentences are grammatically correct, but it is most likely a typo because the two orthographic forms are quite similar, but the meaning of the corrected version is more sensical

Table 3.2: Examples of various morphological suffix-specific edits, and issues with classifying as spelling errors

3.5 Random Decision Forest Classification Model

A random forest model was used, where all features were based only on the edit itself (the context was ignored). The decision trees were implemented with Python sklearn's DecisionTreeClassifier, and min_samples_leaf was set to 2.

The input data was initially split halfway (50/50) into two sets: the validation and test sets. Training data was generated by iteratively sampling from the validation set and was manually labelled (as "spelling correction" or "not a spelling correction") using guidelines as described in the Section 3.3.

3.5.1 Features

The following features were used

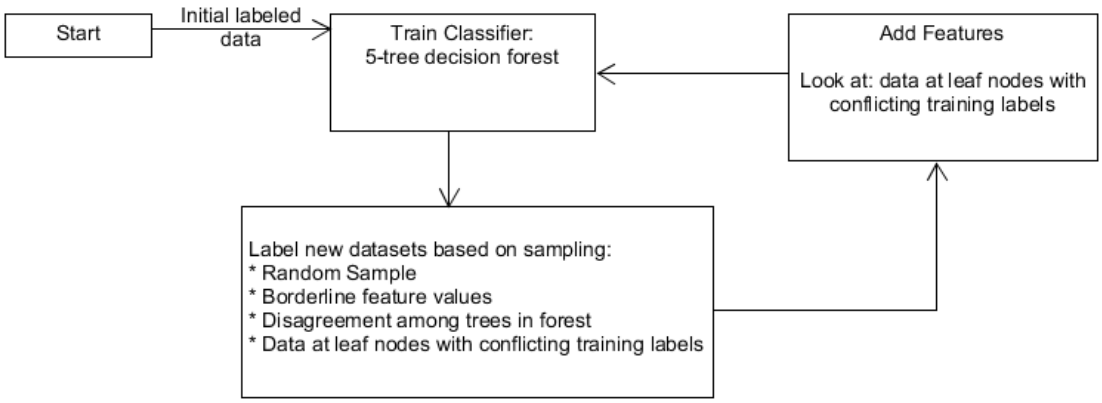
- length(original) - length(corrected)
- Levenshtein distance of original and corrected forms
- Levenshtein distance after converting to lowercase
- Levenshtein distance excluding spaces
- Levenshtein distance excluding apostrophe (')
- Levenshtein distance excluding dash (-)
- Levenshtein distance, where each consecutive repeated letter is ignored
- Levenshtein distance, normalizing for voiced/unvoiced consonant shifting (each pair ("ç", "c"), ("t", "d"), ("p", "b"), ("k", "ğ") is treated as a single letter)
- Levenshtein distance of the asciified versions (e.g. "şaşırmak" → "sasirmak")
- Levenshtein distance, where all vowels are deleted
- Levenshtein distance of the stem
- Levenshtein distance of the stem converted to lowercase
- Levenshtein ratio of each prefix and suffix of length n, where n is between 1 and 3

Additionally, the levenshtein distance after each of the following operations were added as features:

- Convert to lowercase
- Remove apostrophe
- Remove hyphen
- Remove space
- Remove double letters
- Normalize voicing (“ç” → “c”), (“t” → “d”), (“p” → “b”), (“k” → “ğ”)
- Asciify (şaşırmak → sasirmak)

3.5.2 Active Learning

Figure 3.1: Summary of active learning process



In order to minimize the amount of training data needed to be labeled, and to better account for edge cases, I used an active learning strategy to select new training data to be labeled after an iteration of training the model, generating these data sets after each iteration:

1. Random sampling of the validation set

2. Sampling validation set members that change classes if the feature vector is changed by a small amount

3. Sampling validation set members that are assigned to different classes by different trees in the forest.

4. Sampling validation set members who share the same leaf node with training data with conflicting classes

Labeling new data from dataset 1 would ensure that all frequent mistake types would appear in the training set, while datasets 2-4 prefer data that the model is not confident about. This active learning strategy assumes that any given iteration of the model does a good job at classifying a large part of the data, and the optimal way to improve the model is to add more examples from classes that the model is uncertain about.

New features were also added iteratively by analyzing the training data that was unable to be classified correctly. The active learning process is summarized in Figure 3.1.

Whenever both the size of data sets 2-4 and the training data accuracy seemed to converge, I stopped the training process. On the first iteration I used a random sample of 1000 words in the training set. I performed about 25 iterations, and at the end, there were 1623 positive examples (small edits that are spelling mistakes) and 2056 negative examples (small edits that are not spelling mistakes).

3.6 Results

To measure the performance of the model, I randomly sampled 300 data points from both the validation and test set, and manually labelled them (without looking at the model's output). Recall that the validation/test split was 50/50, and the validation set was where I sampled the training set from, so it is possible that some training examples are included in the set of 300 random data points (I don't explicitly exclude training examples, as the training set was not built by uniform sampling, so I believe this is a more intuitive comparison). The test set was not used at all when training the model.

When interpreting this performance data, the caveat is that this is based on my own interpretation of what should be considered a spelling error, which, even after defining prop-

	Precision	Recall
Training	0.984 (1539/1564)	0.948 (1539/1623)
Validation	0.96 (101/105)	0.94 (101/107)
Test	0.96 (110/114)	0.96 (110/114)

Table 3.3: Precision and recall of spelling correction classification

erties a spelling error should have (Section 3.3), is still highly subjective. For this reason, if someone else were to repeat this measurement based on their own judgments, I would expect the precision and recall to be lower. These results, however, are a good measure of how consistently the model performs, and how well it models my own subjective view of what a spelling error in Turkish is.

The precision and recall for the entire training set (3679 data points) is also reported. Note that the precision and recall are expected to be high on the training set, since this is the data the model was trained on. However, precision and recall are not expected to be 100% because of branching constraints (a node will not further branch if it has only 2 members). Also note that the data in the training set was selected via active learning (Section 3.5.2), so its distribution is different from the validation and test sets (which are uniformly random). The training data metrics should primarily be used to check if there is obvious overfitting in the model. It appears that there is not a high degree of overfitting, though the model does have better precision on the training data than on the other data sets.

The model performs well in terms of both precision and recall in both the test and validation sets, performing about the same on both data sets (Table 3.3). This indicates that the model is a good representation of my judgment for Turkish spelling errors, and it has consistent performance across the entire input data.

Table 3.4 lists instances where the model gives the wrong label. Some of the false positives are due to stems with different meanings but relatively short edit distances: ("önermeleri" $\rightarrow$ "teoremleri"; "sabah" $\rightarrow$ "akşam"). The edit ("bölündeki" $\rightarrow$ "bölgedeki") also appears to be one of these examples, except the apparent stem "bölün" is nonsense in this context. This

Error Type	Original Form	Edited Form
False Positive	bölündeki	bölgedeki
False Positive	Fonksiyonlarda TürevKuralları	Fonksiyonların Türevleri
False Positive	iddaalar	iddialarda
False Positive	TA	TAA
False Positive	önermeleri	teoremleri
False Positive	özelliktir	özelliğidir
False Positive	sabah	akşam
False Positive	verememiştir	vermemiştir
False Negative	bisikletileri	bisikletleri
False Negative	Okyanusu'nin	Okyanusu'nun
False Negative	olmasaada	olmasa da
False Negative	Rekifa	Refika
False Negative	konusund aönemli adınlar	konusunda önemli adımlar
False Negative	topraðý n	toprağın
False Negative	eyalerin	eyaletin
False Negative	I2005	2005
False Negative	ger	geri
False Negative	do rulu unu	doğruluğunu

Table 3.4: Incorrectly classified by model

is an example where subjective judging comes into consideration: I thought it was unlikely that someone would write "bölün" when they meant "bölge", since they neither sound alike nor have similar typing motions (perhaps the original author had "bölümdeki" in mind?). Sometimes the model fails on multiword inputs, which may be because I designed most of the features with single words in mind; perhaps this could be solved with further refinement of the features. The model sometimes marks two words with different affixes as positive, when the words actually have significantly different meaning (verememiştir "could not give" vs vermemiştir "didn't give"; özelliktir "is a property" vs özelliğidir "is its property"). For the false negatives, the model seems to have the most trouble on the ends of words, which makes sense given the complexity of dealing with changes to the suffixes.

Chapter 4
ANALYSIS OF TURKISH SPELLING ERRORS

This chapter analyzes what kinds of spelling errors are seen, analysing the nature of the Turkish spelling correction problem. To my knowledge, the data collected in this study is the largest corpus of Turkish spelling mistakes to date.

4.1 Previous Work

Mays et al. (1991) analyzed spelling errors in English and found that around 80% are within 1 edit distance of their correct form. Zesch (2012), however, noted that the average edit distance between real-word errors in both English and German datasets was 1.4, higher than predicted by Mays et al. (1991) so this figure may vary for different classes of edits.

Max and Wisniewski (2010) analyzed spelling errors they extracted from Wikipedia and found that the frequency for real-word and non-real-word spelling corrections in French was about equal (74,100 vs 72,493 extracted pairs, respectively). They also noted that pure diacritic changes were quite common (32.39% of nonword errors). Most of the real-word errors observed shared the same root, as they were due to a change in plurality or gender.

Grundkiewicz (2013), who also extracted errors from Wikipedia, included a similar analysis for the Polish language. From a rule-based categorizer, the edit types relevant to my task are diacritic changes (241,772 pairs extracted), case changes (220,533 pairs), spacing modification (13,782 pairs), spelling correction (356,762 pairs), and probable spelling corrections (157,043 pairs).

To my knowledge, this is the first large-scale analysis of Turkish spelling mistakes. Knowing what kinds of mistakes are typical in the Turkish language can help clarify what kinds of problems need to be solved.

4.2 Analysis

4.2.1 Corpus Overview

The corpus developed in Chapters 2-3 is made available to the public[1]. For each spelling correction, the following information is included:

- Original token(s)
- Corrected token(s)
- Original left context (starting at the beginning of the previous sentence, or the start of the paragraph)
- Corrected left context (different from original left context only if other edits were made to the section)
- Original right context (ending at the end of the next sentence, or the end of the paragraph)
- Corrected right context (different from original right context only if other edits were made to the section)
- Mistake Category (based on the first column of Table 4.2)
- Is the original a *word* or *nonword*? A word is a token with a valid morphological analysis according to TRmorph (Cöltekin, 2010; Çöltekin, 2014). If the original contains multiple tokens, *word* means that every token is a word; *nonword* means that at least one token is a nonword.

A total of 783,332 of edits were marked as spelling corrections, which consisted of 287,661 unique edits, when the context is ignored (number of unique the context-free (original form, edited form) pairs). Table 4.1 gives corpus statistics broken down by the number of space-separated tokens in the original and edited forms. Some key takeaways from this table are: 74.5% of edits involve changing a single word into another single word, and the edited version tends to be slightly longer than the original version (which has a length of 8.5 characters on average).

[1] `https://github.com/tguinard/TurkishWikiSpellingMistakes`

Original Token (*word*) Count	Edited Token (*word*) Count	Edits Count	Unique Edits Count	Average length(original) (*characters*)	Average length(edited) (*characters*)
Any	Any	783332 (100%)	287661 (100%)	8.5	8.6
1	1	583364 (74.47%)	193159 (67.15%)	7.3	7.4
1	2	71992 (9.19%)	24946 (8.67%)	8.6	9.6
1	3	374 (0.05%)	316 (0.11%)	14.7	16.8
2	1	43911 (5.61%)	10155 (3.53%)	8.6	7.7
2	2	64587 (8.25%)	44371 (15.42%)	15.3	15.3
2	3	1665 (0.21%)	1580 (0.55%)	18.0	19.4
3	1	127 (0.02%)	113 (0.04%)	13.1	11.3
3	2	1064 (0.14%)	1044 (0.36%)	17.8	17.0
3	3	16248 (2.07%)	11977 (4.16%)	22.2	22.1

Table 4.1: Corpus statistics

4.2.2 Change Type Classification

Table 4.2 summarizes statistics about spelling error types as seen in the corpus. The word vs. nonword distinction is based on whether the input is recognized by TRmorph (Cöltekin, 2010; Çöltekin, 2014) (recall from Chapter 3 that only edited forms that are recognized by TRmorph are included). Over half of the spelling correction pairs had original versions that were recognized by the morphological analyzer (already a valid word/set of words). Thus, it is clearly not enough to detect spelling mistakes by checking whether the word is morphologically valid.

A mistake/correction pair was considered to be a capitalization change if the lowercase versions of the words were the same (accounting for the Turkish-specific handling of i's (İ → i, I → ı)). A pair was considered a diacritic change if the result of "asciification" of the word[2] was the same. Space and apostrophe changes are based on whether the two versions are equal after the appropriate character (space or apostrophe) is removed. For character-level changes (insertion, deletion, substitution, swapping), the change was categorized as

[2]Based on the result of the Unidecode library: `https://pypi.org/project/Unidecode/`

	* → word	Nonword → word	Word → word
Total	100% (856177)	100% (357270) (45.6% of grand total)	100% (492821) (54.4% of grand total)
Capitalization change	31.6% (247248)	18.5% (66086)	42.5% (181162)
Diacritic change	18.1% (141847)	33.2% (118542)	5.5% (23305)
Space change	12.3% (96361)	6.7% (23805)	17.0% (72556)
Apostrophe change	5.2% (40451)	0.3% (1133)	9.2% (39318)
Apostrophe and capitalization change	1.7% (13406)	0.6% (2232)	2.6% (11174)
Character substitutions	11.5% (89840)	15.8% (56507)	7.8% (33333)
Character deletions	7.6% (59334)	8.8% (31429)	6.5% (27905)
Character insertions	4.5% (35478)	6.3% (22496)	3.0% (12982)
Character swaps	1.1% (8822)	2.2% (7800)	0.2% (1022)
Multiple change types	6.5% (50545)	7.6% (27240)	5.5% (23305)

Table 4.2: Distribution of types of spelling errors

that specific category if the calculation of the (minimal) Damerau-Levenshtein distance only included that one type of character-level change.

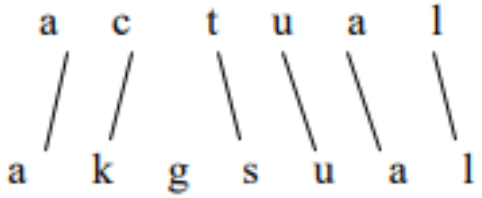
Together, diacritic and capitalization changes accounted for about half of the corrections. Changes in spacing were also a significant portion of the errors (12.3%). Apostrophe modification, or apostrophe+capitalization modification, which is related to named entities (proper nouns have apostrophes before their suffixes) accounted for 6.9% of the changes. Of the remaining changes that involved one type of letter transformation (insertion, deletion, swapping, substitution), substitution was the most common operation, followed by deletion, insertion, and swapping. In total around 93.5% of the corrections could be cleanly classified into one single type of change, which suggests that a rule-based/modular model, such as the system proposed by Torunoğlu-Selamet and Eryiğit (2014) could be rather effective (see Section 6.1.5 for further discussion).

4.2.3 Character-Level Change Classification

Brill and Moore (2000) introduced an influential English spelling correction algorithm that

assumes some character sequence replacements have greater frequency than others. To evaluate whether this is true for Turkish according to this data set, I applied Brill and Moore (2000)'s simple initial alignment model to this data set and analyzed all the candidate unigram to unigram replacements; I assume a simplified version of Brill and Moore (2000)'s model where the original and edited substrings have length 0 or 1. For example, if the input pair is (actual, akgsual), I would use the alignment shown in Figure 4.1, and consider (c, k) and (t, s); I ignore changes where either the original or the replacement is empty, just to focus on character replacement operations. I used python's difflib library to align equal letters, and used a greedy approach to pair up unaligned letters, alternating matching the first available pair on the left, then on the right (for each pair of unaligned strings). The statistics in Table 4.3 show the results of this process applied to the entire spelling correction corpus that was built in Chapter 3, and also filtered on errors according to their type as in Table 4.2.

Figure 4.1: From Brill and Moore (2000): initial character-level alignment of spelling correction



Based on the analysis in Table 4.3, it is clear that the assumption made by Brill and Moore (2000) is quite relevant; the character confusion distribution is far from uniform. When looking at the pure character-substitution changes, the majority of changes substitute one consonant for another or one vowel for another. Substitution based on only consonant voicing change is an example of a class that has rather high confusion probability.

Another observation from this data is that overall, this dataset has a lot of diacritic

	Overall	Character Substitution	Multiple change types
Total: all single-character changes	100% (891656)	100% (104770)	100% (74957)
Change in case	53.1% (473261)	0.9% (958)	24.4% (18308)
Diacritic change	32.5% (290154)	10.4% (10862)	29.4% (22042)
Consonant → consonant (not equal after normalizing voicing)	3.3% (29083)	21.9% (22922)	8.1% (6038)
Vowel → vowel	4.4% (39649)	32.9% (34498)	6.6% (4920)
Consonant → consonant (only voicing is different)	3.7% (32938)	29.4% (30782)	2.9% (2148)
Original or replacement is punctuation or space	2.0% (18094)	0.3% (295)	23.5% (17631)
Vowel → consonant or consonant → vowel	1.0% (8477)	4.3% (4453)	5.2% (3870)

Table 4.3: Distribution of single character substitutions

and capitalization character-level changes, so these operation types warrant specific attention. For edits with multiple change types, diacritic and capitalization changes are also quite common, but overall, this type of edit makes up only 6.5% of edits (Table 4.2), and in "character substitution" changes, diacritic and capitalization changes are relatively low, further supporting the idea that these are distinct change types, and a modular approach is advantageous from this point of view.

4.2.4 Edit Distance Distribution

When discounting pure diacritic-, capitalization-, spacing-, and punctuation-related edits, the spelling corrections tend to be of low edit distance (96% have edit distance of 3 or less; Table 4.4), so a candidate generation approach like Oflazer (1996)'s (where all valid Turkish words within an edit distance n are generated) would be effective for the majority of these cases.

Damerau-Levenshtein distance	Freq
1	72.0% (175685)
2	19.4% (47243)
3	4.7% (11520)
4+	3.9% (9571)

Table 4.4: Distribution of Damerau-Levenshtein distance: including only character substitutions, insertions, deletions, swaps, multiple change types, as classified in Table 4.2

Chapter 5

TEST DATA SETS

This chapter describes datasets used for evaluating different spelling correction systems, both developed in this thesis (Chapters 6, 7) and available from previous research (Torunoğlu-Selamet et al., 2016; Koksal et al., 2020).

5.1 Wikipedia

I use the Wikipedia data set I developed (Chapter 2-4) for both training and testing purposes. For the remainder of the paper, I use an 80/20 split of the data into training/test sets. The 20% test set is also used to test other systems. The training data is primarily used to build an error model according to Brill and Moore (2000), and is additionally used in some experiments with SMT models in Chapter 7.

When testing the systems and sub-systems of Chapter 6 and 7, I further sample from this test set (generally testing on 1000 data points), and different sampling is used based on what type of error the component is designed to handle. For modules that are only meant to handle one type of error, I prefilter the test set based on the error type as defined in Table 4.2, according to what kind of error that module was designed to handle. When testing an end-to-end system, I simply take a uniform sampling of the data. I perform each kind of sampling once, so we can directly compare the results of competing models (two models meant to handle the same type of error, or two end-to-end systems).

Operation	Count
Swap	37
Substitution	314
Insertion	332
Deletion	333

Table 5.1: Damerau-Levenshtein edit types in the noisy channel Twitter test set

5.2 Twitter

5.2.1 Noisy Channel: "Twitter2016"

In order to evaluate how this study builds upon the results of previous Turkish spelling correction research, I use the test set from Torunoğlu-Selamet et al. (2016), which is a list of 1,016 spelling errors along with their corrections. This data was manually extracted and annotated from a corpus of Turkish tweets (Sak et al., 2011), and it excludes errors that are assumed to be handled by previous modules of Torunoğlu-Selamet and Eryiğit (2014), e.g. deasciification and capitalization. This data set is also used for evaluation in Büyük (2020).

By design, this data fits the noisy channel model very well. All of the correction pairs (whether by design or by accident) differ by exactly one character insertion, deletion, substitution, or adjacent swap (1 Damerau-Levenshtein distance). Types of edits are recorded in Table 5.1.

For the remainder of this thesis, I refer to this set as Twitter2016.

5.2.2 General Spelling Correction: "Labeled Twitter"

Koksal et al. (2020) built a small data set[1] of more general spelling correction/normalization errors from the same Twitter source data set (Sak et al., 2011). This data set has 2,000 pairs of original/corrected tweets, and each OOV token is categorized; see Table 5.2 for a list of categories. Note that the categorization is performed per OOV token, and some tokens

[1] `https://github.com/atubakoksal/annotated_tweets`

Error Type	Is true error?	Frequency (%)
Deasciification	ill-formed	44.94
Accent	ill-formed	11.22
Proper Name	ill-formed	9.20
Intentional Char	ill-formed	9.02
Separation	ill-formed	7.68
Foreign Word	well-formed	4.92
Unintentional Char	ill-formed	4.69
Social Media Phrase	well-formed	2.50
Abbreviation	well-formed	2.37
Adjacent	ill-formed	1.36
Neologism	well-formed	0.96
Vowel	ill-formed	0.63
Phonetic Substitution	ill-formed	0.52

Table 5.2: From Koksal et al. (2020): error types in Twitter general spelling error corpus

are actually correct ("well-formed"). Thus, I only consider the tokens labeled as "ill-formed". Also, since Twitter normalization deals with issues that are not as relevant to formal writing (and are out of scope of this thesis), I create a modified data set, where some of the categories of the ill-formed tokens are excluded:

- Accent
- Intentional Char
- Vowel
- Phonetic Substitution

For the remainder of this thesis, I refer to this set as Labeled Twitter.

5.3 Tatoeba

In order to test on a third data set that is not a development or training set for my system or the systems of previous studies, I crawl tatoeba[2], and extract instances where sentences are edited. Tatoeba is a database of sentences manually translated into various languages, where

[2] `https://tatoeba.org/`

users are encouraged to contribute sentences in their native languages. Like Wikipedia, the content is generally of high quality, but typos may occur, and they are often fixed by an admin or the original sentence author.

To create a dataset of spelling corrections, I filter sentence corrections where the corrected form is a valid word and the normalized (asciified, lowercase) original is within 5 Damerau-Levenshtein distance of the normalized corrected form. I shuffle the candidates and select about the first one thousand sentence pairs that are true spelling errors (manually judged by me; I actually selected 1170 sentences).

Chapter 6

CASCADED, STATISTICAL SPELLING CORRECTION

Perhaps the most popular design for spelling correction systems is the so-called noisy channel model (Kemighan et al., 1990). This model predicts $P(s|w)$: the most likely correct word s given an incorrect spelling w by combining an error model $P(w|s)$ and a language model $P(s)$

$$argmax_s P(s|w) = argmax_s P(w|s)P(s)$$

However, such a design is problematic when accounting for issues such as diacritic and capitalization errors, which may be applied more systematically (errors are not made independently of each other).

The spelling correction problem as defined in this thesis is actually somewhat of a hybrid between a traditional spelling correction problem (where the noisy channel is directly applicable) and a normalization problem (often studied in terms of informal writing found in social media). Torunoğlu-Selamet and Eryiğit (2014) introduced a cascaded approach for Turkish social media normalization, specifically for Twitter, where they consider different issues (such as capitalization normalization, accent/pronunciation normalization, diacritic normalization, and spelling correction) in a modular fashion, and they assume that a noisy channel spelling correction model can be applied as a final step. This chapter presents a system that follows this general approach.

6.1 Previous Work

6.1.1 Baseline Noisy Channel Models

In a noisy channel model, an error model typically estimates probabilities of mistakes on the surface level: generally character-level operations. Two popular baseline models are Levenshtein distance and Damerau-Levenshtein distance (Levenshtein, 1966; Damerau, 1964), where the former is the count of character insertions, deletions, and substitutions, while the latter also includes adjacent character swaps. While in English, one can use a fixed-length lexicon to generate all valid words within a certain edit distance of an input string, this is not a trivial problem in Turkish, as in principle, the vocabulary is infinite due to Turkish's productive morphology. Oflazer (1996) presented an algorithm that can generate valid Turkish words within a certain Levenshtein or Damerau-Levenshtein edit distance of an input string by doing a graph search over an FST morphological analyzer.

The language model component predicts how likely the correct word is to be seen in such a context, and is generally implemented at the word- or token-level. A popular baseline is an ngram language model.

Pirinen and Lindén (2010) construct such a baseline error model/language model spelling correction system for Finnish, which, like Turkish, is a morphologically rich language. In their system, candidates are generated by the algorithm proposed by Oflazer (1996). All words with edit distance 1 are considered as candidates, or if none are found all words with edit distance 2. Then the candidate with the highest unigram probability is selected.

6.1.2 Error Models

Brill and Moore (2000) presented an improved "edit distance" statistical model, using probabilities for specific edit operations collected from a corpus of error/correction pairs. This system is able to identify common string transformations that are not well-modeled by Damerau-Levenshtein distance and discount simple but rare transformations. For example, in English, "ph" and "f" are likely to be confused, but "q" and "m" are not.

Torunoğlu-Selamet et al. (2016) experiment with various combinations of error and language models for Turkish spelling correction. Their best model consists of a unigram language model and an error model based on an implementation of the system presented by Wang et al. (2011), which itself is an improvement of Brill and Moore (2000). Their system is tested on data that closely represents the noisy channel model: the errors consist of character insertions, deletions, substitutions, and swaps (issues such as deasciification and accent/pronunciation normalization are assumed to be processed at an earlier pipeline stage as presented in Torunoğlu-Selamet and Eryiğit (2014)).

6.1.3 Language Models

Cucerzan and Brill (2004) suggest that the language model generally has a bigger impact than the error model on such noisy channel-based systems, which is supported by the results of Torunoğlu-Selamet et al. (2016), as their system that uses a unigram language model and no error model performs remarkably better than their system that only uses an error model.

Several variations have been proposed for a language model. A common choice is to use an ngram language model. Despite its simplicity, in some cases it can be competitive with more complex systems. Whitelaw et al. (2009) achieved high accuracy with web-scale language models. Using ngrams from the web worked especially well for adapting to new domains, given the size and diversity of the web corpus. Bergsma et al. (2009) demonstrated that combining various ngram counts (unigrams, bigrams, trigrams over various windows) produced superior results. Simply summing the log probabilities, using their SUMLM system worked surprisingly well, only being slightly outperformed under certain conditions by their system that learns weights via SVM.

Using word vector-based features has been explored in several studies. Fivez et al. (2017) incorporated word vectors into a language model, where one feature type to evaluate a candidate would be that candidate's cosine distance to words in the context. Göker and Can (2018) generated normalization candidates directly by clustering tokens found on social media by their word2vec value. The idea was that different formal/informal variations of

the same word would have similar word vector values. However, this makes the assumption that the exact mistakes will be found in a corpus ahead of time at high frequency.

Schaback and Li (2007) use a rich set of features to train an SVM as a candidate ranking model. The features capture a variety of linguistic information on the character-, phonetic-, word-, syntactic-, and semantic-levels. The character-level features are output from Brill and Moore (2000)'s error model. For the word-level, they use ngram frequencies, for the syntactic-level, they use POS ngram frequencies, and for the semantic level they use a semantic word vector model.

6.1.4 Real-Word Errors and Spelling Mistake Detection

Many studies make the distinction between real-word errors (where the original misspelling is a valid dictionary word) and nonword errors (where the original misspelling is not a word in the language). This kind of approach is designed to minimize correcting words that are actually spelled correctly. Words that are not found in the dictionary are assumed to be wrong, but words that are in the dictionary may be replaced only under specific conditions. Many studies (Brill and Moore, 2000; Torunoğlu-Selamet et al., 2016; Pirinen and Lindén, 2010) consider real-word errors to be out of scope due to the difficulty of mistake detection.

Golding and Roth (1999) develop a system specific to real-word errors, assuming that confusion sets (lists of words that are confused with each other) are known beforehand. Their model is essentially a specialized language model that uses two types of features, collocation and context words, to represent two kinds of information about the surrounding text. A collocation feature is of the form "w occurs within ±k words". A context words feature encodes the word values or POS tags for encompassing n-grams. Below are example features that are useful for disambiguating "weather" and "whether"

- cloudy occurs within ±10 words (collocation feature)
- _ to [verb] (context words feature)

Although it is common to segment the spelling correction problem into real-word and nonword errors, some studies handle both error types jointly. Whitelaw et al. (2009) do not

make such a distinction outright, and instead consider every token as a possible error. The final stage of their system decides whether the word really is a spelling correction. One way to model this is to simply maximize $P(s|w)$, and accept w only if $P(w|w)$ is higher than all other candidates. Another model they tried, which had superior performance, was to use a logistic regression model as the final step. The model includes features such as whether the input word was capitalized, the number of candidates seen at earlier steps, the token length, and the length of ngrams they were able to match.

Similar to Whitelaw et al. (2009), Schaback and Li (2007) also do not outright distinguish the tasks of nonword and real-word spelling correction, and the last stage in their process is a candidate ranking model with several features (see section 6.1.3).

In applications such as search engine queries, deciding not to distinguish real-word and nonword errors is especially advantageous, as many valid OOV words are likely to exist in search queries. Thus, a reranking approach is generally favored. For example, the search engine spelling correction system of Sun et al. (2010) relies on a 2-layer neural network to rerank probabilities for spelling alternatives of search queries.

6.1.5 Cascaded Systems

The system of Eryiğit and Torunoğlu-Selamet (2017), which is an improvement of Torunoğlu-Selamet and Eryiğit (2014), performs social media normalization as a cascaded process. The steps are outlined as follows:

- Replacement rules and lexicon lookup
- Case transformation
- Proper noun lookup
- Diacritic restoration
- Vowel restoration
- Accent normalization
- Spelling correction

The replacement rules and lexicon lookup is a rule-based step that accounts for abbre-

viation and domain-specific vocabulary normalization. The case transformation step uses rules to convert abnormal capitalization into either *Proper Noun Case* or *lowercase* based on surface form only, and the proper noun lookup component performs NER to determine whether each word should be capitalized based on its meaning, using the system of Şeker and Eryiğit (2012, 2017). Vowel restoration accounts for the phenomenon that is common in Turkish social media messages to leave out vowels ("sevyrm" vs "seviyorum"). The accent normalization step accounts for variations in spelling due to pronunciation ("gidicem" vs "gideceğim"). Both the diacritic restoration and vowel restoration steps use the system described in Adali and Eryiğit (2014). The spelling correction system assumes a noisy channel model, using the best system from Torunoğlu-Selamet et al. (2016).

There are a few other studies that explore the problems of diacritic and capitalization restoration for Turkish. The greedy prepend algorithm for decision list learning proposed by Yuret and De La Maza (2006) has been used to create libraries for Turkish diacritic restoration, most notably turkish-mode[1] for GNU Emacs by Deniz Yüret and turkish-deasciifier library[2] by Emre Sevinç. Ozer et al. (2018) proposed a method that uses word vectors to normalize diacritics in turkish tweets, where the cosine similarity between the candidate word and all other words in the tweet are computed.

In addition to Şeker and Eryiğit (2012, 2017), Turkish NER has been studied by Onal and Karagoz (2015) using word embeddings and Çelikkaya et al. (2013) using CRF.

6.2 Architecture

I take an approach similar to Eryiğit and Torunoğlu-Selamet (2017), but with modules specific to error types seen in the Wikipedia dataset. The main differences between our architectures is that I account for word fusion and word splitting corrections ("into" → "in to" or "in to" → "into"), while Eryiğit and Torunoğlu-Selamet (2017) account for informal writing styles in accent normalization and vowel restoration modules.

---

[1] `http://www.denizyuret.com/2006/11/emacs-turkish-mode.html`

[2] `https://github.com/emres/turkish-deasciifier`

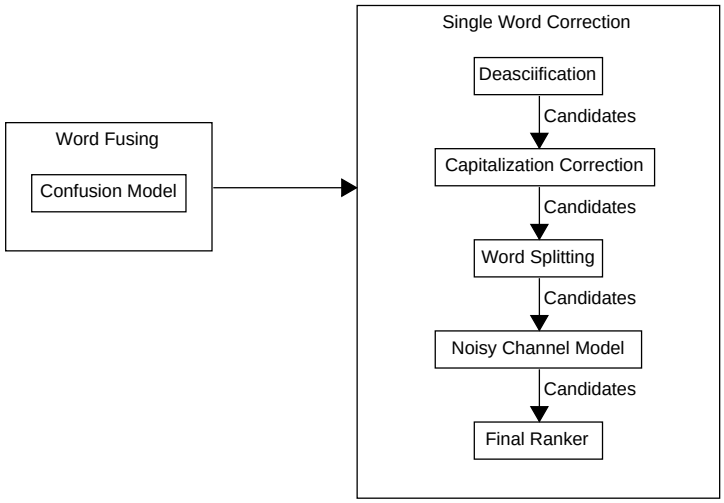


Figure 6.1: Cascaded spelling correction architecture

Figure 6.1 summarizes the model architecture. Correction is performed in two broad steps: one that fuses words in the input, and one that corrects single input words. The input to the first step is two adjacent tokens, and the input to the second step is a single token.

The architecture of this model takes advantage of the fact that most errors in the corpus are a single mistake type (see Section 4.2.2), but errors that require cooperation between multiple modules may not always be handled correctly. One example is writing "hiç birşey" instead of "hiçbir şey", a spacing mistake that is not considered "simple" by this model (it is not a pure "missing space" mistake nor a pure "extra space" mistake). Correct handling of this type of error would rely on the Word Fusing component to suggest "hiçbirşey" as an intermediate candidate, which depends on how likely the intermediate form is according to the language model. For this example, "hiçbirşey" is not suggested as an intermediate candidate in the final system, and the final output is "hiç bir şey", where only the Word Splitting model is triggered.

Many of the modules rely on using an ngram language model. For every ngram language

model, I used Wikipedia as a corpus (the latest version of each page[3]). For each page, I used mwparserfromhell to remove the markup, and I kept all plaintext before the "References" ("Kaynakça") section. In total, the token count of the extracted corpus is 131,860,039.

6.2.1 Word Fusing

Due to the observation that most spelling mistakes involving a space don't involve any other errors, and for simplicity in design, I assume that the only correction that can apply to two adjacent words is fusing them together ("in to" → "into"; "hiç bir" → "hiçbir").

If the fused form is a valid word, but at least one of the original two words is not, the candidate is accepted. If both forms are valid, the form with the highest frequency in the context is accepted. In order to determine if a word or two-word phrase is valid, I used both a morphological analyzer, TRmorph (Cöltekin, 2010; Çöltekin, 2014), and the official dictionary[4] of TDK (Türk Dil Kurumu; Turkish Language Association). I experimented with using the TDK data to disambiguate between single words that are commonly written as two separate words, or two word-phrases that are commonly merged together ("hiç bir" vs "hiçbir", "bilimkurgu" vs "bilim kurgu"). I experimented with using context-free frequency ("pseudo-unigrams" in Table 6.1) and SUMLM (Bergsma et al., 2009), using pseudo-unigrams and bigrams as in Table 6.1. SUMLM is a simple model which combines ngram statistics by summing their log probabilities. The SUMLM model for combining the unigram and two bigrams, as in Table 6.1 is given in equation 6.1. For all of the ngram models in this chapter, I used add-one smoothing.

$$SUMLM(w_i) = log(P(w_i)) + log(P(w_i|w_{i-1})) + log(P(w_i|w_{i+1})) \tag{6.1}$$

[3] `https://dumps.wikimedia.org/trwiki/latest/trwiki-latest-pages-articles.xml.bz2`

[4] `https://sozluk.gov.tr/`

	Original	Correction candidate
unigram	hiç bir	hiçbir
bigram1	Sanırım hiç bir	Sanırım hiçbir
bigram2	hiç bir şey	hiçbir şey

Table 6.1: Ngrams for candidates of "hiç bir" as seen in "Sanırım hiç bir şey yok"

6.2.2 Single Word Correction

Candidates Flow

In the single word correction pipeline, each step keeps candidates from the previous step as well as possibly adding new candidates. This approach is disadvantageous due to the possibility of candidate multiplication at each step, but in practice, the number of total candidates was small (2.13 candidates on average, from a sample of 1000 spelling errors from Wikipedia). This is because of the strict new candidate output criteria at each module: a candidate is only output if it beats the original in a language model, and only the candidates with the max score are output.

Deasciification

I experiment with a baseline ngram model and two previously developed deasciifiers:

- Baseline unigram model: accept the most common form according to frequencies in Wikipedia
- The deasciifier developed by Adali and Eryiğit (2014)[5], used in Eryiğit and Torunoğlu-Selamet (2017)
- A deasciifier[6] that uses GPA decision lists (Yuret and De La Maza, 2006)

Both previously developed deasciifiers are context-dependent, so I use the original text (with context) as input, but only look at the output of the target word (ignoring changes to the

[5] `http://tools.nlp.itu.edu.tr/Deasciifier`

[6] `https://github.com/emres/turkish-deasciifier`

context).

Capitalization

In this step, I also experiment with two different paradigms: using an ngram model and using an NER system.

- Ngram model: accept the most common capitalization according to ngram frequencies. The context tokens are assumed to be already corrected for capitalization. I experiment with unigram frequencies, SUMLM, and stupid backoff (try to match trigram, then try to match bigram, then match unigram)
- İTÜ NER System (Şeker and Eryiğit, 2012, 2017)[7], which was used in Eryiğit and Torunoğlu-Selamet (2017)

Word Splitting

In this step, the word is split into two parts at every possible position, and we consider instances where both components are valid words. A candidate is kept if its score is higher than the original form's. Similar to the word merging problem, I experiment with unigrams and SUMLM (same ngrams as used in Table 6.1).

Noisy Channel Spelling Correction

Figure 6.2.2 shows the general architecture for the noisy channel spelling correction module. Candidates are generated using the algorithm of Oflazer (1996) over an FST morphological analyzer. The error model follows Brill and Moore (2000), and I try variations of ngram language models.

Candidate Generation To generate candidates, I use the algorithm of Oflazer (1996), and implement it over the TRmorph morphological Analyzer (Cöltekin, 2010; Çöltekin, 2014).

[7] `http://tools.nlp.itu.edu.tr/Ner`

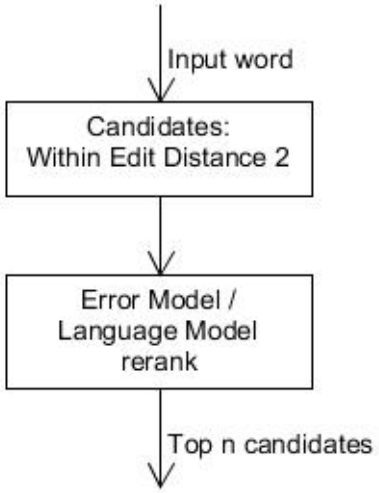


Figure 6.2: Noisy channel module architecture

TRmorph is an FST that runs in foma[8]. I used foma to extract the FST in ATT-format, and I implemented Oflazer (1996)'s algorithm in python. I generate all candidates within Damerau-Levenshtein distance 2 of the input string.

Error Model I collected string transformation statistics as described in Brill and Moore (2000) over the Wikipedia training set (80% of the collected corpus), only including the strings that I classified as "noisy channel" (see Chapter 4), making the training size 166,431 word pairs. $P(w|t)$ was calculated as described in Brill and Moore (2000).

Language Model Torunoğlu-Selamet et al. (2016), a previous implementation of a noisy channel spelling correction system for Turkish, only implemented a unigram language model. I experimented with various ngram models (unigram, SUMLM, stupid backoff), and I found the highest success with stupid backoff.

[8] `https://fomafst.github.io/`

Final Ranker

I experimented with logistic regression (similar to Whitelaw et al. (2009)), experimenting with the following features:

- Ngram probabilities (1 feature for each 1-, 2-, and 3-gram that includes the word)
- POS ngram probabilities (1 feature for each 1-, 2-, and 3-gram that includes the word)
- Stemmed ngram probabilities (1 feature for each 1-, 2-, and 3-gram that includes the word)
- Word vector (fasttext) similarity with context
- Boolean: Whether this candidate is the original
- Boolean: Whether this candidate is changed by step x

The best performance for the ranker only used the ngram probabilities (first item of the feature list), but this did not perform as well as using SUMLM, so I kept SUMLM as the final ranker.

6.3 Results

6.3.1 Evaluation Data

I used the test data sets described in Chapter 5 to evaluate the system and its subcomponents. All types of errors were present in the Wikipedia, Tatoeba, and Labeled Twitter (Koksal et al., 2020) test sets. The Twitter2016 set (Torunoğlu-Selamet et al., 2016) only contains noisy-channel mistakes. Note: the "noisy channel" error type is shorthand for the type of errors I assume can be modeled by a noisy-channel model. The criteria for identifying this type of mistake is that the Damerau-Levenshtein distance between the edited and corrected pair should be no more than 3, and it should not be classifiable as any of the other simple types of mistakes (word merging, word splitting, deasciification, capitalization). Table 6.2 summarizes the frequency of error types of each test corpus.

In general, I used the entire Tatoeba or Twitter datasets, possibly filtered by error type, for evaluation. Since the Wikipedia data set is so large, I used a sample of 1000

	Wikipedia	Tatoeba	Labeled Twitter	Twitter2016
Total Correction Pairs	1000	1170	3782	1016
Word Merging	49 (4.9%)	179 (15.3%)	41 (1.1%)	0 (0%)
Word Splitting	84 (8.4%)	314 (26.8%)	360 (9.5%)	0 (0%)
Deasciification	163 (16.3%)	84 (7.2%)	2582 (68.3%)	0 (0%)
Capitalization	388 (38.8%)	168 (14.4%)	471 (12.4%)	0 (0%)
Noisy Channel	283 (28.3%)	403 (34.4%)	262 (6.9%)	1016 (100%)
Other	33 (3.3%)	22 (1.8%)	66 (1.7%)	0 (0%)

Table 6.2: Distribution of error types for each test dataset

error/correction pairs for each evaluation (1000 unfiltered pairs for end-to-end evaluation, and 1000 pairs of appropriate error types for evaluating subcomponents).

6.3.2 Evaluation Metrics

The ideal behavior of a spelling-correction system involves both accurately detecting and correcting mistakes. Thus, for analyzing overall performance, it may seem reasonable to use traditional metrics such as precision and recall. However, this approach is disadvantageous when comparing performance across different corpora: precision and recall will naturally be affected by the corpora's error rate (which is highly variable among different media and authors). For this study, where the focus is more about choosing the best alternative instead of accurately detecting errors, I found other metrics to be more informative.

When evaluating systems and their subcomponents, I use two main metrics: accuracy for error correction, and false positive correction rate (FPCR).

$$Accuracy = \frac{\|\text{Mistakes Successfully Corrected}\|}{\|\text{Mistakes}\|}$$

$$FPCR = \frac{\|\text{Originally correct words detected as misspelled}\|}{\|\text{Words that were originally correct}\|}$$

The FPCR was evaluated on (presumably) mistake-free sentences from the Wikipedia and Tatoeba corpora. 1000 random sentences from each corpora were extracted. Data from

Twitter was not included since raw Twitter data generally has a higher error rate.

6.3.3 Cascaded Model Subcomponents

I evaluated the performance of each submodule separately, where each module was fed data that it was designed to correct on its own. Data points that involved multiple mistake types were not included.

Word Merging and Word Splitting

The word-merging and word-splitting modules are identical after initial candidate generation: the final output of both modules is the candidate (or original form) that fits the language model best. Table 6.3 shows various language models applied in the merge step, and Table 6.4 shows the same models in the split step. The SUMLM language model generally performed better than the other language models for choosing the best corrections, except for the Tatoeba test set in the word-splitting step, where the unigram model gave the highest accuracy. Using TDK data had an adverse effect on the Wikipedia and Labeled Twitter test sets, but a positive effect on the Tatoeba test set, which is likely due to Tatoeba being a medium that deliberately uses official language. The false positive correction rate was fairly consistent across all language models.

Deasciification Step

Table 6.5 shows the results of running various deasciification models against the different test sets. Surprisingly, the unigram model was most effective at error correction in the Wikipedia and Tatoeba test sets. Adali and Eryiğit (2014), which was specifically designed for Twitter and other social media data, performed best on the Twitter test set. The decision list model performed on par with Adali and Eryiğit (2014) for the Wikipedia and Tatoeba data sets, and on par with the unigram model for the Twitter data set.

The false positive correction rate varied considerably between the models, with the deci-

	Unigram	Stupid Backoff	SUMLM	SUMLM, no TDK data
Accuracy: Wikipedia Test Set	0.882 (882/1000)	0.879 (879/1000)	0.886 (886/1000)	0.914 (914/1000)
Accuracy: Tatoeba Test Set	0.922 (165/179)	0.838 (150/179)	0.860 (154/179)	0.782 (140/179)
Accuracy: Labeled Twitter Test Set	0.610 (25/41)	0.610 (25/41)	0.610 (25/41)	0.610 (25/41)
False Positive Rate: Clean Wikipedia Sentences	0.006 (6/1000)	0.006 (6/1000)	0.005 (5/1000)	0.005 (5/1000)
False Positive Rate: Clean Tatoeba Sentences	0.005 (5/1000)	0.004 (4/1000)	0.003 (3/1000)	0.003 (3/1000)

Table 6.3: Experiments with word-fusing step

	Unigram	Stupid Backoff	SUMLM	SUMLM, no TDK data
Accuracy: Wikipedia Test Set	0.772 (772/1000)	0.787 (787/1000)	0.792 (792/1000)	0.804 (804/1000)
Accuracy: Tatoeba Test Set	0.822 (258/314)	0.831 (261/314)	0.831 (261/314)	0.828 (260/314)
Accuracy: Labeled Twitter Test Set	0.539 (194/360)	0.544 (196/360)	0.544 (196/360)	0.617 (222/360)
False Positive Rate: Clean Wikipedia Sentences	0.006 (6/1000)	0.006 (6/1000)	0.007 (7/1000)	0.007 (7/1000)
False Positive Rate: Clean Tatoeba Sentences	0.023 (23/1000)	0.036 (36/1000)	0.029 (29/1000)	0.032 (32/1000)

Table 6.4: Experiments with word-splitting step

	Unigram	Adali and Eryiğit (2014)	Decision List
Accuracy: Wikipedia Test Set	0.937 (937/1000)	0.905 (905/1000)	0.905 (905/1000)
Accuracy: Tatoeba Test Set	0.869 (73/84)	0.631 (53/84)	0.679 (57/84)
Accuracy: Labeled Twitter Test Set	0.925 (2388/2582)	0.966 (2494/2582)	0.916 (2365/2582)
False Positive Rate: Clean Wikipedia Sentences	0.005 (5/1000)	0.048 (48/1000)	0.001 (1/1000)
False Positive Rate: Clean Tatoeba Sentences	0.014 (14/1000)	0.024 (24/1000)	0.009 (9/1000)

Table 6.5: Experiments with deasciification

sion list performing the best and Adali and Eryiğit (2014) performing the worst, perhaps as an artifact of being trained on social media data, which has a high asciification rate.

Capitalization Step

Table 6.6 shows the performance of various ngram language models on capitalization correction data. All three models perform similarly on the Wikipedia and Twitter data sets, but the unigram model performs significantly better on the Tatoeba data set. The models all performed the best on the Wikipedia data set, which makes sense given that the language model corpus was built from Wikipedia data. Thus, this step of the process is quite sensitive to the data source.

I also considered following Eryiğit and Torunoğlu-Selamet (2017) by using the NER system developed by Şeker and Eryiğit (2012, 2017), but I found that running this system via the API was very slow (about 20 seconds per sentence, since multiple calls were required for morphological analysis and morphological disambiguation as preprocessing). Although the entities recognized seemed to have high precision, there were issues with coverage. The system does not support some entities that are typically capitalized, such as language names, and the system failed to recognize some people/places with obscure names. Again, this shows

	Unigram	Stupid Backoff	SUMLM
Accuracy: Wikipedia Test Set	0.710 (710/1000)	0.724 (724/1000)	0.732 (732/1000)
Accuracy: Tatoeba Test Set	0.631 (106/168)	0.321 (54/168)	0.339 (57/168)
Accuracy: Labeled Twitter Test Set	0.643 (303/471)	0.648 (305/471)	0.648 (305/471)
False Positive Rate: Clean Wikipedia Sentences	0.026 (26/1000)	0.020 (20/1000)	0.016 (16/1000)
False Positive Rate: Clean Tatoeba Sentences	0.148 (148/1000)	0.147 (147/1000)	0.131 (131/1000)

Table 6.6: Experiments with capitalization correction

that this step of the process is very sensitive to the data set in terms of what proper nouns can be recognized.

Noisy Channel Model

Table 6.7 summarizes the performance of the noisy channel spelling correction model on the various datasets, and compares it to existing studies (Büyük, 2020; Torunoğlu-Selamet et al., 2016). The model in this study performs similar to Torunoğlu-Selamet et al. (2016) on the data set introduced by the same study (Twitter2016), but it performs significantly better on all of the other data sets, both in correction accuracy and false positive correction rate. The results of Büyük (2020) are only available for the data set developed by Torunoğlu-Selamet et al. (2016), but the data was augmented by the most common context in the source Twitter corpus. Noting that choosing a random context from the source corpus would give results more in line with the actual data distribution, the gap between Büyük (2020) and Torunoğlu-Selamet et al. (2016) on the Twitter data set is likely overestimated (Torunoğlu-Selamet et al. (2016) used only a unigram language model, ignoring the context).

The noisy-channel model I used was quite similar to Torunoğlu-Selamet et al. (2016), with the main differences being that I used a higher-order ngram language model and an error

	Büyük (2020)	Torunoğlu-Selamet et al. (2016)	This Study: Cascaded Model
Accuracy: Twitter2016 Most common context	0.828 (841/1016)	0.807 (820/1016)	0.809 (822/1016)
Accuracy: Twitter2016 Random context		0.807 (820/1016)	0.788 (801/1016)
Accuracy: Wikipedia Test Set		0.496 (496/1000)	0.695 (695/1000)
Accuracy: Tatoeba Test Set		0.171 (69/403)	0.439 (177/403)
Accuracy: Labeled Twitter Test Set		0.279 (73/262)	0.500 (131/262)
False Positive Rate: Clean Wikipedia Sentences		0.123 (123/1000)	0.022 (22/1000)
False Positive Rate: Clean Tatoeba Sentences		0.065 (65/1000)	0.036 (36/1000)

Table 6.7: Experiments with the noisy channel model

model trained on the Wikipedia test set I developed. The error model of Torunoğlu-Selamet et al. (2016) was constructed from the same source corpus as the test set Twitter2016, where error/correction pairs were generated by feeding OOV words to existing spelling correctors, and accepting the corrections when multiple spelling correctors agreed on the correction. The advantage of the Wikipedia training set is that it doesn't rely on outside spelling correctors, which may have biases of their own.

6.3.4 System Performance

Table 6.8 presents the end-to-end system performance of all errors in each test set, compared to the performance of Eryiğit and Torunoğlu-Selamet (2017) on the same data sets. The cascaded model developed in this study outperformed Eryiğit and Torunoğlu-Selamet (2017) on all data sets in terms of accuracy of corrections. However, Eryiğit and Torunoğlu-Selamet

	Eryiğit and Torunoğlu-Selamet (2017)	This Study: Cascaded Model
Accuracy: Wikipedia Test Set	0.230 (230/1000)	0.713 (713/1000)
Accuracy: Tatoeba Test Set	0.168 (197/1170)	0.549 (642/1170)
Accuracy: Labeled Twitter Test Set	0.568 (2147/3782)	0.759 (2639/3782)
False Positive Rate: Clean Wikipedia Sentences	0.144 (144/1000)	0.039 (39/1000)
False Positive Rate: Clean Tatoeba Sentences	0.025 (25/1000)	0.166 (166/1000)

Table 6.8: End-To-End System Performance

(2017) had a lower false positive correction rate on the Tatoeba data set. The two systems performed most comparably on the labeled Twitter test set, which is likely due to the system of Eryiğit and Torunoğlu-Selamet (2017) being specifically developed for Twitter.

Table 6.9 shows a breakdown of performance by error type of each test data set on the cascaded model of this study. Most of the error types show the best performance on the Wikipedia corpus, which makes sense given that all language models and training data were developed from Wikipedia text. In general, the accuracy of each error type in the end-to-end system is lower than in its dedicated module, and this seems to affect certain error types in certain corpora disproportionately: especially capitalization and deasciification in Tatoeba. Thus, the complexity of having a cascaded approach is corpus-sensitive in terms of error propagation.

6.4 Conclusion

In this chapter, I presented a baseline spelling correction system that takes into account various error types based on the analysis in Chapter 4. In addition to informing the design of this system, the error corpus was used to build an error model with better cross-domain

	Wikipedia	Tatoeba	Labeled Twitter
Deasciification	0.871 (142/163)	0.417 (35/84)	0.774 (1998/2582)
Capitalization	0.696 (270/388)	0.339 (57/168)	0.628 (296/471)
Word Splitting	0.643 (54/84)	0.755 (237/314)	0.433 (156/360)
Word Merging	0.898 (44/49)	0.760 (136/179)	0.659 (27/41)
Noisy Channel	0.682 (193/283)	0.422 (170/403)	0.519 (136/262)
Word Splitting + Noisy Channel	0.200 (2/10)	0.143 (1/7)	0.314 (11/35)
Word Merging + Noisy Channel	0.583 (7/12)	0.500 (6/12)	0.478 (11/23)
Other	0.091 (1/11)	0 (0/3)	0.500 (4/8)

Table 6.9: End-To-End System Performance By Error Type

performance than existing techniques. This system differs from previous studies about Turkish in that it treats spelling correction in formal texts as a normalization problem. Although similar normalization systems exist for informal social media text, they do not perform as well on out-of-domain corpora. The developed system performs the strongest on the data set it was designed with (Wikipedia), but it also shows cross-domain adaptability.

Chapter 7

MACHINE TRANSLATION MODELS FOR SPELLING CORRECTION

As an alternative to a cascaded or noisy channel approach, several studies propose applying machine translation models to the spelling correction problem. Input tokens may either be individual characters or words (using word-level tokens is the typical setup for machine translation). An advantage of this approach is that it can use existing technology and libraries to create an end-to-end system and avoid excessive custom logic. However, a machine translation model typically requires a lot of training data as input, which is especially challenging for low-resource languages.

7.1 Previous Work

Formiga and Fonollosa (2012) was an early study to suggest using character-based statistical machine translation (SMT) for a normalization task: normalizing noisy input for machine translation. Hasan et al. (2015) proposed a character-based SMT model for spelling correction of search queries in English, via a phrase-based translation model using Moses[1]. For spelling correction and normalization in low resource languages, multiple studies explore using neural seq2seq models (Etoori et al., 2018; Göker and Can, 2018; Büyük, 2020), which are also widely used in neural machine translation (NMT). Çolakoğlu et al. (2019) compared using an SMT model to using an NMT model for social media normalization, and in this study the SMT system performed better. Both NMT and SMT systems typically require large amounts of training data, so a major problem each of these studies confront is the lack of such real-world data. Etoori et al. (2018), Çolakoğlu et al. (2019), and Büyük (2020) mit-

[1] `http://www.statmt.org/moses/`

igate this problem by artificially creating spelling errors, while Göker and Can (2018) uses a small set of real-world data (300 manually normalized tweets for training, 300 for testing).

Using real data instead of synthetic has been shown to increase performance for similar tasks. Boyd (2018) demonstrated that using grammar errors extracted from Wikipedia increased existing grammar error correction encoder-decoder system performance by over 5%.

Despite the lack of real-world data, Büyük (2020) was able to outperform Torunoğlu-Selamet et al. (2016) on the Twitter test set presented by the latter, showing promise of this approach. One other major difference is that Büyük (2020) used context information: a "consonant skeleton" of the two surrounding context words (essentially the stems of the previous word and the next word), while Torunoğlu-Selamet et al. (2016) only used a unigram language model.

7.2 Approach

7.2.1 Character-Level Noisy Channel Model

I experimented with using both NMT and SMT systems as a noisy channel model. For both systems, the input format was based on that of Büyük (2020): using the consonant skeleton of the two neighbor context words. The consonant skeleton is defined as the entire word if the length is at most 3, or the first three consonants of the word if the word is longer. Filler characters are used if the skeleton is less than length three, and special characters are used for word breaks. For example to the input "öğrenciler kitapp okuyor", where we want to perform spelling correction on "kitapp", is encoded as "ğrn|kitapp|kyr". "Bu kitapp oku" is encoded as "Bu*|kitapp|oku". For the NMT system, characters were treated as the tokens. For the SMT system, the context consonant skeletons and the characters of the target word were the input tokens (the input for "öğrenciler kitapp okuyor" would be "ğrn k i t a p p kyr").

I built a seq2seq model using the architecture described in Büyük (2020) using the same

model parameters (number of LSTM layers, number of hidden and latent dimensions). I used keras[2] for implementation. Unfortunately I was unable to get comparable performance using my implemented NMT model, due to lack of computing resources. Büyük (2020) used 1 billion training examples, which was orders of magnitude larger than I was able to process.

I implemented the SMT model with Moses, using a fairly standard setup and collection of tools, following some configuration from Çolakoğlu et al. (2019). I used MGIZA with grow-diag-final-and for alignment and KenLM for a language model. The corpus for my language model was a full version of Wikipedia, with all plaintext before the References ("Kaynakça") section. The input to KenLM was each word in the corpus with their surrounding consonant skeletons. The order of the model was 6 (up to 6-grams), and I pruned 6-grams that only occurred once (i.e. I ran: lmplz -o 6 –prune 0 0 0 0 0 1)

Comparing Training Data Types

Now that a corpus of spelling corrections is available, a natural question is: how does using real-world data impact model performance?

In addition to using the Wikipedia data as input, another possibility is to combine the Wikipedia spelling correction set with artificially generated errors. Cahill et al. (2013) investigated using the preposition error correction data along with artificial data generated from confusion distributions they saw in their extracted data set, and found that training on artificial data performed comparably to real data in their experiments.

In order to evaluate which kind of training data is most beneficial, I experiment with using the following training sets, and report results on each test data set (see Chapter 5). For this set of experiments, we assume a noisy channel model similar to Büyük (2020) and Torunoğlu-Selamet et al. (2016).

- Pure synthetic training data, as used by Büyük (2020) (Total training pairs: 11,799,007)
- Wikipedia spelling corrections only (filtered to exclude non-noisy channel errors) (Total

[2] `https://keras.io/`

training pairs: 169,266)

- Synthetic data generated using the error model only (which is based on Brill and Moore (2000)) (Total training pairs: 11,051,494)
- Wikipedia spelling corrections + synthetic data generated using the error model (Total training pairs: 11,220,760)

My generation of the pure synthetic data was a little bit simpler than that of Büyük (2020), in that I didn't take into account the keyboard layout for adjusting the probabilities of character substitution/insertion. Instead I used a uniform distribution over all lowercase and uppercase Turkish letters.

When generating synthetic data according to the Brill and Moore (2000) error model, I directly used the estimated probabilities of the model, so consequently, most of the data points had only one mistake according to the model, which may be different from what is represented in the real data, due to the data not fitting the noisy channel model assumptions perfectly.

7.2.2 Word-Level SMT Model

While Etoori et al. (2018) and Büyük (2020) approach a classical noisy-channel spelling correction problem, Göker and Can (2018) and Çolakoğlu et al. (2019), solve the problem of social media normalization, which is much broader, more comparable to dealing with my entire dataset at once. I performed a small experiment to investigate which part of the problem an SMT system is most adept to solve.

Noticing that the number of instances of context-free word/correction pairs is smaller (more likely to repeat) for non noisy-channel error types (capitalization, deasciification, word merging, and word splitting), I implemented a word-based SMT model for correcting only these types of errors. For this model, the same tools/setup was used as in the Character-Level SMT experiments, except the language model was the Wikipedia corpus split on sentences, and reordering was disallowed (distortion-limit was set to 0 in Moses). For this experiment, only the Wikipedia spelling corpus, prefiltered to exclude noisy-channel errors, was used as

training data (no synthetic data).

7.3 Results

7.3.1 Noisy-Channel SMT

Table 7.1 summarizes performances of SMT models trained on the various data sets described in Section 7.2.1. The model trained on synthetic data from the Brill and Moore (2000) model performs the best across all datasets, even when this same dataset is combined with the raw error/correction pairs. I suspect this is because the process for generating the synthetic errors is not perfectly representative of how the original errors were formed. After being trained on an idealized version of the data, the SMT's error model is more similar to the Brill and Moore (2000) model than the synthetic+real data-trained SMT's version is. The model trained from raw Wikipedia data had consistently lower accuracy than the synthetic data it was used to generate, showing that the data size was not big enough to train the SMT model optimally. The pure synthetic data performed worse than the data generated using the better error model, proving the importance of building an accurate error model in this situation.

Table 7.2 compares how the best SMT model performs against the model introduced in Chapter 6 and the model of Torunoğlu-Selamet et al. (2016). The SMT model performs worse than my baseline noisy channel model on the Twtter2016 data sets, which have up to 1 edit distance that may not reflect the actual distribution of mistakes. The SMT model has a higher error-correction accuracy on the other data sets which I believe to have mistakes that are more representative of the actual distribution in their respective sources. However, the false positive correction rate is higher for the Tatoeba set.

Table 7.3 shows how the system performance is affected when the SMT model replaces the noisy channel model in the cascaded architecture. The end-to-end model with SMT performs slightly worse for all of the test datasets, due to the model erroneously changing corrections made by other modules. In future research, it would be interesting to investigate

	Pure Synthetic Data	Pure Wikipedia Data	Synthetic Data from Error Model	Wikipedia Data + Synthetic Data from Error Model
Accuracy: Twitter2016 Most common context	0.486 (494/1016)	0.367 (373/1016)	0.763 (775/1016)	0.756 (768/1016)
Accuracy: Twitter2016 Random context	0.473 (481/1016)	0.353 (359/1016)	0.692 (703/1016)	0.676 (687/1016)
Accuracy: Wikipedia Test Set	0.275 (275/1000)	0.259 (259/1000)	0.750 (750/1000)	0.741 (741/1000)
Accuracy: Tatoeba Test Set	0.181 (73/403)	0.489 (197/403)	0.541 (218/403)	0.531 (214/403)
Accuracy: Labeled Twitter Test Set	0.202 (53/262)	0.244 (64/262)	0.595 (156/262)	0.569 (149/262)

Table 7.1: SMT noisy channel model experiments with different training data

	Torunoğlu-Selamet et al. (2016)	This Study: Cascaded Model (Chapter 6)	This Study: SMT Noisy Channel (Error Model Synthetic Data)
Accuracy: Twitter2016 Most common context	0.807 (820/1016)	0.809 (822/1016)	0.763 (775/1016)
Accuracy: Twitter2016 Random context	0.807 (820/1016)	0.788 (801/1016)	0.692 (703/1016)
Accuracy: Wikipedia Test Set	0.496 (496/1000)	0.695 (695/1000)	0.750 (750/1000)
Accuracy: Tatoeba Test Set	0.171 (69/403)	0.439 (177/403)	0.541 (218/403)
Accuracy: Labeled Twitter Test Set	0.279 (73/262)	0.500 (131/262)	0.595 (156/262)
False Positive Rate: Clean Wikipedia Sentences	0.123 (123/1000)	0.022 (22/1000)	0.024 (24/1000)
False Positive Rate: Clean Tatoeba Sentences	0.065 (65/1000)	0.036 (36/1000)	0.066 (66/1000)

Table 7.2: SMT noisy channel model compared with Chapter 6 results

	Eryiğit and Torunoğlu-Selamet (2017)	Chapter 6: Cascaded Model	Chapter 7: Cascaded Model + SMT noisy channel
Accuracy: Wikipedia Test Set	0.230 (230/1000)	0.713 (713/1000)	0.694 (694/1000)
Accuracy: Tatoeba Test Set	0.168 (197/1170)	0.549 (642/1170)	0.515 (603/1170)
Accuracy: Labeled Twitter Test Set	0.568 (2147/3782)	0.759 (2639/3782)	0.684 (2586/3782)
False Positive Rate: Clean Wikipedia Sentences	0.144 (144/1000)	0.039 (39/1000)	0.024 (24/1000)
False Positive Rate: Clean Tatoeba Sentences	0.025 (25/1000)	0.166 (166/1000)	0.138 (138/1000)

Table 7.3: Overall System Performance, with SMT integrated as noisy channel model

if there are better ways to integrate the SMT model into the cascaded model.

7.3.2 Word-Level SMT

As shown in Table 7.4, the performance of the word-level SMT model did not approach that of the cascaded model. The modules in the cascaded model exploit the simplistic properties of these types of errors: the general form of these common errors is known and there are few candidates for each word to enumerate. The propagation between these modules is also relatively low since each model as well as the final ranker relies on a similar language model, thus creating a de facto unified subsystem.

7.4 Conclusion

Although various machine translation models have performed well for normalization models in other studies, it does not look to be a promising end-to-end system for Turkish spelling correction as defined in this study. For spelling mistakes that follow a noisy-channel model, SMT is a viable alternative to directly apply the model of Brill and Moore (2000), as the SMT system is just another type of noisy-channel model. Introducing the SMT model into

	Eryiğit and Torunoğlu-Selamet (2017)	Chapter 6: Cascaded Model	SMT Word Model
Deasciification	0.650 (106/163)	0.871 (142/163)	0.748 (122/163)
Capitalization	0.090 (35/388)	0.696 (270/388)	0.407 (158/388)
Word Splitting	0.202 (17/84)	0.643 (54/84)	0.369 (31/84)
Word Merging	0	0.898 (44/49)	0.245 (12/49)
Noisy Channel	0.223 (63/283)	0.682 (193/283)	-
Word Splitting + Noisy Channel	0	0.200 (2/10)	-
Word Merging + Noisy Channel	0	0.583 (7/12)	-
Other	0	0.091 (1/11)	-

Table 7.4: Accuracy by type of error on Wikipedia test set

the cascaded architecture has a slightly negative effect because it introduces more error propagation, and future research is needed to determine how to optimally integrate the different modules.

Chapter 8

CONCLUSION

In this thesis, I presented a novel large data set for Turkish spelling correction and evaluated how this new data can improve existing spelling correction techniques. I showed how Wikipedia edit history can be used to extract a large number of spelling errors and corrections, which can be leveraged to build a robust error model.

From the error correction corpus, we are also able to get a better sense of typical mistakes people make when writing Turkish, especially for the formal writing domain. In addition to character operations modellable by a character-level noisy channel model, changing in spacing, capitalization, and diacritics are also important error types to consider in Turkish, which can be dealt with more effectively on the word-level.

Because Turkish spelling errors fit well into this categorization of error types, an effective model for spelling correction is a cascaded system, where each module handles a specific error-type. The error correction corpus is important for training the error model of the noisy channel component of this system. An SMT model can also be effectively trained using the error correction corpus, though integration into the cascaded model has mixed results.

8.1 Contributions

The results of this thesis can be summarized as four main contributions:

- An efficient method for extracting small edits from Wikipedia (Chapter 2), which is freely available[1] and applicable to many languages
- A large corpus[2] of Turkish spelling corrections including the surrounding context (pri-

[1] `https://github.com/tguinard/WikiSmallEdits`

[2] `https://github.com/tguinard/TurkishWikiSpellingMistakes`

marily developed in Chapters 2-3), also including error-type annotation (Chapter 4)

- An analysis of types of spelling mistakes found in Turkish. I present both my considerations for classifying an edit as a spelling correction (Chapter 3), as well as a descriptive analysis of the spelling errors in the resulting corpus (Chapter 4)
- Initial spelling correction systems for Turkish (Chapters 6-7). A cascaded model (Chapter 6) is a promising architecture, and incorporating SMT as a noisy channel model is a viable option (Chapter 7). Both of these systems use a subset of the spelling correction corpus (the "noisy channel" error type) in order to train an error model according to Brill and Moore (2000)

8.2 Future Work

In the future, similar techniques may be applied to build corpora for the spell correcting problem in other languages. A similar baseline architecture may also be effective for typical mistakes in other language, as the types of changes seen are not particular to Turkish. For example, many European languages also suffer from asciification.

Additionally, I hope that future studies can improve spelling correction models beyond the baseline cascaded system presented in this study. The error correction corpus developed in this thesis can be used by future studies for training and evaluating such systems.

BIBLIOGRAPHY

Kübra Adali and Gülşen Eryiğit. Vowel and Diacritic Restoration for Social Media Texts. In *Proceedings of the 5th Workshop on Language Analysis for Social Media (LASM)*, pages 53–61, 2014.

B Thomas Adler, Luca De Alfaro, Santiago M Mola-Velasco, Paolo Rosso, and Andrew G West. Wikipedia Vandalism Detection: Combining Natural Language, Metadata, and Reputation Features. In *International Conference on Intelligent Text Processing and Computational Linguistics*, pages 277–288. Springer, 2011.

Shane Bergsma, Dekang Lin, and Randy Goebel. Web-Scale N-gram Models for Lexical Disambiguation. In *Proceedings of the 21st international jont conference on Artifical intelligence*, pages 1507–1512, 2009.

Adriane Boyd. Using Wikipedia Edits in Low Resource Grammatical Error Correction. In *Proceedings of the 2018 EMNLP Workshop W-NUT: The 4th Workshop on Noisy User-generated Text*, pages 79–84, 2018.

Eric Brill and Robert C Moore. An Improved Error Model for Noisy Channel Spelling Correction. In *Proceedings of the 38th annual meeting of the association for computational linguistics*, pages 286–293, 2000.

Christopher Bryant, Mariano Felice, and Edward Briscoe. Automatic Annotation and Evaluation of Error Types for Grammatical Error Correction. Association for Computational Linguistics, 2017.

Osman Büyük. Context-Dependent Sequence-to-Sequence Turkish Spelling Correction. *ACM*

Transactions on Asian and Low-Resource Language Information Processing (TALLIP), 19 (4):1–16, 2020.

Aoife Cahill, Nitin Madnani, Joel Tetreault, and Diane Napolitano. Robust Systems for Preposition Error Correction Using Wikipedia Revisions. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 507–517, 2013.

Gökhan Çelikkaya, Dilara Torunoğlu, and Gülsen Eryiğit. Named Entity Recognition on Real Data: A Preliminary Investigation for Turkish. In *2013 7th International Conference on Application of Information and Communication Technologies*, pages 1–5. IEEE, 2013.

Talha Çolakoğlu, Umut Sulubacak, Ahmet Cüneyd Tantuğ, et al. Normalizing Non-canonical Turkish Texts Using Machine Translation Approaches. In *The 57th Annual Meeting of the Association for Computational Linguistics Proceedings of the Student Research Workshop*. The Association for Computational Linguistics, 2019.

Cagri Cöltekin. A Freely Available Morphological Analyzer for Turkish. In *LREC*, volume 2, pages 19–28, 2010.

Çagri Çöltekin. A set of open source tools for Turkish natural language processing. In *LREC*, pages 1079–1086, 2014.

Silviu Cucerzan and Eric Brill. Spelling correction as an iterative process that exploits the collective knowledge of web users. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*, pages 293–300, 2004.

Fred J Damerau. A technique for computer detection and correction of spelling errors. *Communications of the ACM*, 7(3):171–176, 1964.

Johannes Daxenberger and Iryna Gurevych. Automatically Classifying Edit Categories in Wikipedia Revisions. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 578–589, 2013.

Gülşen Eryiğit and Dilara Torunoğlu-Selamet. Social media text normalization for Turkish. *Natural Language Engineering*, 23(6):835, 2017.

Pravallika Etoori, Manoj Chinnakotla, and Radhika Mamidi. Automatic Spelling Correction for Resource-Scarce Languages using Deep Learning. In *Proceedings of ACL 2018, Student Research Workshop*, pages 146–152, 2018.

Oliver Ferschke, Torsten Zesch, and Iryna Gurevych. Wikipedia Revision Toolkit: Efficiently Accessing Wikipedia's Edit History. In *Proceedings of the ACL-HLT 2011 System Demonstrations*, pages 97–102, 2011.

Pieter Fivez, Simon Suster, and Walter Daelemans. Unsupervised Context-Sensitive Spelling Correction of Clinical Free-Text with Word and Character N-Gram Embeddings. In *BioNLP 2017*, pages 143–148, 2017.

Lluis Formiga and José AR Fonollosa. Dealing with Input Noise in Statistical Machine Translation. In *Proceedings of COLING 2012: Posters*, pages 319–328, 2012.

Sinan Göker and Burcu Can. Neural Text Normalization for Turkish Social Media. In *2018 3rd International Conference on Computer Science and Engineering (UBMK)*, pages 161–166. IEEE, 2018.

Andrew R Golding and Dan Roth. A Winnow-Based Approach to Context-Sensitive Spelling Correction. *Machine learning*, 34(1):107–130, 1999.

Roman Grundkiewicz. Automatic Extraction of Polish Language Errors from Text Edition History. In *International Conference on Text, Speech and Dialogue*, pages 129–136. Springer, 2013.

Roman Grundkiewicz and Marcin Junczys-Dowmunt. The WikEd Error Corpus: A Corpus of Corrective Wikipedia Edits and Its Application to Grammatical Error Correction. In *International Conference on Natural Language Processing*, pages 478–490. Springer, 2014.

Saša Hasan, Carmen Heger, and Saab Mansour. Spelling Correction of User Search Queries through Statistical Machine Translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 451–460, 2015.

Sara Javanmardi, David W McDonald, and Cristina V Lopes. Vandalism Detection in Wikipedia: A High-Performing, Feature–Rich Model and its Reduction Through Lasso. In *Proceedings of the 7th International Symposium on Wikis and Open Collaboration*, pages 82–90, 2011.

Mark D Kemighan, Kenneth Church, and William A Gale. A spelling correction program based on a noisy channel model. In *COLING 1990 Volume 2: Papers presented to the 13th International Conference on Computational Linguistics*, 1990.

Asiye Tuba Koksal, Ozge Bozal, Emre Yürekli, and Gizem Gezici. # Turki $ hTweets: A Benchmark Dataset for Turkish Text Correction. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings*, pages 4190–4198, 2020.

Vladimir I Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710. Soviet Union, 1966.

Aurélien Max and Guillaume Wisniewski. Mining Naturally-occurring Corrections and Paraphrases from Wikipedia's Revision History. In *LREC*, 2010.

Eric Mays, Fred J Damerau, and Robert L Mercer. Context based spelling correction. *Information Processing & Management*, 27(5):517–522, 1991.

Marcin Miłkowski. Automated building of error corpora of Polish. *Corpus Linguistics, Computer Tools, and Applications–State of the Art. PALC*, pages 631–639, 2007.

Rani Nelken and Elif Yamangil. Mining Wikipedia's Article Revision History for Training Computational Linguistics Algorithms. In *Proceedings of the AAAI Workshop on Wikipedia and Artificial Intelligence: An Evolving Synergy*, pages 31–36, 2008.

Kemal Oflazer. Error-tolerant Finite State Recognition with Applications to Morphological Analysis and Spelling Correction. *Computational Linguistics*, 22(1):73–89, 1996.

Kezban Dilek Onal and Pinar Karagoz. Named Entity Recognition from Scratch on Social Media. In *Proceedings of 6th International Workshop on Mining Ubiquitous and Social Environments (MUSE), co-located with the ECML PKDD*, volume 104, 2015.

Zeynep Ozer, Ilyas Ozer, and Oguz Findik. Diacritic restoration of Turkish tweets with word2vec. *Engineering Science and Technology, an International Journal*, 21(6):1120–1127, 2018.

Tommi Pirinen and Krister Lindén. Finite-State Spell-Checking with Weighted Language and Error Models: Building and Evaluating Spell-Checkers with Wikipedia as Corpus. In *Proceedings of LREC 2010 Workshop on creation and use of basic lexical resources for less-resourced languages*, 2010.

Martin Potthast. Crowdsourcing a Wikipedia Vandalism Corpus. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, pages 789–790, 2010.

Haşim Sak, Tunga Güngör, and Murat Saraçlar. Resources for Turkish morphological processing. *Language resources and evaluation*, 45(2):249–261, 2011.

Johannes Schaback and Fang Li. Multi-Level Feature Extraction for Spelling Correction. In *IJCAI-2007 Workshop on Analytics for Noisy Unstructured Text Data*, pages 79–86. Citeseer, 2007.

Gökhan Akın Şeker and Gülşen Eryiğit. Initial explorations on using CRFs for Turkish named entity recognition. In *Proceedings of COLING 2012*, pages 2459–2474, 2012.

Gökhan Akın Şeker and Gülşen Eryiğit. Extending a CRF-based named entity recognition model for Turkish well formed text and user generated content. *Semantic Web Journal*, (doi:10.3233/SW-170253), 2017.

Xu Sun, Jianfeng Gao, Daniel Micol, and Chris Quirk. Learning Phrase-Based Spelling Error Models from Clickthrough Data. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 266–274, 2010.

Dilara Torunoğlu-Selamet and Gülşen Eryiğit. A Cascaded Approach for Social Media Text Normalization of Turkish. In *Proceedings of the 5th Workshop on Language Analysis for Social Media (LASM)*, pages 62–70, 2014.

Dilara Torunoğlu-Selamet, Eren Bekar, Tugay Ilbay, and Gülsen Eryigit. Exploring Spelling Correction Approaches for Turkish. In *Proceedings of the 1st International Conference on Turkic Computational Linguistics at CICLING, Konya*, pages 7–11, 2016.

Ziqi Wang, Gu Xu, Hang Li, and Ming Zhang. A Fast and Accurate Method for Approximate String Search. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 52–61, 2011.

Casey Whitelaw, Ben Hutchinson, Grace Chung, and Ged Ellis. Using the Web for Language Independent Spellchecking and Autocorrection. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 890–899, 2009.

Mark Yatskar, Bo Pang, Cristian Danescu-Niculescu-Mizil, and Lillian Lee. For the sake of simplicity: Unsupervised extraction of lexical simplifications from Wikipedia. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 365–368, 2010.

Deniz Yuret and Michael De La Maza. The Greedy Prepend Algorithm for Decision List Induction. In *International Symposium on Computer and Information Sciences*, pages 37–46. Springer, 2006.

Torsten Zesch. Measuring Contextual Fitness Using Error Contexts Extracted from the Wikipedia Revision History. In *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*, pages 529–538, 2012.

Torsten Zesch, Christof Müller, and Iryna Gurevych. Extracting Lexical Semantic Knowledge from Wikipedia and Wiktionary. In *LREC*, volume 8, pages 1646–1652, 2008.