T.C.
MARMARA UNIVERSITY
FACULTY OF ENGINEERING
COMPUTER ENGINEERING DEPARTMENT

CSE4197 - Analysis and Design Document

GRAMMAR AND SPELL CHECKING FOR TURKISH LANGUAGE

Group Members

150116035 Furkan Kerem Eyisoy
150115013 Mert Kelkit
150114016 Rümeysa Eliöz

Supervisor

Assoc. Prof. Murat Can Ganiz

January 4, 2020

1 Introduction

Although the amount of usable textual data increases day by day, thanks to microblog websites like Twitter and Facebook, most of the data obtained from these sources is not in canonical form and it makes them hard to use for ML (Machine Learning) purposes. Correcting these ill-formed texts requires excessive labor force. Even though many state of the art grammar and spell checking methods have very successful results, it has not been reached to a satisfying level in many agglutinative languages such as Turkish since most of the methods are rule-based and language-specific. Main aim of this project is to solve this problem by developing a new method.

1.1 Problem Description and Motivation

The usage of social media is still increasing. On average, 500 million tweets are posted daily on Twitter. And due to this high volume, social platforms are becoming a major source of data for ML algorithms in many different fields. Especially in Twitter, there exists different kinds of phrases that are trending and they are changing frequently. Moreover, these trends are usually not grammatically correct. Due to this situation, it is not possible to use these data without pre-processing them. Commonly used old fashioned rule based algorithms developed can not keep up with these frequent changes on social media. There has been done significant number of studies to benefit from the above mentioned volume of data with maximum efficiency by grammar and spelling checking. Due to the difference in natures of languages, it is still not possible to create a method that works for every language and it is possible to say that developing more accurate methods for agglutinative languages is still one of the most challenging tasks in NLP (Natural Language Processing).

Major motivation behind this study is to help popular supervised and unsupervised NLP studies by expanding the amount of usable data. For morphologically rich languages such as Turkish, with which we concern, suffixes can create tremendously long unique words. This causes there to be many unique words in comparison to the corpus size, resulting in a deterioration of the training signal Because of this reason, the following tasks are very challenging if misspellings are present in the data:

- Document classification and clustering
- Named entity recognition
- Part of speech tagging
- Training word embeddings
- Chatbots
- Sentiment analysis
- Text summarization

Our minor motivations are listed below:

- To encourage the correct use of the Turkish language by providing an application/browser add-on.
  It is a fact people do not pay attention to grammatical rules when they are sharing their instant thoughts and moments on social media. The increase in the misuse of any language leads to the depreciation of that language in time. We are going to develop a publicly available application so as to prevent depreciation of the Turkish language by helping people with instant grammar and spell checking.

- To inspire new state of the art studies on grammar and spell checking for the Turkish language.
  To our knowledge there are not many studies that apply grammar and spell checking with up to date deep learning architectures for the Turkish language. Most of the studies have been done in this field are rule based and outdated. By using a complex and open to development semi-supervised sequence2sequence models we are hoping to inspire new studies. The semi-supervised nature of the models may be achieved trough multitask learning, transfer learning or other means.

We believe that in case of successful completion of our project, necessary manpower for text preprocessing will decrease and the number of promising studies in Turkish NLP field will accelerate. Based on this, we believe this study is important and valuable.
We are planning to implement methods proposed in section 4 about Turkish social media grammar and spell checking in order to create a baseline. Afterwards, we will implement our own deep learning architectures for this task.

1.2 Scope

Our primary purpose on this project is to propose a sophisticated method and an interface for grammar and spell checking for Turkish language on social media. To accomplish this purpose, various NLP and ML methods will be applied. There can be found various studies for this problem with supervised, unsupervised and rule-based approaches, but we mostly focus on studies that support our methodology such as, studies where morphological disambiguation and neural text normalization are used. Most of the NLP solutions will benefit from our methodology but some of these NLP algorithms are not in the scope of this project such as document classification, named entity recognition and part of speech tagging. Punctuation errors are in the scope under the domain of grammar checking. Words that are misused in a sentence but correctly spelled are also in the scope e.g. *"Adamın yükü çok ağırdı."* is a meaningful sentence in Turkish however *"Adamın yükü çok sağırdı."* is not, even though *"sağırdı"* is correctly spelled. This project aims to solve this ambiguity. Implementation and comparison of present studies is in our scope, too.

1.2.1 Steps

Our planned steps are listed below:

1. Literature survey on studies about Turkish text normalization.
   Investigation of neural, ML and rule-based approaches.

2. Gathering noisy and clean data.
   Collection of publicly available tweets, dataset that Kemik Group [7] provides as noisy data and Turkish news and Wikipedia dataset as clean data.

3. Implementation of algorithms that generate synthetic data.
   We will implement rule based synthetic example generation algorithm with common misspelling patterns with various edit distances.

4. Implementation of state of the art baselines.
   The methodologies obtained by literature survey on step one will be implemented in order to test their ability to correct spellings recently collected data and to compare with our methodology.

5. Design and implementation of Neural architecture.
   We will experiment with RNN architectures such as encoder-decoder with attention. Design of the specific architecture as well as other hyper parameters will be tuned by hand on the training data.

6. Training and testing phase.
   This step is expected to be the longest and most power consuming step of the project. During the training and testing phases results will be analyzed frequently.

7. Developing an application or browser add-on
   An application that dynamically checks grammar and spellings of input texts and suggests a list of possible corrected forms of misspelled words or phrases. Users may send feedbacks about suggestions so we can train our model with users' contribution.

1.2.2 Constraints

- Any kind of text is sufficient as an input but it is not guaranteed or required to correct meaningless phrases such as random character sequences like *"sdfsdfa"* which stands for the English abbreviation *"lol"* in Turkish social media.
- It is not guaranteed to correct OOV and foreign words or phrases.
- Since our corpus is limited naturally, results for OOV words may not be accurate.
- Our dataset will be composed of tweets, news and Wikipedia dataset.
- Python and Java languages will be used in implementations.
- Training time depends on computer's processing power.
- System should provide an output for a given input less than one second.

1.2.3 Assumptions

- It is assumed that users will enter only non-empty input.
- It is assumed that users will have an internet connection.
- It is assumed that at least 80% words of any input will be Turkish.
- It is assumed that there are sufficient training data for our model to be successful.

1.3 Definitions, Acronyms and Abbreviations

– **Machine Learning (ML):** Methods that enable computers to learn from past data.

– **Natural Language Processing (NLP):** A set of methods that enable machines to understand texts.

– **Recurrent Neural Networks (RNN):** A special type of neural network that is specialized for sequence data.

– **Long-Short Term Memory (LSTM):** A special type of RNN cell that solves vanishing gradient problem caused by long term dependencies of text sequences.

– **Out of Vocabulary (OOV):** Words that are not present in the given vocabulary.

– **Sequence to Sequence (seq2seq):** A special architecture of RNN where both inputs and outputs are sequences.

– **Graphical Processing Unit (GPU):** An external processing unit for graphical calculations which performs very good on parallelized matrix multiplication.

– **Hidden Markov Models (HMM):** State machine that probabilistically formed with past observations.

– **Bilingual Evaluation Understudy (Bleu):** A metric that is used for machine translation tasks.

– **Noisy Data:** Text data is not in the grammatically correct form.

– **Clean (Canonical) Data:** Text data that is in the grammatically correct form.

– **Corpus:** Collection of text datasets for NLP applications.

2 Related Work

There are several studies about grammar and spell checking for Turkish language. Some of these studies provide candidate generation for ill-formed words, some of them just locates the ill-formed words. While relatively old studies have adopted a rule-based approach, there have been recent studies in which neural or machine learning approaches have been applied.

Torunoğlu and Eryiğit [2] proposed a cascaded architecture for Turkish text normalization of social media. Authors defined 7 rule based steps in order to generate the correct form of an ill-formed word. First, the ill-formed word is detected using a morphological analyzer. After detection of ill-formed word, candidate generation steps are applied sequentially. These steps focus on detecting proper nouns, abbreviations, recognizing social media specific keywords (*mentions, hashtags, RT, etc.*), letter repetitions on words (*güzeeel - güzel*), vowel restorations (*slm - selam*) and deasciification (*yakisikli - yakışıklı*). Since all of these steps are rule based, it makes this study language specific and it is unable to handle new misspelling patterns. However, this method can be useful for our project in order to prepare big part of our training data. Besides we can use this algorithm as a baseline. Steps of this study are shown in Figure 1.

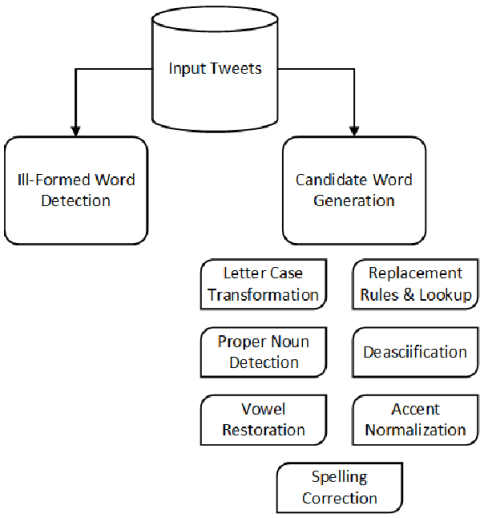

Figure 1: Cascaded architecture for Text Normalization. [2]

Bölücü and Can [3] proposed a method which combines Noisy Channel Model and HMM(Hidden Markov Models). This study considers contextual information of a word while correcting any spelling and grammar mistake. There is a learning model in the proposed method which is a HMM with emission probabilities obtained by Noisy Channel Model and this HMM serves as a language model. Learning model is trained on the BOUN dataset [4] which consists of 3 news and 1 website datasets in the Turkish language and it is assumed that each word or phrase is spelled correctly. After training the language model, Zemberek [5] is used for noisy word detection. If Zemberek finds any noisy word, it generates various candidates for the noisy word

and passes it to HMM. HMM applies Viterbi algorithm in order to find most probable candidate word of given noisy word. This study is also suitable for our training dataset preparation. This method is another possible baseline of our project. Figure 2 shows an example use case of Viterbi algorithm.

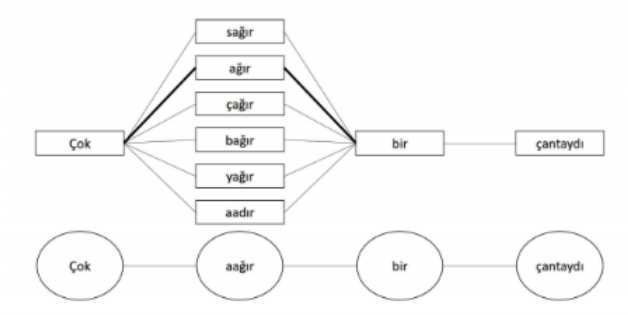

Figure 2: Viterbi algorithm for finding most probable sequence given a noisy word. [3]

Sinan Göker proposed two methods for Turkish text normalization purposes in his M.Sc. Thesis. First one is an unsupervised method. Study uses word embeddings so as to represent words and creates a lexicon which is a noisy-canonical(not noisy) word pairs with their cosine similarities. After that, cosine similarity values between word pairs is converted to lexical similarity cost which is a metric that is more lexicon-based. After that step, a bi-gram language model is constructed with BOUN dataset. Remaining steps are candidate extraction and candidate traversal which are very similar to the method proposed by Bölücü and Can. Method finds candidate words for noisy words, then applies Viterbi algorithm in order to find most probable sequence. This unsupervised method might be successful for fixing letter repetitions and one word misspellings. But it cannot handle phrases and abbreviations that are misspelled very well since the method is a combination of lexical metrics and neural word embeddings.

The second method that Göker proposes uses bidirectional LSTMs which is a supervised method. This method approaches to normalization problem as a machine translation problem, because of that it uses encoder-decoder architecture in order to convert grammatically mistaken sentences to their canonical form. While training the model, (noisy, canonical) word pairs are used. This method provided an efficient solution, but RNN (Recurrent Neural Networks) requires huge dataset in order to result well. If enough data is provided to encoder-decoder model, this method will give satisfying results. The two methods proposed by Göker worked better than Microsoft Word's spell checker, Zemberek's normalizer, rule based methods and method proposed in [2]. Figure 3 shows encoder-decoder architecture.

Solak and Oflazer [6] proposes a rule based method for spell checking purposes. This study finds and locates misspelled words by using a set of rules specific to Turkish language. Words are parsed with a morphological analyzer, then roots and

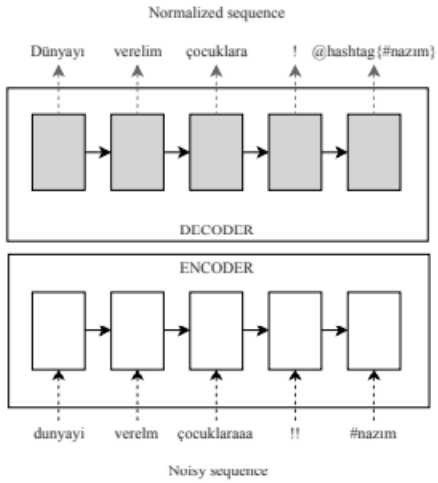

Figure 3: Encoder-decoder architecture for translation between noisy and canonical form. [5]

suffixes of words are passed to a finite state machine which is built by morphemic and phonetic rules of Turkish language e.g. vowel harmony. After traversing the finite state machine, this method checks whether given word is misspelled or not. This method might be used for detecting abnormal words in our project. Figure 4 shows the finite state machine architecture.

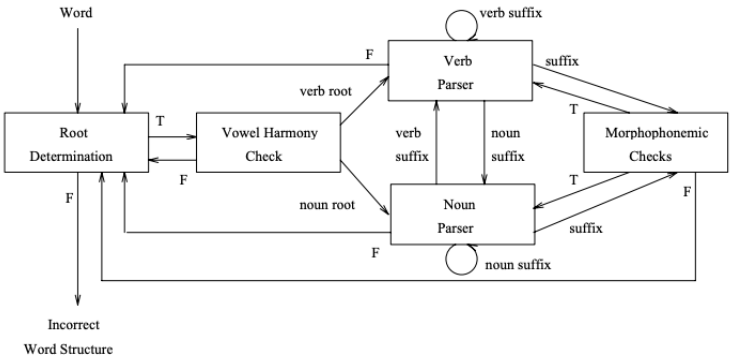

Figure 4: Finite state machine architecture of morphological analyzer. [6]

Göker's encoder-decoder method is the most similar approach in the literature that we have found to our approach. But Göker applied a word level normalization without any preprocessing layers and with very limited human-annotated dataset. Our aim is correcting spelling mistakes in sentence level with a huge amount of data gathered from Twitter and sufficiently corrected by previously mentioned methods automatically. Fortunately, Göker shared his encoder-decoder model's hyperparameters so we can use this hyperparameters while setting up our experiments.

Our plan also includes implementing methods proposed in [1, 2, 3] in order to exam-

ine their performance on new trend social media language and compare their results with our method.

3 System Design

3.1 System Model

Basic flowchart of our system is given in Figure 5.

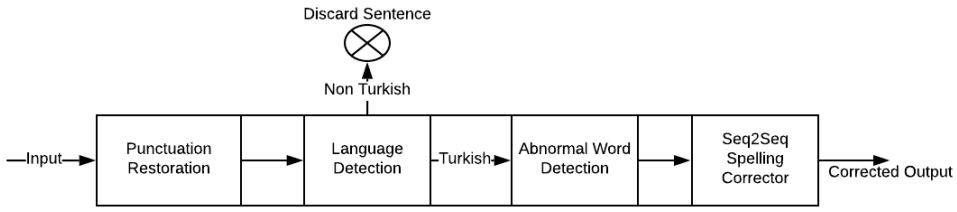

Figure 5: Basic flowchart of the system.

3.1.1 Punctuation Restoration

Punctuation restoration is the process of correcting punctuation mistakes in a text, especially correcting sentence-ending punctuations. We apply this process in order to segment text into sentences correctly, since it is important to correct spellings in sentence level. There can be more than one sentence in an input text and the meaning of these sentences can be very different from each other. If we can segment text into sentences correctly, we can correct spellings in sentence level without mixing the meanings of sentences.
This will be the first process applied to input given to our system. We will train a seq2seq model for punctuation restoration. This model requires sequence pairs as $(wrong\ punctuaion, correct\ punctuation)$ for training. We have collected Turkish news and Wikipedia data and we assumed that their punctuations are all correct. We will create their wrong punctuation version by removing/replacing punctuations randomly, especially sentence-ending punctuations. Input of the model will the wrong punctuation sentence, output of the network will be the sentence that punctuation mistakes are corrected. After preparing training data by this strategy and training, seq2seq model is ready for correcting punctuation mistakes.

3.1.2 Language Detection

Our system discards sentences that has less than 80% Turkish words. After punctuation restoration and sentence segmentation, system checks that sentence is 80% Turkish or not. We will use morphological analyzer and we will search each word of the sentence in the Turkish dictionary.

3.1.3 Abnormal Word Detection

If a sentence passes language detection, with more than 80% of words are Turkish, system needs to label abnormal words to fix. Proper nouns and abbreviations that could not pass language detection as Turkish, will not be labeled as abnormal words. Each word in the sentence is divided into roots and suffixes by a stemmer/morphological analyzer. Then roots and suffixes checked with a Turkish dictionary. After deciding which words are abnormal in the sentence, we need to check if it is a typo error or not.

3.1.4 Typo Detection

Detected abnormal words that have smaller minimum edit distance to the words in dictionary than some threshold value will be labeled as typo errors. The threshold value will be determined empirically.

3.1.5 Seq2Seq Model for Spelling Correction

After all mentioned steps, the sentences will be passed to seq2seq model for spelling correction purpose. Seq2seq models need huge amount of noisy/clean data pairs to be trained properly and it is hard to find satisfying amount of data for Turkish. In order to solve this problem, we will use our baseline models to create partially or fully corrected counterparts of collected noisy data. We will also synthetically generate noisy counterparts of collected clean data from news.

3.2 Description of The Algorithms

Word2Vec & fastText for word embeddings
There are various supervised and unsupervised methods for learning word embeddings. The most famous and revolutionary one is Word2Vec which was created in 2013 by Tomas Mikolov et al. at Google. Word2Vec algorithm described in [9, 10]. Word2Vec learns word embeddings by a shallow two-layer neural network. What makes Word2Vec special is its training speed and it considers context words while learning the embedding of target word by philosophy of "You shall know a word by the company it keeps.". Basically it says what determines a word's embedding is the surrounding words. For example, consider two sentences *"The cat sat on the mat"* and *"The dog sat on the mat"*. If we choose target words as *"cat"* and *"dog"* in each sentence, their context words will be exactly same. If we train Word2Vec algorithm with this corpus, we are expecting that words *"cat"* and *"dog"* will be close to each other in vectoral space.
While preparing training data for Word2Vec, one of the hyper parameters of algorithm is *window size*. Consider sentence *"The cat sat on the mat."*. If we choose window size as 2 and target word as *"sat"*, training data would become in the format (context, target):

- (w_{t-2}, w_t) - ("the", "sat")
- (w_{t-1}, w_t) - ("cat", "sat")

- (w_{t+1}, w_t) - ("on", "sat")
- (w_{t+2}, w_t) - ("the", "sat")

Word2Vec has two architectures which are CBOW(Continuous bag of words) and skip-gram. These architectures are shown in Figure 6. CBOW tries to predict target word given context words. Skip-gram tries the predict context words given target word. But what Word2Vec considers is not accuracy of classification; it considers about weights learnt by backprop between input layer and projection layer which is called "Embedding Matrix". Each row of this "Embedding Matrix" corresponds to a word vector.

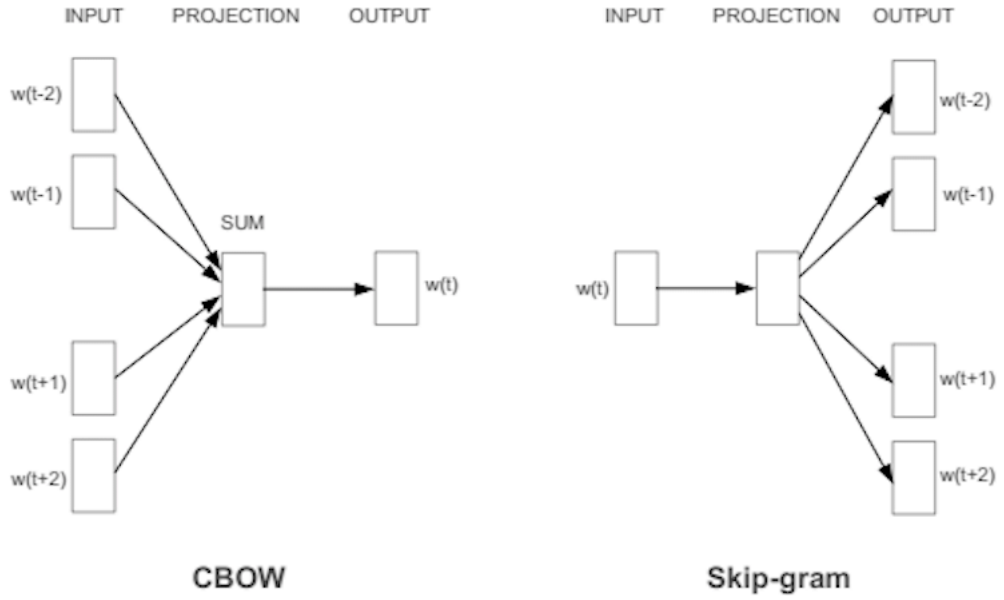

Figure 6: Word2Vec network architectures. [10]

Word2Vec has a big problem. It cannot generate embeddings for OOV words. "OOV words" means words which are not present in the corpus that used to train Word2Vec model. An alternative algorithm is fastText which is an improved version of Word2Vec. fastText also learns ngram vectors from training corpus, by this way fastText can generate word embeddings for OOV words by averaging ngram vectors.

fastText has also capability of text classification besides learning word embeddings. It is created by Facebook's AI Research (FAIR) lab. Algorithm's details are explained in studies [11, 12]. Main benefit of fastText is it can generate vectors for OOV words. It uses bag of ngrams to maintain efficiency without losing accuracy. It is very inefficient to store all ngrams with their vectors, so it uses hashing trick in order to maintain fast and memory efficient mapping of ngrams. There is a hyper parameter of fastText "bucket" which determines the number of available unique hashes. fastText first learns a vector for an ngram, then hashes the ngram (hash will be a value between 0 and [bucket-1]), then maps the vector to the $ngramvectors[hash]$. By this method, there can be collisions with ngrams resulting

same hash values, it is actually not a bad thing, it may add some randomness to fastText which can be considered as a good thing in NLP field. For words present in corpus, fastText has exactly the same processes with Word2Vec algorithm.

We are planning to use pretrained fastText models by way of Gensim library in order to obtain our word embeddings.

Edit Distance
Edit distance is the algorithm that will help us achieve typo detection task. Edit distance is the minimum number of operations, such as deletion, substitution and insertion, required for converting one text to another text. Also it is a famous dynamic programming algorithm. Consider two words *"yatak"* and *"battık"*. Edit distance between these two words is calculated as follows:

1. *yat**a**k to yat**ı**k*, substitution of a with ı.
2. ***y**atık to **b**atık*, substitution of y with b.
3. *batık to bat**t**ık*, insertion of t.

We can conclude that words *yatak* and *battık* has edit distance of 3 between them.

Encoder-Decoder Models
Encoder-decoder model is a type of sequence-to-sequence RNNs. It has sequence input and sequence output. Encoder-decoder architectures are mostly used in machine translation and speech recognition tasks. An encoder-decoder model can be divided into three parts, mostly designed as LSTM (Long Short Term Memory) cells, which are Encoder network, decoder network, and a state vector between them. Encoder network is responsible for extracting meaningful information from given input sequence and forwarding this extracted information to decoder network by way of state vector. Decoder network is responsible for unpacking information stored in state vector and generating sequence outputs.

Ilya Sutskever et. al. [13] showed efficiency of this architecture on machine translation task. The grammar and spell correction task might be considered as a machine translation problem, as Göker [1] did in his work, since input is an ill-formed text; output is a corrected form of given input text. An example of encoder decoder network shown in Figure 7.

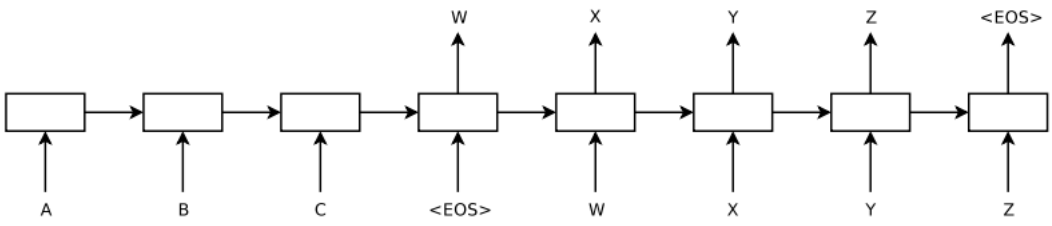

Figure 7: Sequence-to-sequence model for machine translation. [13]

We are planning to use this architecture while implementing our main deep learning architecture for grammar and spell checking purposes.

3.3 Comparison Metrics

Since the proposed the model falls into semi-supervised classification paradigm, standard metrics for classification can be used as success metrics. Primary metric for classification is accuracy. Accuracy will also be an efficient metric in order to compare our method with previously mentioned baseline algorithms since all of them used accuracy scores to show the success of proposed methods. In general, confusion matrix is generated at first after a classification task which is shown in Figure 8.

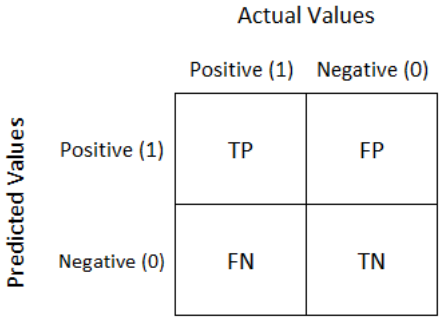

Figure 8: Confusion matrix.

- **TP**(True Positive): The number of samples that are actually positive and correctly classified as positive. In our case, this will be the number of correctly fixed misspelled inputs.
- **FP**(False Positive): The number of samples that are actually negative and classified as positive. In our case, this will be the number of corrected inputs which should not be corrected.
- **FN**(False Negative): The number of samples that are actually positive and classified as negative. In our case, this will be the number of inputs that are not corrected which should be corrected.
- **TN**(True Negative): The number of samples that are actually negative and classified correctly as negative. In our case, this will be the number of inputs that are not corrected which should not be corrected.

Accuracy score is calculated as shown in Equation 1. Since accuracy score is not sensitive to imbalanced datasets, we may need to use metrics which are more sensitive for imbalanced datasets. These metrics are precision, recall and F1 scores.

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{1}$$

Precision score gives information about how many of positive predictions are actually true. Its formula is given in Equation 2.

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

Recall score gives information about how many of actual positives predicted correctly. Its formula is given in Equation 3.

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

F1 score is a metric as a weighted average of recall and precision scores. By way of weighted averaging, precision and recall scores become more meaningful. Its formula is given in Equation 4.

$$F1Score = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} \tag{4}$$

Since we are going to implement a encoder-decoder architecture, there is a metric proposed by Papineni et. al. [8] called BLEU Score which is used for automatic evaluation of machine translation task. This metric might be useful for us to evaluate our encoder-decoder model.

Our project phases contain implementation of rule-based algorithms. We are planning to implement unit tests for each sub-components of rule-based algorithms. If these sub-components passes designed unit tests, they will be assumed as successful.

This project is considered as successful if these objectives are successfully completed:

- Since each phase of this project is a method of text normalization, completion of any phase will indicate that this project is successful.
- At least 70% accuracy obtained by a balanced (*Evenly distributed classes e.g. 50% positive class, 50% negative class.*) testing data set will indicate that this project is successful.
- If a balanced testing dataset cannot be obtained, at least 70% of F1-score will indicate that this project is successful.

3.4 Data Sets

All data that will be used in this project will be in text format. As mentioned before, we have collected noisy and clean data. Big portion of noisy data (tweets) are stored as structured Solr documents. Solr is an effective search engine application. Some of the tweets are stored only in text format without any other features, others are stored as text data, but the parts we are not interested in of tweets are annotated e.g. links, mentions, hashtags, emoticons, so we can easily remove these redundant parts from tweets easily. Noisy data that we do not store in Solr are stored in pure text format in text files. Clean data that we have obtained are stored in text files

with pure text format.

Our approach (encoder-decoder model) requires huge amount of noisy-canonical text sequence pairs to train and it is hard to obtain labeled data set for our purposes, especially in Turkish. We are planning to use baseline algorithms (See Section 2) to create partially or fully corrected forms of noisy data (tweets) to extend our training data set (noisy-canonical pairs). Another approach to extend training data set is to automatically generate noisy forms of clean data that we have obtained from Turkish news. We will generate noisy forms of these clean data by synthetic data generation algorithm that we will improve. Basically there will be 4 operations to generate noisy forms which are insertion of a new character, deletion of an existing character, transposition of two existing characters and substitution of an existing character in the text.

4 System Architecture

At first, raw input is given to the system. System applies punctuation restoration which helps segmenting text into sentences. As we stated in section 5.3, it is assumed that at least 80% of words in a sentence are Turkish. Language detection is applied in order to check 80% limit. If this limit is not satisfied, input will be discarded. If given limit is satisfied, sentence is tokenized and for each token, system checks that given token is abnormal or not using a morphological analyzer and a dictionary. If given token is not abnormal, given token will not be changed. If given token is abnormal, typo detection will be applied using edit distance and passed to the language model. If language model gives a probability higher than a threshold, which will be determined by testing different values, it outputs the canonical form of given abnormal token. If the input of language model coming from typo detection phase is not correct and language model cannot correct this token, raw input will be passed directly to sequence-to-sequence model. Same raw input will be provided to edit distance based algorithm. Sequence-to-sequence model will combined with edit distance based algorithm in order to generate the corrected form of given token.

High level flow chart of this project shown in Figure 9.

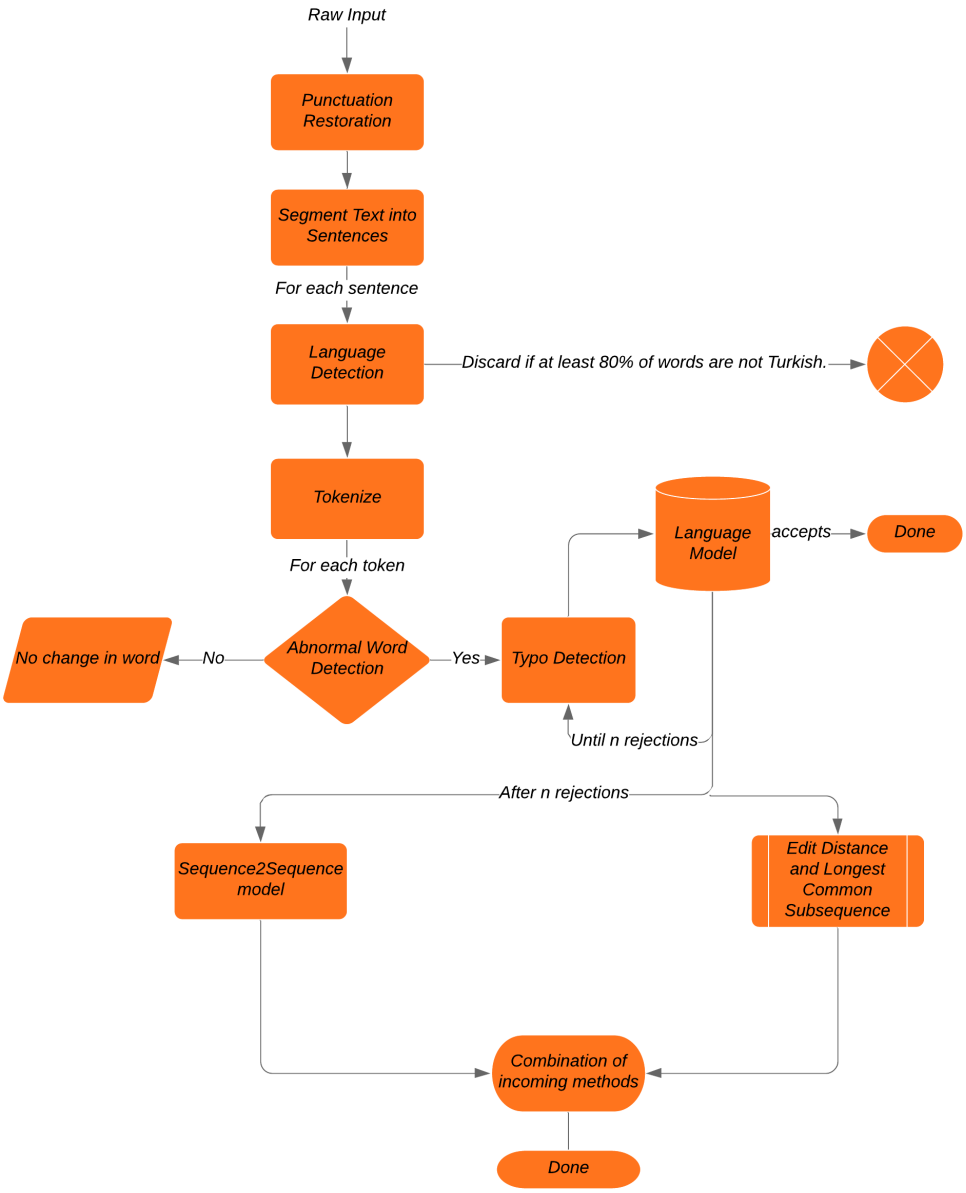

Figure 9: Flowchart of the system.

5 Experimental Study

5.1 Experimental Setup

We will start our experiments after the implementation of encoder-decoder model. We will train encoder-decoder model with clean and noisy data described in Section 3.4. It is assumed that data set that we have collected is enough for encoder-decoder model to be trained successfully. In training phase, it is assumed that we will have enough computational power. We will use NVIDIA RTX2080 Ti graphic card to have reasonable training time.

For testing the model, we will collect a couple hundreds of noisy tweets and annotate them manually. This data set will be used for comparison of our model with other existing studies. Result will be compared according to their accuracy/F1 score since this problem is considered as binary classification problem in baseline algorithms.

6 Tasks Accomplished

6.1 Current state of the project

- Clean data collected from news.
- Noisy data collected from twitter.
- Simple version of synthetic data generation is implemented.
  It selects random words from given sentences and randomly applies random operations to random characters. This method will be improved to catch common error patterns.
- Language detector for Turkish is implemented.
- Solr client is implemented with Java.
  Client is removing the unnecessary parts of the data and returns only meaningful text.

6.2 Task Log

Metting#1

Date: 06.06.2019
Location: Marmara University
Period: 2 Hours
Attendees: All group members
Objectives: Simple research about the algorithms that will be used in the project
Decisions and Notes: Our project scope has been decided

Metting#2

Date: 03.10.2019
Location: Marmara University
Period: 2 Hours
Attendees: All group members
Objectives: Presenting related works
Decisions and Notes: Methodology and baseline algorithms are determined

Metting#3

Date: 10.10.2019
Location: Marmara University
Period: 1 Hours
Attendees: Mert Kelkit
Objectives: -
Decisions and Notes: Presentation of related works has been done

Metting#4

Date: 10.10.2019
Location: Marmara University
Period: 1 Hours
Attendees: All group members
Objectives: We will start to collect data
Decisions and Notes: Methods for data collection is determined

Metting#5

Date: 17.10.2019
Location: Marmara University
Period: 1 Hours
Attendees: All group members
Objectives: Improvements will be done on PSD document
Decisions and Notes: We talked about draft version of the PSD document

Metting#6

Date: 10.10.2019
Location: Marmara University
Period: 1 Hours
Attendees: All group members
Objectives: Small mistakes will be corrected before submitting the final version
Decisions and Notes: Final version of PSD document is examined

Additional meetings has been done on Tuesdays when needed, to talk about the current state of the project and the future work.

6.3 Task Plan with Milestones

Tasks are listed below with their descriptions.

Task 1 Literature survey for related works.
Investigation of methodologies developed so far for Turkish text normalization, grammar and spell checking.

Task 2 Improvement of rule based algorithm for generation of synthetic examples.
A process that generates noisy words from canonical words by way of common error patterns.

Task 3 Implementation of [1, 2, 3] that will be used as baselines for comparison.

Task 4 Typo detection via syllable analysis and edit distance.
Detection of non-intentional misspellings.

Task 5 Sequence2sequence model for punctuation restoration.
Detection and correction of misplaced or missing punctuations.

Task 6 Development of semi-supervised sequence2sequence model on synthetic data (labelled) and real twitter examples (mostly unlabelled/some labeled via rule based algorithms).
Implementation of encoder-decoder architecture proposed in this project.

Task 7 Combining the sequence2sequence model with edit-distance/most-common subsequence based algorithm.
Combining this project's encoder decoder architecture with edit distance and longest common subsequence metrics.

Task 8 Implementation of web application for end users. Publishing final model as REST API and implementing a web application that uses API for end users.

Second semester Gantt chart is shown in Figure 10.

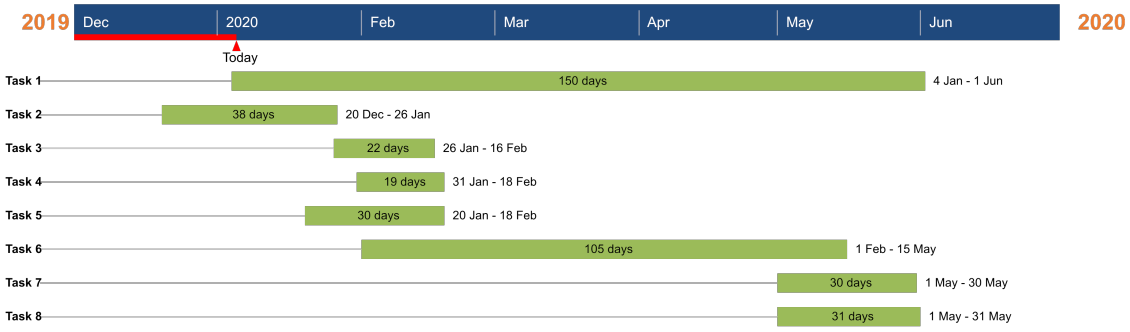

Figure 10: GANTT Chart for second semester tasks.

Our milestones are listed below:

1. Getting results of baselines after completion of Task 3.
2. Advanced sentence segmentation that will provide sentence level normalization after completion of Task 5.
3. Developing our deep learning method after completion of Task 6.

References

[1] S. Göker and B. Can, "Neural text normalization for turkish social media," in *2018 3rd International Conference on Computer Science and Engineering (UBMK)*, pp. 161–166, IEEE, 2018.

[2] D. Torunoglu and G. Eryigit, "A cascaded approach for social media text normalization of turkish," in *Proceedings of the 5th Workshop on Language Analysis for Social Media (LASM)@ EACL*, pp. 62–70, Citeseer, 2014.

[3] N. Bölücü and B. Can, "Context based automatic spelling correction for turkish," in *2019 Scientific Meeting on Electrical-Electronics Biomedical Engineering and Computer Science (EBBT)*, pp. 1–4, April 2019.

[4] H. Sak, T. Güngör, and M. Saraçlar, "Turkish language resources: Morphological parser, morphological disambiguator and web corpus," in *International Conference on Natural Language Processing*, pp. 417–427, Springer, 2008.

[5] A. A. Akın and M. D. Akın, "Zemberek, an open source nlp framework for turkic languages," *Structure*, vol. 10, pp. 1–5, 2007.

[6] A. Solak and K. Oflazer, "Design and implementation of a spelling checker for turkish," 1993.

[7] "Kemik - our datasets." `http://www.kemik.yildiz.edu.tr/?id=28`. (Accessed on 11/01/2019).

[8] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: a method for automatic evaluation of machine translation," in *Proceedings of the 40th annual meeting on association for computational linguistics*, pp. 311–318, Association for Computational Linguistics, 2002.

[9] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, pp. 3111–3119, 2013.

[10] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[11] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *Transactions of the Association for Computational Linguistics*, vol. 5, pp. 135–146, 2017.

[12] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, "Bag of tricks for efficient text classification," *arXiv preprint arXiv:1607.01759*, 2016.

[13] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, pp. 3104–3112, 2014.