İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

YAZIM KURALLARINA UYGUN YAZILMAMIŞ TÜRKÇE METİNLERİ MAKİNE ÇEVİRİSİ YÖNTEMLERİYLE NORMALLEŞTİRME

YÜKSEK LİSANS TEZİ

Talha ÇOLAKOĞLU

Bilgisayar Mühendisliği Anabilim Dalı

Bilgisayar Mühendisliği Programı

Tez Danışmanı: Doç. Dr. Ahmet Cüneyd TANTUĞ

EKİM 2019

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

YAZIM KURALLARINA UYGUN YAZILMAMIŞ TÜRKÇE METİNLERİ MAKİNE ÇEVİRİSİ YÖNTEMLERİYLE NORMALLEŞTİRME

YÜKSEK LİSANS TEZİ

Talha ÇOLAKOĞLU
(504161530)

Bilgisayar Mühendisliği Anabilim Dalı

Bilgisayar Mühendisliği Programı

Tez Danışmanı: Doç. Dr. Ahmet Cüneyd TANTUĞ

EKİM 2019

İTÜ, Fen Bilimleri Enstitüsü'nün 504161530 numaralı Yüksek Lisans Öğrencisi Talha ÇOLAKOĞLU, ilgili yönetmeliklerin belirlediği gerekli tüm şartları yerine getirdikten sonra hazırladığı "YAZIM KURALLARINA UYGUN YAZILMAMIŞ TÜRKÇE METİNLERİ MAKİNE ÇEVİRİSİ YÖNTEMLERİYLE NORMALLEŞTİRME" başlıklı tezini aşağıdaki imzaları olan jüri önünde başarı ile sunmuştur.

Tez Danışmanı :	**Doç. Dr. Ahmet Cüneyd TANTUĞ** İstanbul Teknik Üniversitesi	.....
Jüri Üyeleri :	**Doç. Dr. Gülşen ERYİĞİT** İstanbul Teknik Üniversitesi	.....
	Dr. Öğr. Üyesi Murat ORHUN İstanbul Bilgi Üniversitesi	.....
		.....

Teslim Tarihi : **25 Eylül 2019**
Savunma Tarihi : **04 Ekim 2019**

Hiçliğe,

ÖNSÖZ

Bu çalışmada bana büyük yardımları dokunan sayın danışmanım Doç. Dr. A. Cüneyd Tantuğ'a, 3000 km öteden fikirlerini ve eleştirilerini benden esirgemeyen, bu tezin başlamasında büyük katkısı olan sevgili arkadaşım Umut Sulubacak'a teşekkürlerimi iletmek isterim.

Ekim 2019

Talha ÇOLAKOĞLU
Bilgisayar Mühendisi

İÇİNDEKİLER

Sayfa

ÇİZELGE LİSTESİ

Sayfa

ŞEKİL LİSTESİ

Sayfa

YAZIM KURALLARINA UYGUN YAZILMAMIŞ TÜRKÇE METİNLERİ MAKİNE ÇEVİRİSİ YÖNTEMLERİYLE NORMALLEŞTİRME

ÖZET

Sosyal medya kullanımının hayatımızda yaygınlaşmasıyla beraber, çevrimiçi üretilen içerik daha önce görülmemiş boyutlara ulaşmıştır. Çoğunlukla dil bilgisi kurallarına uyulmadan yazıldığı için, bu metinleri geleneksel doğal dil işleme araçlarıyla işlemek zordur. Metin normalleştirmesi, kurallara uyulmadan yazılmış metinlerin, dil bilgisi kurallarına uygun yazılmış hale çevrilmesi işlemine denir. Genellikle bu işlem diğer doğal dil işleme araçlarının ön aşamasıdır ve o araçların başarımını artırmaktadır.

Metin normalleştirme, geleneksel normalleştirme yaklaşımlarının dışında, kuralsız-kurallı metin arasında makine çevirisi problemi olarak da ele alınabilir. Fakat bu tarz sistemleri eğitebilmek için büyük miktarlarda paralel veri gerekir. Diğer makine çevirisi problemlerinin aksine, bu konuda daha önce işaretlenmiş herhangi bir veri bulunmadığı için, bu tez kapsamında öncelikle yapay paralel veri üretilmesine odaklanılmıştır.

Paralel veri üretmek için iki yönteme başvurulmuştur: raslantısal olarak belli kurallar kapsamında düzgün kelimeyi bozma ve düzgün-bozuk kelime çiftlerini eşleştirme. Bu yöntemler için, hatalı ve doğru yazılmış kelimelerin çoğunlukta olduğu iki farklı derleme ihtiyaç vardır. Hatalı yazılmış kelimeleri barındıran derlemi oluşturmak için Twitter'dan Kasım 2018 - Ocak 2019 tarihleri arasında yirmi beş milyon tekil tweet çekilmiştir. Düzgün kelime kaynağı olaraksa altyazı derlemi seçilmiştir. Bu iki derlemdeki kelimeler ön işlemden geçirildikten sonra, yukarıda bahsettiğimiz yöntemler kullanılarak paralel veri üretilmiştir. Düzgün-Bozuk kelime çiftlerini eşleştirme yöntemi için ağırlıklandırılmış Levenshtein uzaklığı esas alınmıştır. Bu yöntemin yakalayamadığı veya yakalamasının zor olduğu hata tipleri için ise raslantısal olarak belli kurallar kapsamında düzgün kelimeler bozulmuştur.

Üretilen yapay paralel veriyle beraber makine çevirisi sistemleri eğitilmiştir. Buna ek olarak, ön ve son işleme bileşenleriyle beraber metin normalleştirme sistemi oluşturulmuştur. Oluşturulan sistemin önceki sistemleri yüksek bir farkla geçtiği görülmüştür.

NORMALIZING NON-CANONICAL TURKISH TEXTS USING MACHINE TRANSLATION APPROACHES

SUMMARY

With the increase of online user generated content (UGC), text normalization has gained a huge importance on natural language processing. People now express their ideas and thoughts using social media platforms such as Twitter, Facebook and Youtube. Additionally, it is common to have a comment section in news page and review section e-commerce pages. Ignoring grammatical rules, omitting letters and other spelling mistakes are often the case while posting content to these places. The content of these data are quite valuable to many firms which like to keep track of their customer's satisfaction. Nevertheless, colloquial writing makes difficult to process immense amount of data generated by humans.

Many NLP tools are designed to work well only with formally written text rather than informal. Due to UGC's idiomatic nature, traditional NLP tools require a preprocessing step that we call text normalization. Text normalization for Turkish has been categorized into 6 groups: Letter Case Correction, Diacritic Restoration, Vowel Restoration, Accent Normalization, Spelling Correction, Other Errors (repeating characters, abbreviation). Since the labeled data is limited in this field, previous systems tend to use rule based system and statistical systems (trained on autmatically created data). However, these systems can not generalize as well as machine learning algorithm does. It is shown that machine translation systems are capable of correcting not only local errors like letter omission but also grammatical errors like subject-verb disagreement [1].

Labeled data for text normalization is a scarce resource. In addition to that, labeling data, especially training data, takes a lot time and human resource to finish. Given the lack of manually data for the text normalization problem and the data hunger of machine translation systems, the necessity of artificially producing data has emerged. Producing data only using rule-based methods (such as dropping vowels, replacing letters with similar ones) will inevitably lead to bias in the data. At the same time, this method may sometimes produce unrealistic words. Not only that, we will not be able to know every error in the language, and even if we do, it will be very difficult to realize, so this method will be insufficent to produce artificial data. Therefore, in order to obtain a more realistic corpus, the artificial data was produced by using weighted Levenshtein algorithm. Since this method matches the noisy words obtained from a real collection with the clean words, it is aimed to produce more realistic data than the previous method. Clean words are collected from an existing collection and filtered through the morphological analyzer. The Twitter website is chosen as a ill-formed word source and millions of tweets are scraped from this site. These tweets were then parsed with a tokenizer and the non-word components were eliminated. However, due to computing limitations, even though catching local errors are easy, it is more difficult to spot errors that spread over multi-words (e.g. question phrases like *gruyormusun?*

vs *görüyor musun?* with this approach. To tackle this problem, we use additional rule-based approach that roughly simulates these types of errors.

Since Turkish is a morphologically rich, additive language, it has a very rich vocabulary. Therefore, it is hard for to catch the all spelling errors found in the Turkish by pure Levenshtein algorithm. Based on this, new features have been added to the Levenshtein algorithm to capture of spelling errors broadly. One of the spelling errors occurred online environments is typographical errors and one of these errors is the accidental typing of a letter with adjacent letter on the keyboard. For example, it is possible to write "geldfi" when trying to write "geldi" since letter *d* and letter *f* are adjacent letter on Q keyboards. It is impossible to catch these error using existing Levenshtein weights. For that reason, "Delete Adjacent Weight" feature is added to the algorithm to detect these errors. Another spelling error of occurs a lot is the intentional spelling mistakes by rewriting a certain letter to stress a word or writing the spoken language as it is. In addition to that, Since the Levenshtein algorithm make comparison only between two strings, it cannot capture the errors that occur in only one string. To overcome this, "Repetitive Letter Weight" feature, which includes the letters in the same string is added the Levenshtein algorithm. In addition to all,"Stopping by Threshold" feature is added to minimize running time of the algorithm.

To produce clean-noisy word pairs of high quality, one needs to set parameters of the Levenshtein algorithm precisely. To achieve that the parameters are first set intuitively, then the parameters are fine-tuned on Val_{small} with repetitive runs. After the parameters are determined and the algorithm is run, as a result a noisy word lists for every clean word is obtained. These lists doesn't always contain correct noisy words which needs to be dismissed. Eliminating all mismatches is not possible because it requires manual control. But leaving the lists intact will negatively affect the performance of the translation system. Therefore, it is necessary to apply a selection method that will negatively affect the performance of the translation system to a minimum and eliminate as many false matches automatically as possible without requiring manual control. As a selection method, a popular genetic algorithm "Roulette Wheel Selection" is chosen. Using this method, first a Levenshtein distance is selected from 0.0 to 1.0 range. After the distance selected, one word in that distance is randomly selected as a noisy word for that spesific clean word. Apart from these, there have been situations such as catching the wrong spelling of multi-word phrases and catching the wrong spelling of special names. Such errors are simulated by means of coincidence according to random rules.

The proposed translation pipeline consists of 3 main components, a pre- and post-processing component as well as a translation component. The preprocessing component provides a basic normalization at the letter level, but also converts all letters to lower case. This reduces the complexity of the translation process as it reduces vocabulary. The post-processing component truecases output of the translation component consisting of only lower case letters. Lastly, the translation component performs the normalization.

In this thesis, It is proposed to solve the Turkish text normalization problem by using machine translation approaches. In contrast to the previous state of the art, the proposed system can solve the problem with fewer components and include the context. Furthermore, the system is less affected by human errors as it does not contain rule-based components. In addition, the system we present does not require external natural language processing tools.

According to the results, it has been observed that the character-based statistical machine translation method proposed in this thesis exceeds the best available study with high margins on different set of test sets. These results show that our unattended parallel data generation methods and machine translation method show high performance in solving Turkish text normalization problem.

1. GİRİŞ

Web 2.0 ile birlikte çevrimiçi olarak üretilen içerikte eşi görülmemiş bir artış yaşanmıştır. Facebook ve Twitter gibi etkileşimli internet sitelerinin popülerleşmesiyle beraber, insanların duygu ve düşüncelerini sıklıkla bu sitelerde yayınlaması bu çevrimiçi içeriğin büyük bir kısmını oluştururken, haber ve e-ticaret sitelerinde yer alan yorum bölümleri de bu içeriğin oluşmasında katkıda bulunmuştur. Birçok şirket veya devlet kuruluşu için, bu içerikler önem arz etmektedir. Fakat üretilen bu verinin büyük bir çoğunluğunun herhangi bir dil bilgisi kuralını takip etmemesi, klavye hatalarından doğan yazım hataları ve bilerek bir kelimenin yanlış yazılması gibi nedenlerden dolayı, geleneksel dil araçlarıyla işlenmesi için öncelikle ön bir aşamadan geçmesi gerekmektedir. Kuralsız yazılmış bir metni, dil bilgisi kurallarına uygun yazılmış bir metne çevirmeye metin normalleştirmesi denir. Metin normalleştirmesi, genellikle kök çözümleme (lemmatization), POS etiketlemesi, ayrıştırma (parsing), varlık isim tanıma (NER) gibi diğer doğal dil işleme işlerinin önadımıdır. Tipik bir metin normalleştirme sistemi Şekil 1.1'de görülmektedir.

CRF'ler, SVM'ler ve yapay sinir ağları gibi denetimli makine öğrenme yöntemleri, çok çeşitli dil işleme işlemleri için standart çözümler getirmiştir. Bu yöntemler tipik olarak veri odaklıdır ve potansiyellerinin zirvesine ulaşmak için fazla miktarlarda veri ile eğitilmesi gereklidir. Bu tür bir verinin elde edilmesi için çoğunlukla elle işaretleme yapılması gerektiğinden, bu durum bu tarz sistemlerin en büyük kısıtlayıcı noktasını oluşturmaktadır. Elle işaretlenmiş verinin az olduğu görevlerde, bu durum daha da ön plana çıkmaktadır. Bu görevlere, elle işaretlenmiş verilerin görece az olduğu ve otomatik yöntemler kullanılarak veri elde etmenin zor olduğu, metin normalleştirme ve makine çevirisi sistemleri örnek olarak verilebilir.

Şekil 1.1 : Tipik metin normalleştirme örneği.

Metin normalleştirme problemi, birbiriyle yakından ilişkili iki dil arasında makine çevirisi problemi olarak ele alınabilir. İstatiksel makine çeviri yöntemleri, uzun bir süredir bu alandaki en güçlü sistemlerdi. Yakın zamanda derin öğrenme algoritmalarının popülerleşmesiyle beraber yapay sinir ağlarını kullanan yapay sinir ağı temelli çeviri yöntemleri de makine çevirisi problemini çözmede sıklıkla kullanılır olmuştur. Modüler yapısından dolayı, istastiksel çeviri sistemleri veri azlığı problemine karşı daha dirençlidir. Aynı zamanda, bu sistemlerin bazı bileşenleri (örneğin dil modeli eğiticisi) paralel veri bile gerekmeden eğitilebilmektedir. Yapay sinir ağı temelli çeviri yöntemleri, istatiksel çeviri yöntemlerinin aksine, eğitim için bol miktarda veriye ihtiyaç duymaktadır. Genellikle bu sistemler bol veriden görece daha iyi öğrenebilirken, veri azlığında performansları düşmektedir. İki yöntem de genellemede, yani eğitim verisinden yansız öğrenmede, oldukça güçlüdür.

Metin normalleştirme problemi için elle işaretlenmiş veri eksikliği, makine çevirisi sistemlerinin veri açlığı gözönünde bulundurulduğunda, yapay olarak veri üretme zorunluluğu ortaya çıkmıştır. Sadece kurallara bağlı (ünlülerin düşürülmesi, harflerin benzerleriyle değiştirilmesi gibi) veri üretmek, veride kaçınılmaz bir yanlılığa sebep olacaktır. Aynı zamanda bu yöntem, bazen gerçekçi olmayan kelimeler üretebilecektir. Bununla da kalmayıp, dilde gerçekleşen her hatayı bilemeyeceğimizden, bilsek bile bunu gerçekleştirmek çok zor olacağından bu yöntem yapay veri üretmede eksik kalacaktır. Bundan ötürü, daha gerçekçi bir derlem elde etmek için, Levenshtein algoritması kullanılarak veri üretilmesi yoluna gidilmiştir. Bu yöntem, gerçek bir derlemden elde edilen bozuk yazılmış kelimeleri, düzgün yazılmış kelimelerle eşleştirdiğinden diğer yönteme göre daha gerçekçi verilerin üretilmesi hedeflenmiştir. Düzgün yazılmış kelimeler, halihazırda varolan bir derlemden çekilmiş ve biçimbilimsel çözümleyici ile filtrelenmiştir. Bozuk yazılmış kelime kaynağı olarak Twitter websitesi seçilmiş ve bu siteden milyonlarca tweet çekilmiştir. Sonrasında bu tweetler bir bölütleyici ile (tokenizer) ayrıştırılmış (parse) sözcük olmayan bileşenler elenmiştir.

Türkçe, biçimbilimsel olarak zengin, eklemeli bir dil olmasından ötürü oldukça geniş bir kelime haznesine sahiptir. Dolayısıyla, Türkçe'de bulunan yazım hataları da saf Levenshtein algoritmasının yakalayamayacağı boyuttadır. Bundan yola çıkarak, Levenshtein algoritmasına yeni özellikler eklenmiş ve Türkçe'de varolan

hataların daha iyi yakalanması hedeflenmiştir. Buna ek olarak, sadece Levenshtein algoritmasıyla üretilen veriler yetersiz kalacağı için, ek olarak kurallara bağlı rastlantısal yöntemlerle de veriler üretilmiştir. Bunların sonucunda ortaya çıkan yapay paralel derlem üzerinde çalışmalar yapılmıştır.

Bu çalışmada, makine çevirisi yöntemlerinin Türkçe metin normalleştirme problemini çözmedeki potansiyelleri araştırılmıştır. Bu amaç doğrultusunda, çeşitli yöntemler ile yapay paralel veri üretilmiş ve bu veriyle birlikte karakter tabanlı istatiksel ve nöral makine çevirisi sistemleri eğitilmiştir. Bu sistemlere ek olarak, ortografik normalleştirici ve büyük-küçük harf düzeltici sistemleri oluşturulmuştur.

1.1 Literatür Taraması

Türkçe için metin normalleştirme problemine sunulan ilk çözümler kural tabanlı ve istatiksel yöntemlerin bir arada kullanılmasıyla oluşmuştur. Bu konuda Türkçe için şu ana kadar en iyi sonuçları veren çalışma, [2]'in devamı niteliğinde olan [3] çalışmasıdır. Bu çalışmada, Türkçe metin normalleştirme probleminin temelleri atılmış ve varolan problemler 7 ayrı kategoride incelenmiştir. Her kategorideki problem için kural tabanlı sistemler kullanılmış veya istatiksel tabanlı sistemler eğitilmiş, bu sistemler sonrasında karmaşık bir işhattı ile birbirlerine bağlanmıştır. Sistemin kural tabanlı bileşenleri, karmaşık işhattı yapısı ve bunlara ek olarak harici dil doğrulayıcısına ihtiyaç duyması bu sistemin zayıf tarafını oluşturmaktadır. Bildiğimiz kadarıyla bu çalışmadan sonra yapılan tek çalışma yapay sinir ağlarını bu problem üzerinde deneyen [4] çalışmasıdır. Bu çalışma, önceki en iyi sistemden [2] daha iyi başarı gösterirken, daha sonra geliştirilen en iyi sisteme [3] göre daha düşük performans göstermektedir.

Sosyal medyanın ve çevrimiçi içeriğin yaygın olmadığı devirlerde, metin normalleştirme sistemleri kısa mesajlar (SMS) ve elektronik ortam metinleri (sohbet mesajları, e-posta metinleri) normalleştirme için kullanılıyordu. Kısa mesajlar için ilk çalışmalardan biri olan [5] çalışmasında, normalleştirme problemi çözmek için Saklı Markov Modeli kullanılmıştır. [6] çalışmasında kısa mesajların normalleştirilmesi, öbek tabanlı istatiksel makine çeviri yöntemiyle yapılmıştır. [7] çalışmasında ise kısa mesajların gürültülü kanal modeli ile normalleştirilmesi önerilmiştir. Yine Twitter metinlerinin yanı sıra kısa mesaj normalleştirmesi üzerinde de duran, [8]

çalışmasında, bir adet sınıflandırıcı ile yanlış yazılmış kelimeler saptanmış, sonrasında bu kelimeler için kelime benzerliğine göre aday kelimeler üretilmiştir. Son olarak adaylar arasında bağlam da dikkate alınarak en olası kelime seçilmiştir. Aynı alanda [9] çalışmasında, metin normalleştirme problemi diğer çalışmalardan farklı kategorize edilmemiş ve gözetimsiz eğitilmiş bir sistemle çözülmüştür. Fransızca kısa mesajların normalizasyonu için [10] çalışmasında, metin normalleştirme probleminin yazım hatası düzeltme, makine çevirisi, otomatik konuşma tanıma sistemlerinde çözülen problemlere benzerliğinden bahsedilip, bu problemleri çözen sistemlerin avantajlarından yararlanan hibrit bir yöntem önerilmiştir. [11] çalışmasında yine Fransızca kısa mesaj metinleri için, kural tabanlı, makine çevirisi ve yazım hatası düzeltme sistemlerine benzer sistemlerinin birleşimi bir yöntem normalleştirme probleminin çözümü için önerilmiştir.

Sosyal medyanın yaygınlaşmasıyla beraber, araştırmacılar bu alandaki metinlerin normalleştirmesine yönelmiştir. [12] çalışmasında kural tabanlı sözlük bakma algoritmasıyla sosyal medya metinleri normalleştirilmeye çalışılmıştır. [13] çalışmasında katar ve fonetik benzerlik algoritmaları kullanılarak aday kelimeler elde edilmiş ve bu kelimeler gürültülü kanal modeli (Noisy Channel Model) eğitiminde kullanılmıştır. [14] çalışması, [8] çalışmasının iyileştirilmiş hali olup, yine problemi benzer şekilde çözmüştür. [15] etiketsiz veri kullanılarak, bağlamı da gözeterek oluşturulan ikili çizge (bipartite graph) üzerinde rastsal yürüme algoritması koşturularak normalleştirme işlemini gerçekleştirmiştir. [16] Almanca sosyal medya yazıları elle oluşturulmuş kurallar yardımıyla normalleştirilmiştir. [17] çalışmasında, heceler temel birim olarak kabul edilip, geleneksel gürültülü kanal modeli kullanılarak normalleştirme yapılmıştır. Önerilen yöntem dilden bağımsız ve etiketli veri gerekmeden kullanılabilmektedir. [18] öncelikle 6 farklı kural tabanlı sistem kullanarak her yanlış yazılmış kelime için aday düzgün kelimeler oluşturmakta, sonrasında ise dil modeli kullanarak bu adaylar arasında seçim yaparak normalleştirme işlemini gerçekleştirmektedir. [19] çalışmasında dillerden bağımsız modüler ve verimli bir sistem üretmiştir. Sistemde öncelikle kural tabanlı ve derin öğrenme yollarıyla aday kelimeler oluşturulur. Sonrasında ise bu aday kelimeler arasından Rastgele Orman algoritması ile seçim yapılır. Bu çalışmanın birçok dilde, hali hazırda bulunan en iyi sistemleri geçtiği görülmüştür. [20] çalışmasında Uygurca metin

normalleştirme problemi, gürültülü kanal modeli ve kodlayıcı-çözücü mimarisiyle çözülmeye çalışılmıştır. Bu sistemlerin eğitimi için gereken veri basit bir rastlantısal karakter değiştirme yöntemi ile oluşturulmuştur.

Karakter tabanlı makine çevirisi yöntemi ilk olarak [21] çalışmasında tranliterasyon için kullanılırken, daha sonra bu methodun metin normalleştirmesi için kullanılabileceği de önerilmiştir. Bu öneriden sonra bir çok çalışma karakter tabanlı istatiksel makine çevirisi yöntemlerini birbirine yüksek benzerlik gösteren iki metin arasında çeviri için kullanmıştır. [22] çalışmasında kısa mesajlarda bulunan yazım hatalarını karakter tabanlı makine çevirisi yöntemiyle düzeltilmesi önerilmiştir. Sistemleri elle etiketlenmiş veri üzerinde eğitilmiş ve bağlamı da işin içine katmıştır. Bu sistemin yanı sıra makine çevirisi yöntemleri benzer şekilde [23–27] çalışmalarında metin normalleştirmesi için kullanılmıştır. Karakter tabanlı istatiksel makine çevirisi yöntemlerinin Türkçe'deki performansı ilk kez bu çalışmada araştırılmıştır.

2. MAKİNE ÇEVİRİSİ YÖNTEMLERİ

2.1 İstatiksel Makine Çevirisi Yöntemleri

İstatiksel makine çevirisi, iki dilli paralel derlemler kullanılarak eğitilen istatiksel modellere dayanan bir çeviri yöntemidir. İstatiksel makine çevirisi yönteminde, bir metnin çevirisi kaynak dildeki metnin (örneğin Türkçe) hedef dildeki metne (örneğin İngilizce) çevrilme olasılığı $p(Ing|Tr)$ olarak ifade edilir. Bu ifade *Bayes teoremi* kullanılarak açılırsa $p(Ing|Tr) \propto p(Ing) \times p(Tr|Ing)$ olarak ifade edilebilir. Bu iki ifadeyi makine çevirisi bağlamında anlamlandırmak gerekirse, $p(Tr|Ing)$ ifadesi kaynak dildeki metin dizisinin hedef dildeki dizinin çevirisi olma olasılığını ifade ederken, $p(Ing)$ ise çevrilen metnin hedef dilde bulunma olasılığını ifade eder. Başka bir deyişle, $p(Tr|Ing)$ çeviri modelini temsil ederken, $p(Ing)$ hedef dilin *dil modelini* temsil etmektedir. Son olarak en iyi çeviri, bu ifadelere göre en yüksek olasılığı veren cümle olarak seçilir. Bu da matematiksel olarak $\tilde{Ing} = \underset{Ing}{\text{argmax}} p(Ing|Tr) = \underset{Ing}{\text{argmax}} p(Ing) \times p(Tr|Ing)$ ifade edilebilir.

2.2 Yapay Sinir Ağı Temelli Makine Çevirisi Yöntemleri

yapay sinir Ağı temelli makine çevirisi, yapay sinir ağlarını kullanarak, tüm çeviri işlemini bütünleşik bir model ile gerçekleyen bir makine çeviri yöntemidir. İstatiksel yöntemlerin aksine, çeviri işlemi ayrı ayrı bileşenler yerine uçtan uça bütünleşik sistem üzerinden yapılır. Bu tarz sistemlerin eğitilmesi için büyük miktarda veri ve bilgisayar kaynağı gerekmektedir. Fakat eğitilmiş bir model, istatiksel yöntemlere göre oluşturulmuş modele göre daha az bilgisayar kaynağı gerektirmektedir. Bu yöntemde kullanılan derin öğrenme teknikleri sayesinde, istatiksel yöntemlerde başarılamayan veya başarılması çok büyük hafıza gerektiren işlemler kolaylıkla yapılabilmektedir. Örneğin kelime vektörleri sayesinde teorik olarak sonsuz uzunlukta bağlam makine çevirisinde kullanılabilir. Yapay sinir ağı temelli makine çevirisi sistemleri çoğunlukla kodlayıcı-kod çözücü mimarisindedir. Bu mimaride genellik RNN, LSTM ve GRU gibi derin öğrenme sistemleri kullanılır. Son olarak birçok büyük şirketin makine çeviri

sistemleri istatiksel yöntemlerden yapay sinir ağı temelli makine çevirisi yöntemlerine kaymıştır.

3. VERİLERİN ELDE EDİLMESİ VE İŞLENMESİ

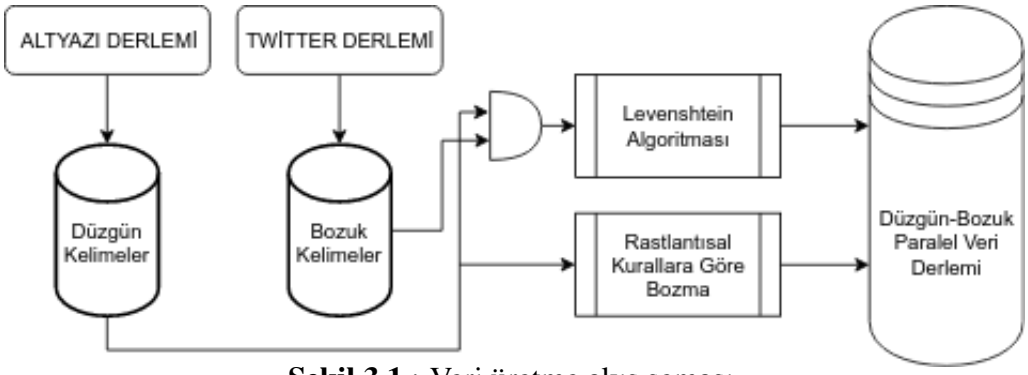
Makine çevirisi sistemlerini eğitebilmek çeviri yapılacak dillerden oluşan paralel derleme ihtiyaç vardır. İngilizce-Türkçe paralel derlemi, içerik olarak aynı olan (altyazı gibi) Türkçe ve İngilizce metinlerden oluşan derlemlerin otomatik hizalanmasıyla paralel derlem kolayca oluşturulabilmektedir. Fakat, metin normalleştirme konusunda daha önce işaretlenmiş yeterli paralel verinin bulunmaması ve bu verinin otomatik olarak başka kaynaklardan oluşturulamamasından dolayı, yapay olarak paralel veri üretilmesi gerekliliği ortaya çıkmıştır. Bu tez kapsamında paralel derlem üretilmesinde iki yöntem izlenmiştir:

- Bozuk ile Düzgün Kelimelerin Levenshtein Algoritması ile Eşleştirilmesi
- Rastlantısal Kurallara Bağlı Olarak Veri Üretme

Bu yöntemlerin birbirlerine göre farklı avantajları ve dezavantajları vardır. Her ne kadar rastlantısal kurallara bağlı yöntemlerle veri üretmek kolay olsa da, bu yöntemin birkaç dezavantajı vardır. Bunlar:

- Bu yöntem kullanılarak düzgün kelimeden üretilen bozuk kelime aslında gerçek yazım hatalarıyla bağdaşmayabilir. Örneğin, bu yöntemi kullanılarak "aslında" kelimesini rastlantısal olarak bozmaya çalışırsak "slıda" gibi bir kelime üretebiliriz. Bu tip durumlardan kaçınılmaya çalışılsa bile, bu yöntemin doğası gereği veri üretimine kattığı bir yanlılık (bias) vardır.
- Dilde bulunan her yazım hatası tipini bilmediğimizden ötürü birçok yazım hatası modellemek imkansızdır. Dolayısıyla veri üretmek için sadece bu yöntemi kullanmak birçok hatayı ıskalamak anlamına gelmektedir.

Bu dezavantajlardan ötürü, veri üretimi için Levenshtein algoritması ile bozuk-düzgün kelime çifti eşleştirilmesi yöntemi tercih edilmiştir. Bu yöntemle birlikte gerçek yazım hataları yakalanabilecektir. Fakat bu yöntemin de kendi içerisinde birçok dezavantajı vardır. Bunlar:



Şekil 3.1 : Veri üretme akış şeması.

- Bu yöntemle sadece iki kelime birbirleriyle karşılaştırıldığından birden fazla kelimeyi kapsayan yazım hatalarını yakalamak zordur. Her ne kadar algoritmaya girdi olarak ikiden fazla kelime verilebiliyor olsa da bu durum bilgisayımsal (computational) olarak zorluk çıkaracaktır. Örnek olarak, "Yapıyonmu", "Ankara da", "hiç bir" gibi hataları yakalamak bu yöntemle zordur.

- Her kelime çifti için sonuç üretilememektedir. Örneğin "negatifliğinden" kelimesi için herhangi bir sonuç üretilememektedir.

- Doğru aday kelimeleri belirlemek zordur. Örneğin "geliyorum" kelimesi için, "gliyrm" aday kelimesi üretilebileceği gibi "geriliyorum" aday kelimesi de üretilebilmektedir.

- Bu yöntemle yeterli miktarda veri üretebilmek için yüksek seviyede bilgisayar kaynağı ve süre gerekmektedir.

Sistemlerin barındırdığı bu dezavantajlar, veri üretmek için bizi hibrit model kullanmaya yöneltmiştir. Bu modelin akım şeması Şekil 3.1'de görülmektedir. Paralel derlem oluşturabilmek için öncelikle düzgün yazılmış kelimeler bulunduran ve bozuk yazılmış kelimeler bulunduran iki farklı derlem gerekmektedir. Sonraki bölümlerde, bu iki derlemin oluşturulmasından ve istatistiklerinden bahsedildikten sonra, bu verilerin işlenmesi için geliştirilen algoritmalar ve paralel veri üretiminin adımları anlatılacaktır.

```
{
  "_id":ObjectId("5c6583ccea09615879405569"),
  "created_at":"Thu Feb 14 15:05:43 +0000 2019",
  "id":NumberLong("1096062939549839361"),
  "id_str":"1096062939549839361",
  "text":"@sebnemziyagil https://t.co/4TB8duhZTf'a ilan verelim senin için",
  "display_text_range":[ ⊞ ],
  "source":"<a href=\"http://twitter.com/download/iphone\" rel=\"nofollow\">Twitter for iPhone</a>",
  "truncated":false,
  "in_reply_to_status_id":NumberLong("1095964076117757953"),
  "in_reply_to_status_id_str":"1095964076117757953",
  "in_reply_to_user_id":NumberLong("1025677183325556736"),
  "in_reply_to_user_id_str":"1025677183325556736",
  "in_reply_to_screen_name":"sebnemziyagil",
  "user":{ ⊞ },
  "geo":null,
  "coordinates":null,
  "place":null,
  "contributors":null,
  "is_quote_status":false,
  "quote_count":0,
  "reply_count":0,
  "retweet_count":0,
  "favorite_count":0,
  "entities":{ ⊟
    "hashtags":[ ⊞ ],
    "urls":[ ⊞ ],
    "user_mentions":[ ⊞ ],
    "symbols":[ ⊞ ]
  },
  "favorited":false,
  "retweeted":false,
  "possibly_sensitive":false,
  "filter_level":"low",
  "lang":"tr",
  "timestamp_ms":"1550156743310"
}
```


Şekil 3.2 : Tweet objesi örneği.

3.1 Verilerin Toplanması

Paralel veri oluşturulması için gereken iki derlem kaynağı olarak Twitter'da bulunan Türkçe tweetler ve Türkçe film altyazıları seçilmiştir. Twitter kuralsız yazılmış metin kaynağı olarak kullanılırken, altyazılar düzgün yazılmış metin kaynağı olarak kullanılmıştır. Bu veri kümelerine ait istatistikler Çizelge 3.1'de görülmektedir.

Çizelge 3.1 : Derlem kaynağı büyüklükleri.

Derlem Kaynağı	# Cümle/Tweet	# Tekil Sözcük
Twitter	25M	5M
Altyazı	161M	1.1M

3.1.1 Twitter verisi

Kasım 2018-Ocak 2019 tarihleri arasında Twitter API kullanılarak 40 milyona yakın tweet çekilmiştir. Bu tweetler MongoDB veritabanına kaydedilmiştir. Fakat saf haliyle bu tweetler bazı sorunlar içermekteydi. Bunlar:

- Twitter'dan gelen çok şekilli tweet objeleri (Şekil 3.2)
- Tekrar eden tweetler (Retweetler)
- Tweet objelerindeki gereksiz öznitelikler
- Türkçe olmayan tweetler
- Yetişkin içeriğe sahip tweetler

Öncelikle tweet objelerinde bulunan gereksiz öznitelikler ve tekrar eden tweetler elenmiştir. Sonrasında ise bir MongoDB betiğiyle beraber bütün tweet objeleri tek şekil haline getirilmiştir. Yetişkin içeriğe sahip olan tweetler Twitter'ın sağladığı "possible_sensitive" özniteliğiyle beraber elenmiştir. Nihai olarak, yaklaşık olarak 25 milyon tweet kalmıştır. Bu tweetler bir bölütleyici ile beraber bileşenlerine ayrıştırılmıştır. Elde edilen sözcükler bozuk kelime kaynağı olarak kullanılmıştır.

3.1.2 Altyazı verileri

Düzgün yazılmış metin kaynağı olarak OPUS reposunda bulunan OpenSubtitles2018[1] [28] derlemi kullanılmıştır. Buradan elde edilen tekil altyazılar birleştirilmiş ve MongoDB veritabanına aktarılmıştır. Fakat altyazı derleminde de bazı sorunlar bulunmaktadır. Bunlar:

- Konuşulan dile yakınlığından dolayı altyazı derlemi seçilmiştir. Fakat bu derlemde bazen doğal olmayan cümlelere rastlamak mümkündür. Örnek olarak "Korkarım, onların yapmasını siz sağlamak zorundasınız." cümlesi verilebilir.
- Her ne kadar düzgün yazılmış metin olarak kabul etsek de altyazı derleminde konuşma dilinin direkt aktarımından dolayı oluşan yazım hataları barındırmaktadır (e.g. geliyooooooor).

[1] `http://www.opensubtitles.org/`

- Bunlar dışında çevirmenden kaynaklanan yazım hataları bulunmaktadır. Örnek olarak "Yemek cok guzel hafnendi." verilebilir.

3.2 Verilerin İşlenmesi

3.2.1 Modifiye edilmiş ağırlıklı Levenshtein algoritması

Levenshtein mesafesi, iki katar arasındaki farkı ölçmek kullanılan bir birimdir. Matematiksel olarak, [29] çalışmasında tanıtılmıştır. Katarlar kelime olarak kabul edilirse, bu ölçüt iki kelime arasındaki benzerliği en az harf ekleme, silme ve değiştirme sayısı olarak bulabilir. İki katar arasındaki Levenshtein mesafesi, katarlar a ile b ve katar uzunlukları $|a|$ ile $|b|$ olmak üzere, Formül 3.1 kullanılarak $lev_{a,b}(|a|,|b|)$ olarak hesaplanabilir.

$$lev_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{eğer } \min(i,j) = 0, \\ \min \begin{cases} lev_{a,b}(i-1,j) \ + \ 1 \\ lev_{a,b}(i,j-1) \ + \ 1 \\ lev_{a,b}(i-1,j-1) \ + \ 1_{(a_i \neq b_j)} \end{cases} & \text{geri kalan durumlar için} \end{cases} \tag{3.1}$$

Bu denklemde $1_{(a_i \neq b_j)}$ indikatör fonksiyon olarak görev yapmakta olup, $a_i = b_j$ olduğunda 0, $a_i \neq b_j$ olduğunda 1 değerini almaktadır. Teknik olarak $lev_{a,b}(i,j)$, a katarının ilk i tane karakteriyle, b katarının ile j tane karakteri arasındaki Levenshtein mesafesini vermektedir.

Tipik bir Levenshtein algoritmasında harf ekleme, silme ve değiştirme ağırlıkları 1 birim olarak kabul edilir. Bu tez kapsamında ise Levenshtein algoritmasının hafif değiştirilmiş versiyonu olan ağırlıklı Levenshtein kullanılmıştır. Bu algoritmayla beraber harf ekleme, silme ve değiştirme ağırlıkları sabit değil, harfe veya harf çiftine göre değişkenlik gösterebilmektedir. Karakterlere göre ağırlıkları belirleyen fonksiyonlar ekleme, silme ve değiştirme için sırasıyla *ins_cost*, *del_cost* ve *sub_cost*

olmak üzere, Formül 3.2'de ağırlıklı Levenshtein algoritması tanımlanmıştır.

$$lev_{a,b}(i,j) = \begin{cases} 0 & i = 0 \text{ ve } j = 0 \\ lev_{a,b}(i-1,\ j) \ +\ del\_cost(a_i) & j = 0 \text{ ve } i > 0 \\ lev_{a,b}(i,\ j-1) \ +\ ins\_cost(b_j) & i = 0 \text{ ve } j > 0 \\ \min \begin{cases} lev_{a,b}(i-1,\ j) \ +\ del\_cost(a_i) \\ lev_{a,b}(i,\ j-1) \ +\ ins\_cost(b_j) \\ lev_{a,b}(i-1,\ j-1) \ +\ sub\_cost(a_i,\ b_j) \end{cases} & \text{geri kalan durumlar} \end{cases} \tag{3.2}$$

Bu denklemde $sub\_cost(a_i,\ b_j)$ fonksiyonu önceki denklemde bulunan indikatör fonksiyonuna benzer bir işleve sahip olup $a_i = b_j$ olduğunda 0 değerine sahip olup, $a_i \neq b_j$ olduğunda a_i ve b_j harflerine atanan harf değişme ağırlığına göre değişen bir değere sahip olmaktadır.

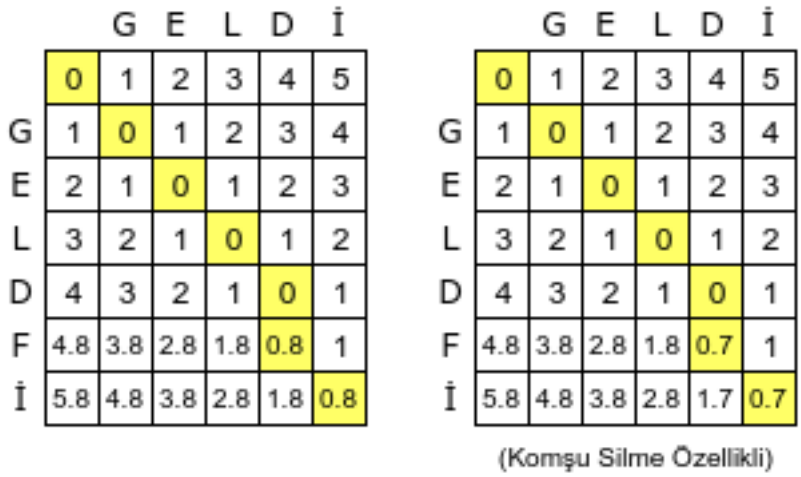
Bu özelliklere karşın, ağırlıklandırılmış Levenshtein algoritması ile Türkçe'de bulunan yazım hatalarının hepsini yakalamak mümkün değildir. Aynı zamanda, tez kapsamında algoritma büyük bir veri üzerinde çalışacağı için performansı iyileştirecek bazı eklemelerin yapılması gereklidir. Bunlardan yola çıkarak algoritmada aşağıdaki değişiklikler yapılmıştır:

- Komşu Harf Silme Ağırlığı
- Tekrar Eden Harf Ağırlığı
- Eşik Değerine Göre Durma

Bu değişikliklerin detayları sonraki bölümlerde anlatılmıştır.

3.2.1.1 Komşu harf silme ağırlığı

Çevrimiçi ortamlarda yapılan yazım hatalarından biri de klavyeden kaynaklanan yazım hataladır. Bu hataların başında, klavyeden bir harfe basılırken yakınında bulunan başka bir harfe de yanlışlıkla basılması gelmektedir. Örneğin, klavyede d harfi ile f harfi yanyana bulunmaktadır. Bu da, "geldi" yazılmaya çalışılırken, yanlışlıkla "geldfi" yazılması gibi hatalara sebep olabilmektedir. Dolayısıyla f harfinin yanlış yazılma olasığı kendisinden önce gelen harfe göre değişmektedir. Örneğin, f harfinden önce p harfi geliyorsa (klavyede görece uzak bir harf), f harfinin orada bulunması klavyedeki yakınlıktan ötürü doğan bir durum olarak görülemez. Dolayısıyla f harfinin, d veya p harfinden sonra gelmesini (veya önce gelmesini) farklı ağırlıklarla gösterebilecek

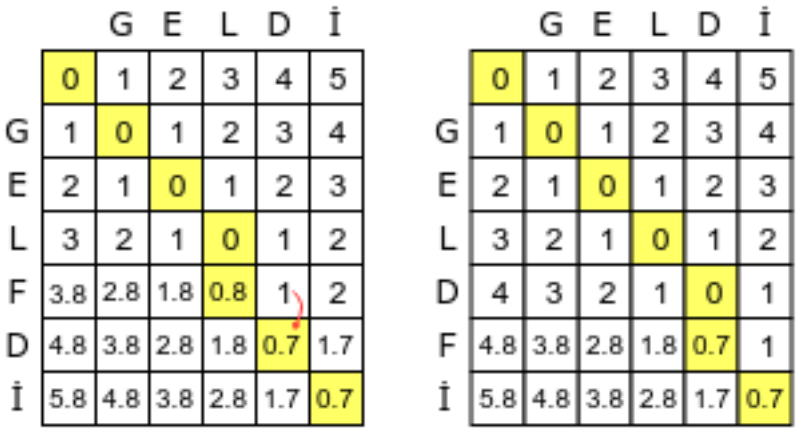


Şekil 3.3 : İki farklı Levenshtein algoritmasının çalışması.

bir ağırlık tipi tanımlanmasına ihtiyaç duyulmuştur. Bunu algoritmada hali hazırda bulunan tipik ağırlık tipleriyle yapmak mümkün değildir. Çünkü Levenshtein algoritması kelimeler üzerinde koşturulurken, kararlar hedef veya kaynak kelimedeki o anki harfe belirlenmektedir, kararın ne olacağı önceki harfe göre değişmemektedir. Örneğin, yukarıda bahsedilen "geldfi" yanlış yazımında f harfinin yanlış yazımının tespiti için, basitçe harf silme ağırlığının kullanıldığını varsayalım. Bu, d harfinin varlığını gözardı etmektedir. Bu durumu aşmak için Levenshtein algoritmasına *Komşu Harf Silme Ağırlığı* özelliği eklenmiştir. Parametre olarak f harfinin silinme ağırlığı 0.8 birim olan ve f harfinin silinme ağırlığı 0.8 birim ile d harfinin yanındaki f harfinin silinme ağırlığı 0.7 birim olan, iki Levenshtein algoritmasının sonuçları Şekil 3.3'te görülmektedir.

Bu özellik bu çalışma kapsamında klavye hatalarından doğan yazım hatalarını yakalamak için kullanılmıştır. Bu özellik "koşullu silme" olarak da düşünülenebilir. Dolayısıyla kullanımı sadece klavye hatalarını yakalamak için sınırlı değildir. Bu yeni özelliğin eklenmesiyle beraber değişen Levenshtein algoritmasının formülü 3.3, 3.4 ve 3.5'te gösterilmiştir.

$$lev_{a,b}(i,j) = \begin{cases} 0 & i = 0 \text{ ve } j = 0 \\ lev_{a,b}(i-1,\ j) + del\_cost(a_i) & j = 0 \text{ ve } i > 0 \\ lev_{a,b}(i,\ j-1) + ins\_cost(b_j) & i = 0 \text{ ve } j > 0 \\ \min \begin{cases} lev_{a,b}(i-1,\ j) + calc\_del\_cost(a_i,\ b_j,\ a_{i-1}) \\ lev_{a,b}(i,\ j-1) + ins\_cost(b_j) \\ lev_{a,b}(i-1,\ j-1) + sub\_cost(a_i,\ b_j) \end{cases} & \text{diğer durumlar} \end{cases} \tag{3.3}$$



Şekil 3.4 : Komşu silme özelliğinin iki durumu.

$$calc\_del\_cost\left(a_i,\, b_j,\, a_{i-1}\right) = \min\left(calc\_del\_adj\left(a_i,\, b_j,\, a_{i-1}\right),\, del\_cost\left(a_i\right)\right) \tag{3.4}$$

$$calc\_del\_adj\left(a_i,\, b_j,\, a_{i-1}\right) = \begin{cases} \infty & i-j \neq 1 \text{ olduğunda} \\ del\_adj\_cost\left(b_j,\, a_{i-1}\right) \;-\; sub\_cost\left(b_j,\, a_{i-1}\right) & i-j=1 \text{ ve } a_i = b_j \\ del\_adj\_cost\left(a_i,\, b_j\right) & \text{diğer durumlar} \end{cases} \tag{3.5}$$

Formül 3.3'ten de görülebileceği üzere silme ağırlığını hesaplaması basit bir fonksiyon olmaktan çıkıp *calc_del_cost* fonksiyonuna dönüşmüştür. Bu fonksiyon Formül 3.4'te tanımlanmıştır. Tanıma göre *calc_del_adj* ve *del_cost* fonksiyonlarının sonuçlarının minumumu nihai silme ağırlığı olarak kabul edilmiştir. Yine formülden de görüleceği üzere *calc_del_adj* fonksiyonu sadece i indisi ve j indisi arasında yalnızca 1 fark varken yani sadece komşu silme durumlarında çağrılmaktadır. Buna karşın, *del_cost* fonksiyon indisler arasındaki farkın 1 olduğu durum dahil, bütün durumlarda çağrılmaktadır.

Formü 3.5'te *calc_del_adj* fonksiyonunun iki adet duruma sahip olduğu görülmektedir. Bu durum komşu harfin kaynak harften önce veya sonra gelebilmesinden dolayı kaynaklanmaktadır. Örneğin, "geldfi" ve "gelfdi" kelimelerinde görüldüğü üzere komşu silme özelliğinin iki durumu bulunmaktadır. Şekil 3.4'te komşu harf silme özelliğinin iki durumu daha iyi anlaşılması için gösterilmiştir.

3.2.1.2 Tekrar eden harf ağırlığı

Çevrimiçi içeriklerde, kasti olarak yapılan bir dizi yazım hatası bulunmaktadır. Bu hataların en barizlerinden bir tanesi de pekiştirme için veya konuşma dilini olduğu gibi yazıya aktarmaktan dolayı ortaya çıkan belli bir harfin tekrar ederek yazılmasıyla oluşan ("oluuuuuuur" gibi) yazım hatalarıdır. Bu yazım hataları da saf Levenshtein algoritmasıyla tespit edilememektedir. Levenshtein algoritması sadece iki katar arasında karşılaştırma yaptığından dolayı, sadece bir kataın içinde oluşan durumları yakalayamamaktadır. Bunu aşmak için, Levenshtein algoritmasına aynı kataın içinde bulunan harfleri de işin içine katan *Tekrar Eden Harf Ağırlığı* özelliği eklenmiştir. Bu özellikle beraber güncellenen $calc\_del\_cost$ fonksiyonunun denklemi Formül 3.6'da gösterilmiştir.

$$calc\_del\_cost\left(a_i,\, b_j,\, a_{i-1}\right) = \min\left(calc\_del\_adj\left(a_i,\, b_j,\, a_{i-1}\right),\, calc\_del\_rep\left(a_i, a_{i-1}\right),\, del\_cost\left(a_i\right)\right) \tag{3.6}$$

$$calc\_del\_rep\left(a_i, a_{i-1}\right) = \begin{cases} del\_rep\_cost(a_i) & a_i = a_{i-1} \text{ olduğunda} \\ \infty & \text{diğer durumlar} \end{cases} \tag{3.7}$$

Denklemde de görüldüğü üzere, $calc\_del\_cost$ fonksiyonuna yeni bir denklem eklenmiştir. Bu denklem esas itibariyle sadece $a_i = a_{i-1}$ olduğunda, yani katardaki harf tekrar ettiğinde, çalışmakta ve del_rep_cost fonksiyonuna göre sonuç döndürmektedir.. Fonksiyonun geri kalan çalışma düzeninde bir değişiklik olmamıştır.

3.2.1.3 Eşik değerine göre durma

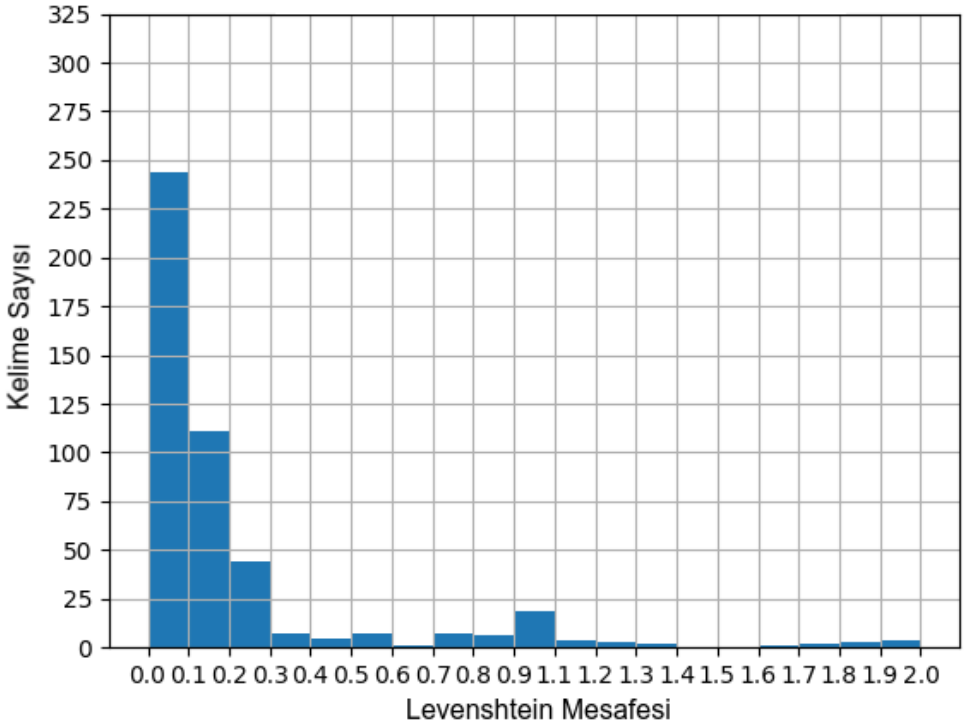
Levenshtein algoritması büyük veriler üzerinde koşturulduğunda, çalışma süresi çok uzun olabilmektedir. Bu çalışmada da, algoritma çok büyük veriler üzerinde koşturulacağı için, çalışma süresini azaltacak önlemlere ihtiyaç vardır. Bundan dolayı, algoritmaya *Eşik Değerine Göre Durma* özelliği eklenmiştir. Bu özellikle beraber, algoritma belli bir Levenshtein mesafesine ulaştığında otomatik olarak çalışmayı bitirmekte ve sonuç üretmemektedir. Bu da, halihazırda geçerli bir sonuç üretmeyecek kelime çiftlerinin karşılaştırmasının önceden bitirilmesini sağlamaktadır.

3.3 Paralel Veri Üretimi

Gerçekçi düzgün-bozuk çiftlerini içeren paralel veri üretmek için, bir önceki bölümde anlatılan Levenshtein algoritmasının modifiye edilmiş hali kullanılmıştır. Bu algoritmayı kullanarak, düzgün bir kelimeye karşılık gelen bozuk kelimeyi bulmak kolay bir görev değildir. Bu süreçte karşılaşılan bazı zorluklar şunlardır:

- Yaklaşık olarak 5 milyon adet bozuk kelimeyle, 1 milyondan fazla düzgün kelimeyi eşleştirmek için 5 trilyondan fazla karşılaştırılma yapılması gerekmektedir. Bir karşılaştırma ortalama olarak 0.001 saniye sürse bile, bütün karşılaştırılmaların bitmesi için 158 yıl gerekmektedir. Algoritma paralel hale getirilse bile, yeterince kısa zamanda bitmeyecektir.
- Levenshtein karşılaştırmasından sonra her düzgün kelime için bozuk aday kelime listesi oluşmaktadır. Fakat, örneğin "kalıyoruz" kelimesi için "kiliyorz" bozuk kelimesi aday olarak çıkabilmektedir. Bu tarz yanlış eşleşmelerin akıllı bir yöntemle elenmesi gerekmektedir.
- Her düzgün kelimeye karşılık gelen bozuk kelime sayısı eşit olmamaktadır. Örneğin, "ağlıyordu" kelimesi için "agliodu, ağlıyoru, ağliyoorduu, ağlıyorduuuu, ağliyordu" gibi aday bozuk kelimeler bulunmaktayken, "tahta" kelimesi için "thta, thtaaa, tahtta" gibi daha az sayıda aday bozuk kelime bulunmaktadır. Dolayısıyla, her düzgün kelime için aday listesinden 5 tane bozuk kelime seç gibi bir yol izlemek uygun olmayacaktır. Her düzgün kelime için, akıllı bir şekilde bozuk kelime adedi seçilmesi gerekmektedir.

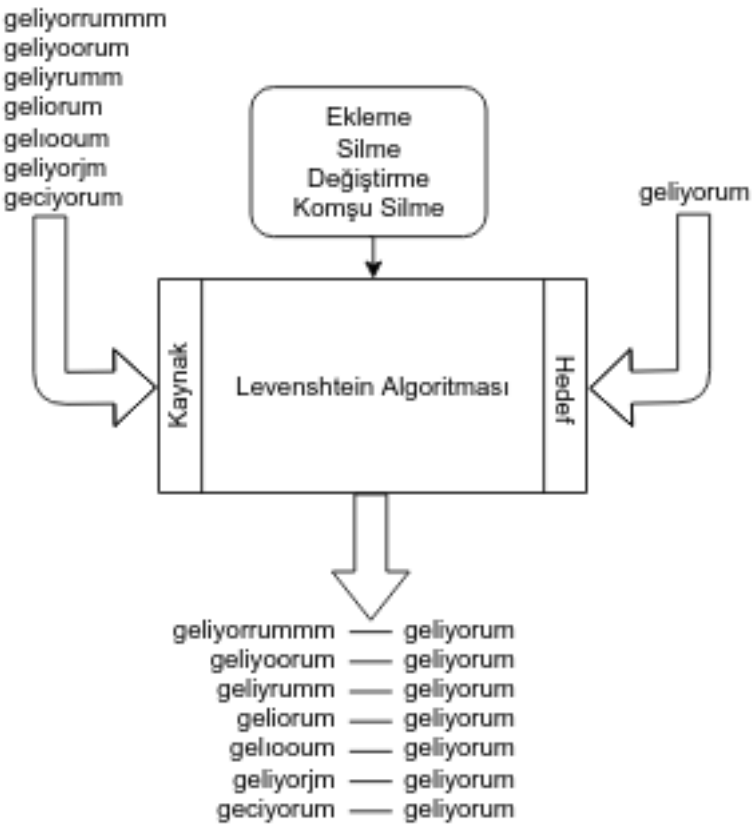
Bu zorlukların dışında çok sözcüklü öbeklerin yanlış yazılmasının yakalanması ve özel isimlerin yanlış yazılmasının yakalanması gibi durumlar ortaya çıkmıştır. Bu gibi hatalar rastlantısal kurallara göre bozma yöntemiyle simüle edilmiştir. Bundan sonraki bölümlerde bu iki yöntemi kullanarak nasıl veri üretildiği ve yukarıda bahsedilen zorlukların nasıl çözüldüğü anlatılacaktır.



Şekil 3.5 : Val_{Small} veri kümesindeki kelimelerin Levenshtein mesafesine göre histogramı.

3.3.1 Aday kelimelerin belirlenmesi

Değiştirilmiş ağırlıklı Levenshtein algoritması kullanılarak her düzgün kelime için aday bozuk kelimeler elde etmek için öncelikle Levenshtein algoritmasının parametrelerinin belirlenmesi gerekmektedir. Bunlar temel olarak, harflerin eklenme, silinme, değişme gibi ağırlıklarının belirlenmesi demektir. Bunlara ek olarak, bu tez kapsamında Levenshtein algoritmasına Türkçe yazım hatalarını daha iyi yakalamak için eklenmiş olan, komşu harf silme ve tekrar eden harf silme ağırlıklarının da belirlenmesi gerekmektedir. Bu ağırlıkları belirleme sürecinde, daha önce [2] çalışmasında tanıtılan Val_{Small} veri setinden yararlanılmıştır. Bu veri setindeki bozuk-düzgün kelime çiftleri incelenerek, sezgisel olarak harf ağırlıkları belirlenmiştir. Belirlenen ağırlıklara göre Levenshtein algoritması Val_{Small} veri kümesinde bulunan düzgün-bozuk kelime çiftleri üzerinde koşturulmuş, her bir çift için Levenhstein mesafesi değeri bulunmuştur. Bir koşma sonrasında ortaya çıkan örnek değerler, Şekil 3.5 histogramında gösterilmiştir. Histogramda altta bulunan değerler bozuk kelime ile düzgün kelime arasındaki uzaklık mesafesini belirtirken, soldaki değerler



Şekil 3.6 : Levenshtein algoritmasının çalışma şekli.

bu mesafede bulunan kaç kelime çifti olduğunu belirtmektedir. Bu histogram göz önünde bulundurularak belirlenen ağırlıklar üzerinde değişiklikler yapılmıştır ve en iyi ağırlık kombinasyonlarının bulunmasına çalışılmıştır. Histogramda görülebileceği üzere seçilen ağırlıklar neticesinde ortaya çıkan uzaklık değerlerine göre, bozuk kelimeler ile düzgün kelimeler arasında çoğunlukla 0 veya 0.1 uzaklık mesafesi korunmaya çalışılmıştır. Bu sayede, çoğunlukla yapılan yazım hatalarının bu mesafeye sıkıştırılması hedeflenmiştir. Fakat bütün kelime çiftleri bu uzaklık mesafesine sıkıştırılmamıştır. Val_{Small} kümesinde bulunan kelime çiftleri arasındaki uzaklığın, ufak bir uzaklık mesafesine sıkıştırılması, algoritmanın başka bir kümede çalıştırıldığında yalancı pozitif verme olasılığını artıracaktır. Bir başka deyişle, bu yolu izlememiz ağırlık seçimlerimiz genel hataları temsil etmekten ziyade sadece Val_{Small} kümesinde bulunan hataları temsil etmesine sebep olacaktır.

Şekil 3.6'te gösterildiği gibi Levenshtein algoritmasında kaynak tarafında bozuk kelimeler, hedef tarafında düzgün kelimeler bulunmaktadır. Algoritma da bulunan her bir ağırlık tipi belirli bir hata tipini yakalamak için kullanılmıştır. Bu bilgiler ışığında, harf ekleme ağırlığı düzgün kelimede eksik olan harfleri yakalamak için,

harf silme ağırlığı fazla olan harfleri yakalamak için, harf değiştirme ağırlığı birbirleri yerine kullanılan harf çiftlerini yakalamak için, komşu harf silme ağırlığı ise bir harfin yanına yanlışlıkla eklenen harfleri yakalamak için kullanılmıştır. Örneğin, harf ekleme ağırlığı, sosyal medyada sıklıkla yapılan hatalardan birisi olan eksik ünlü harflerin yakalamak için kullanılmıştır. Sadece ünlü harflerle kalmayıp, "r, h, ğ" gibi sıklıkla ihmal edilen harfler de bu ağırlığa dahil edilmiştir. Harf değiştirme ağırlığı ise birden fazla türde yazım hatalarını yakalamak için kullanılmıştır. Bunların arasında fonetik yazım hataları (s→ş, o→ö gibi), klavyedeki harf yakınlığından doğan yazım hataları ("tsm"→"tam" gibi) ve fonetik veya ortografik nedenlerle birbirine benzeyen harflerin birbirleri yerine kullanılmasından doğan yazım hataları (w→v, j→c, q→g gibi) bulunmaktadır. Harf silme ağırlığı fazla harf hatalarını yakalamak için ("tiren"→"tren") kullanırken, komşu harf silme ağırlığı ise yine klavyedeki harf yakınlığından doğan yazım hatalarını ("gelşiyor"→"geliyor") yakalamak için kullanılmıştır. Bunların dışında tekrar eden harf ağırlığı, kelimelerde tekrar eden harfleri yakalamak için ("oluuuur" gibi) ve harf yer değiştirme ağırlığı da yine klavyeden dolayı oluşan yazım hatalarını ("kalvye" gibi) yakalamak için kullanılmıştır. Diğer ağırlıklardan farklı olarak, tekrar eden harf ağırlığı tüm harfler için 0 ve harf yer değiştirme ağırlığı da her harf çifti için 1 olarak atanmıştır.

Çizelge 3.2 : Örnek Levenshtein ağırlıkları.

Ağırlık Tipi	Harfler/Harf Çiftleri	Değer	Anlam
Ekleme	a, e, ı, i, u, ü, o, ö	0.1	Eksik ünlü harfleri yakalama
Ekleme	r, y, ğ, h	0.1	İhmal edilen bazı ünsüzlerin yakalama
Değiştirme	s→ş, c→ç, g→ğ, u→ü	0.0	Fonetik Hataları Yakalama
Silme	ı, i	0.8	"sıtandart", "tiren"
Komşu Silme	r→e, r→d, r→f, r→t	0.9	Klavye hatalarını yakalama

Yukarıda anlatılan yazım hatası-ağırlık tipi ilişkisi göz önünde bulundurularak, paralel veri üretmek için Levenshtein algoritmasının ağırlıkları belirlenmiştir. Bazı harf ağırlıkları bütün kelime tipleri için ortak iken, bazı harf ağırlıkları sadece fiil veya isim kökenli kelimeler için kullanılmıştır. Örneğin, "a, e, ı, i, u, ü, o, ö" harflerinin ekleme ağırlıkları, "s→ş, c→ç, g→ğ, i→ı, o→ö, u→ü" harflerinin değişme ağırlıkları

bütün kelime tipleri için aynıyken, “k→z, i→e, i→a” harflerinin değişme ağırlıkları sadece fiiller için kullanılmıştır. Çizelge 3.2’de tez kapsamında kullanılan ağırlıkların bir kısmı, açıklamaları ile birlikte gösterilmiştir.

Daha önce bahsedildiği üzere, Levenshtein algoritmasını bu kadar büyük bir veri kümesinde çalıştırmak uzun bir zaman alacaktır. Algoritmanın çalışma süresini azaltmak için birçok önlem alınmıştır.

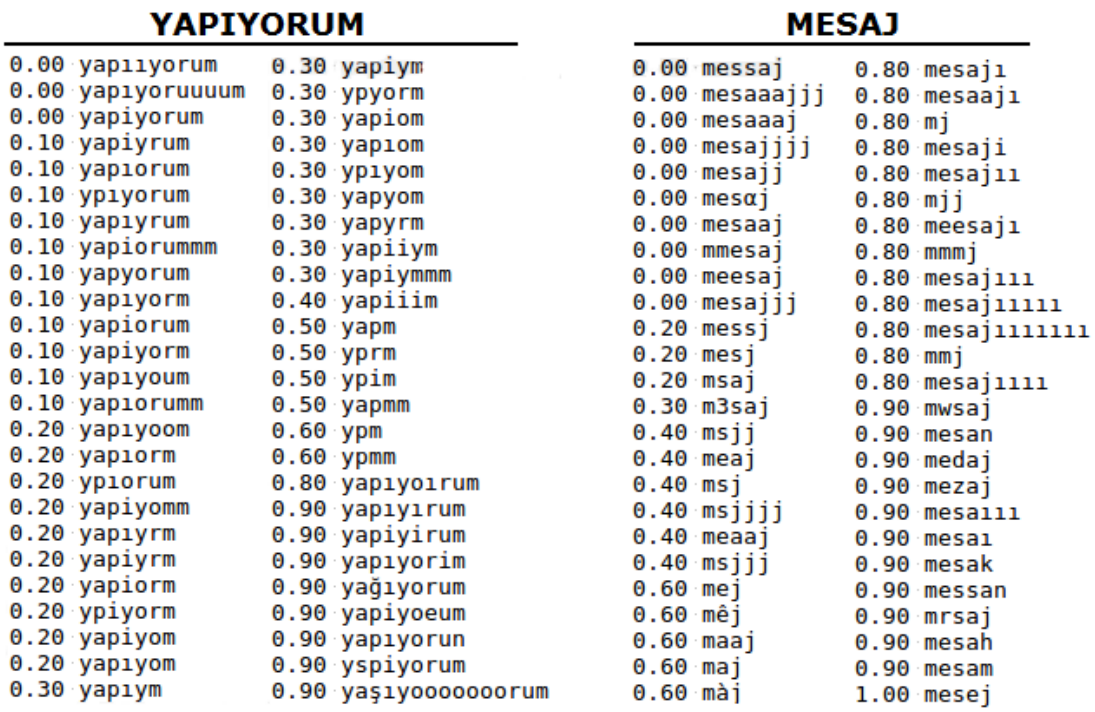
İlk olarak, kelimeler içinde karşılaştırma grupları oluşturulmuştur. Sadece aynı gruba ait kelimeler arasında karşılaştırma yapıldığından toplam karşılaştırma sayısı büyük ölçüde azalmıştır. Baş harfleri aynı veya ortografik olarak benzeyen kelimeler aynı karşılaştırma grubuna koyulmuştur. Bu gruplarda baş harfi “c veya ç”, “u veya ü” olan kelimeler aynı grupta yer almışlardır. Bu sayede baş harfi *t* olan bir harf ile baş harfi *p* olan bir kelimenin karşılaştırılmasının önüne geçilmiştir.

İkinci olarak alınan önlem ise kelimeler arası uzaklık için bir eşik değerinin belirlenmesidir. Birbirlerinden çok farklı iki kelime için algoritmanın çalışmaya devam etmesine gerek yoktur. Bu sebeple, varolan Levenshtein algoritmasına daha önceden de bahsedildiği gibi eşik değerine göre algoritmanın durması özelliği eklenmiştir. Bu sayede birbirinden çok farklı iki kelimenin (“güzel” ve “görüş” gibi) karşılaştırılması önceden sonlanabilecek ve bu da gereksiz karşılaştırmaları engellediği için algoritmanın çalışmasını hızlandıracaktır. Bu çalışmada, çizelge 3.2’de bulunan harf ağırlıkları gözüne alınarak Levenshtein algoritmasının eşik değeri 1 birim olarak seçilmiştir.

Son olarak algoritma paralel karşılaştırma yapabilecek hale getirilmiş ve karşılaştırma süresini kısaltmak için hafıza ve hız optimizasyonları uygulanmıştır. Sonrasında, algoritma Twitter ve altyazı derlemlerinden elde edilen kelimeler üzerinde koşturulmuştur. Şekil 3.7’te görülebileceği üzere her düzgün kelime için, bozuk aday kelime listesi elde edilmiştir.

3.3.2 Uygun aday kelimelerin seçilmesi

Bozuk aday kelime listeleri birçok yanlış eşleşme içermektedir. Bu yanlış eşleşmelerin tümünü elemek manüel kontrol gerektirdiğinden mümkün değildir. Fakat listeleri olduğu gibi bırakmak da çeviri sisteminin başarımını negatif olarak etkileyecektir.



YAPIYORUM			
0.00	yapııyorum	0.30	yapiym
0.00	yapıyoruuuum	0.30	ypyorm
0.00	yapiyorum	0.30	yapiom
0.10	yapiyrum	0.30	yapıom
0.10	yapıorum	0.30	ypıyom
0.10	ypıyorum	0.30	yapyom
0.10	yapıyrum	0.30	yapyrm
0.10	yapiorummm	0.30	yapiiym
0.10	yapyorum	0.30	yapiymmm
0.10	yapıyorm	0.40	yapiiim
0.10	yapiorum	0.50	yapm
0.10	yapiyorm	0.50	yprm
0.10	yapıyoum	0.50	ypim
0.10	yapıorumm	0.50	yapmm
0.20	yapıyoom	0.60	ypm
0.20	yapıorm	0.60	ypmm
0.20	ypıorum	0.80	yapıyoırum
0.20	yapiyomm	0.90	yapıyırum
0.20	yapıyrm	0.90	yapiyirum
0.20	yapiyrm	0.90	yapıyorim
0.20	yapiorm	0.90	yağıyorum
0.20	ypiyorm	0.90	yapiyoeum
0.20	yapiyom	0.90	yapıyorun
0.20	yapıyom	0.90	yspiyorum
0.30	yapıym	0.90	yaşıyooooooorum

MESAJ			
0.00	messaj	0.80	mesajı
0.00	mesaaajjj	0.80	mesaajı
0.00	mesaaaj	0.80	mj
0.00	mesajjjj	0.80	mesaji
0.00	mesajj	0.80	mesajıı
0.00	mesαj	0.80	mjj
0.00	mesaaj	0.80	meesajı
0.00	mmesaj	0.80	mmmj
0.00	meesaj	0.80	mesajııı
0.00	mesajjj	0.80	mesajııııı
0.20	messj	0.80	mesajııııııı
0.20	mesj	0.80	mmj
0.20	msaj	0.80	mesajıııı
0.30	m3saj	0.90	mwsaj
0.40	msjj	0.90	mesan
0.40	meaj	0.90	medaj
0.40	msj	0.90	mezaj
0.40	msjjjj	0.90	mesaııı
0.40	meaaj	0.90	mesaı
0.40	msjjj	0.90	mesak
0.60	mej	0.90	messan
0.60	mêj	0.90	mrsaj
0.60	maaj	0.90	mesah
0.60	maj	0.90	mesam
0.60	màj	1.00	mesej

Şekil 3.7 : "Yapıyorum" ve "mesaj" kelimeleri için örnek aday bozuk kelimeler.

Dolayısıyla çeviri sisteminin başarımını en az şekilde negatif ekleyecek, yanlış eşleşmeleri manüel kontrol gerektirmeden otomatik olarak olabildiğince eleyebilecek bir seçim yöntemi uygulanması gerekmektedir.

Yanlış eşleşme oranı, iki kelime arasındaki Levenshtein uzaklığı arttıkça yükselmektedir. Dolayısıyla, bozuk aday kelime listesinde düzgün kelimeye en yakın adayları (uzaklığı en az olan) seçmek, yanlış eşleşme oranını en çok azaltacağından bir seçim yöntemi olarak tercih edilebilir. Fakat bu yaklaşım sadece belli başlı hataları yakalayacağından, bu yöntemle oluşturulan veri kümesi yazım hatası çeşitliliğinin azlığından dolayı eğitilen sistemin zayıf performans göstermesine sebep olacaktır. Bu sebeple, hem çeşitliliği hem de yanlış eşleşme oranını azaltan bir seçim yöntemi kullanılmasına ihtiyaç vardır.

Seçim yöntemi olarak genetik algoritmalarda çokça kullanılan rulet çarkı seçim (Roulette wheel selection) yöntemi tercih edilmiştir. Rulet çarkı seçim yöntemi, genetik algoritmalarda bir popülasyondan bir bireyin uygunluk derecesine (fitness value) göre seçilmesi için kullanılan bir metottur. Naif bir yaklaşımla, aday listesinden kelime seçme probleminde "popülasyon" aday bozuk kelime listesine, "bireyler" her bir kelimeye ve "uygunluk derecesi" de Levenshtein uzaklığına denk olarak kabul edilebilir. Fakat, bir Levenshtein uzaklığında örneğin 0.6 uzaklığında

Çizelge 3.3 : Düzgün elimelere ait aday bozuk kelime sayıları.

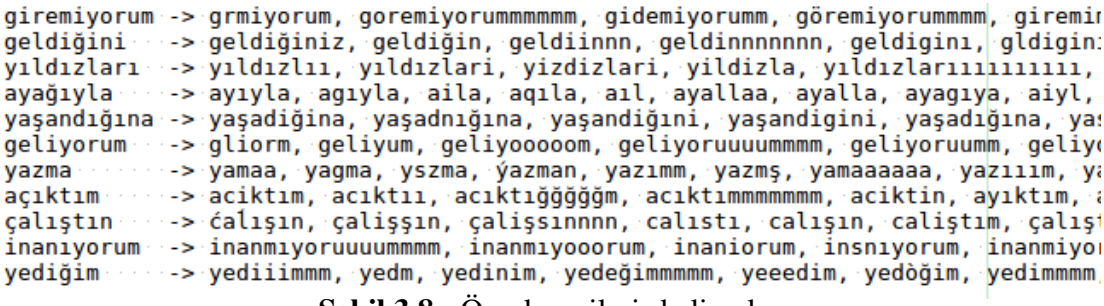
Örnek Kelime	Toplam Aday Sayısı	0.2 Lev. Mesafesine Kadarki Aday Sayısı
metin	92	9
bozuk	152	53
bozukluk	26	4
geliyorum	231	72
gidiyor	351	70
tahta	188	47
ziyarette	61	11

birden fazla kelime bulunabileceğinden bu kelimeler arasında uygunluk derecesine bakılarak ayrım yapılamayacaktır. Bu bilgiler göz önünde bulundurulduğunda, bu çalışmada rulet çarkı seçiminde kullanılacak bireyler olarak Levenshtein uzaklıkları seçilmiştir. Bir başka deyişle, rulet çarkı seçiminin sonucunda aday bozuk kelimeler yerine Levenshtein uzaklık değerleri seçilmiştir. Rulet çarkı seçimi gerçekleştikten sonra seçilen Levenshtein uzaklığında bulunan kelimeler arasında ise rastgele seçim yapılmıştır. Seçilen uzaklıkta herhangi bir kelime bulunmaması durumunda ise seçim tekrarlanmamış, yapılan seçim geçersiz sayılıp herhangi bir aday kelime seçilmemiştir. Uygunluk değerleri olarak, Val_{Small} derleminden elde edilen bilgiler kullanılmış, her Levenshtein uzaklığında bulunan kelime sayıları uygunluk derecesi olarak kabul edilmiştir. Bu değerler Çizelge 3.4'te gösterilmektedir.

Çizelge 3.4 : Uygunluk değerleri

Mesafe	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Uygunluk Derecesi	244	97	34	64	7	5	7	1	7	6	19

Bu süreci bir örnekle anlatmak istersek, öncelikle 0.0-1.0 aralığından Çizelge 3.4'te bulunan uygunluk derecesine bir sayı seçilir. Sonrasında belirlenen bir düzgün kelime için, bu örnek için "geliyorum" olsun, seçilen sayıya göre, bu örnek için 0.6 olsun, aday listesinden o mesafede bulunan bozuk kelimeler bulunur. Son olarak da bu kelimeler arasından rastgele bir seçim yapılır ve bu seçim yapay derlemde düzgün-bozuk kelime çifti olarak yerini alır. Eğer belirlenen mesafede hiç bozuk kelime yer almıyorsa hiçbir sonuç üretilmez.



Şekil 3.8 : Örnek seçilmiş kelimeler.

Bir düzgün kelimeye ait kaç tane bozuk yazılmış kelime seçilmesi gerekliliği uygun aday kelimelerin belirlenmesindeki başka bir problemdir. Daha önce de ifade edildiği üzere, her düzgün kelime için sabit sayıda bozuk kelime seçilmesi doğru bir yaklaşım olmayacaktır. Bu yaklaşım, Çizelge 3.3'te de görülebileceği gibi "geliyorum" gibi bir çok bozuk aday kelimeye sahip düzgün kelimelerdeki hataların yeterince temsil edilmemesine sebep olacaktır.

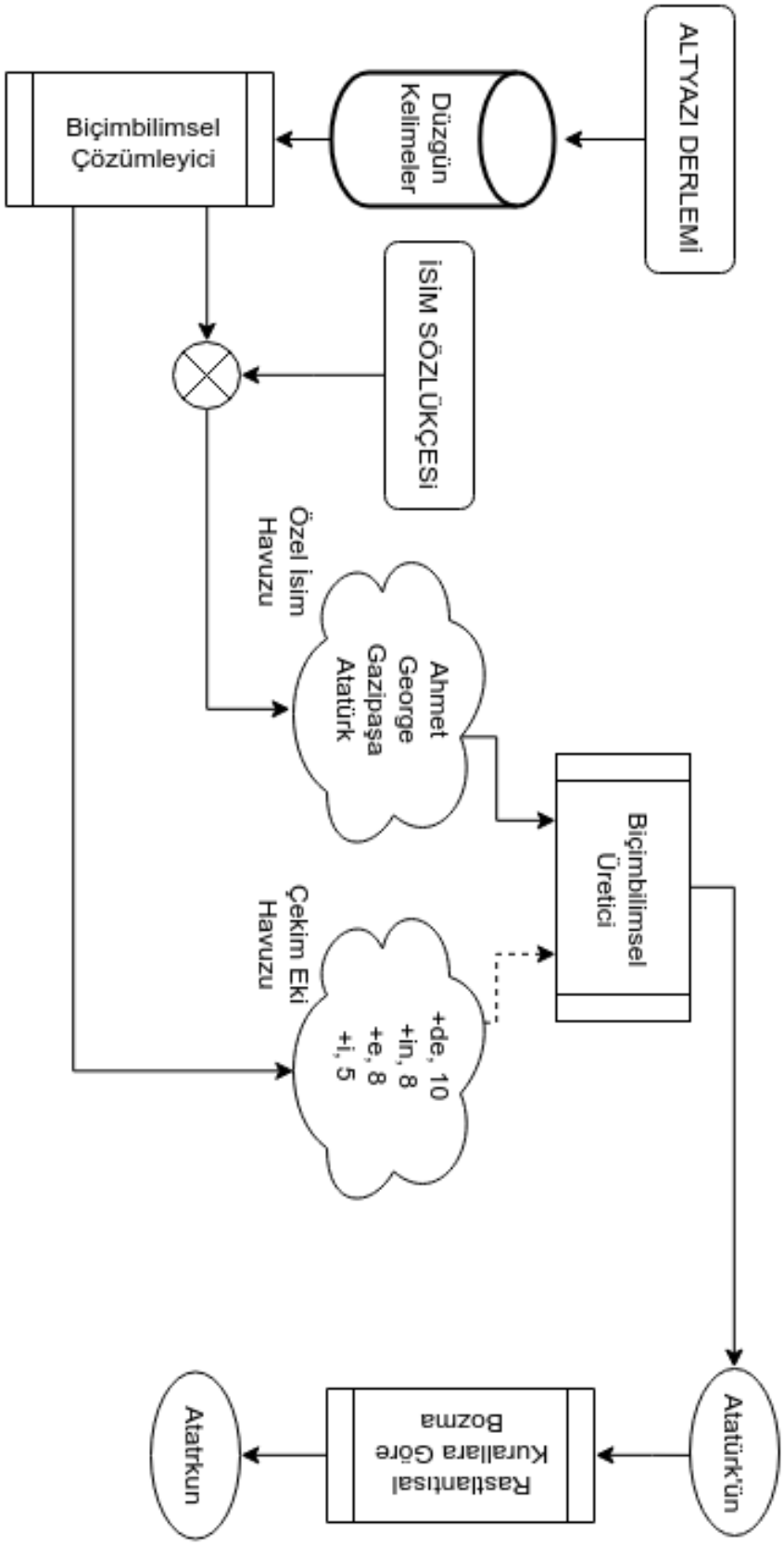
Bunun yerine, rulet çarkı seçim yöntemine oldukça benzeyen, yerine koymalı ağırlıklı rastgele örnekleme (weighted sampling with replacement) yöntemi kullanılmıştır. Bu yöntemi bir örnekle açıklamak gerekirse; 10 milyon sözcüklü bir derlem oluşturulmak istenildiğinde, düzgün kelime havuzundan daha önce belirlenen ağırlıklara göre düzgün kelime seçimi yapılacak, her ayrı seçimden sonra seçilen kelime tekrar havuza geri dönecek ve bu işlem 10 milyon kere devam edecektir. Bu yöntemle beraber, daha çok bozuk aday kelimeye sahip olan kelimelerin seçilme ihtimali arttığından dolayı derlemde diğer kelimelere göre daha çok temsil edilecektir. Fakat bu çalışmada, bir düzgün kelimenin seçilme ağırlığı olarak o düzgün kelimeye ait aday bozuk kelime listesinin boyutu seçilmemiştir. Bunun yerine, Levenshtein uzaklığı 0.2 birime kadar olan aday kelimelerin sayısı düzgün kelimenin seçilme ağırlığı olarak belirlenmiştir. Bu kararda, Levenshtein uzaklığı düzgün kelimeye az olan aday kelimelerin daha kaliteli sonuç vereceği düşüncesi etkili olmuştur. Düzgün kelimeler derlemi belirlendikten sonra, rulet çarkı yöntemiyle her kelimeye karşılık gelen bozuk kelimeler seçilmiştir. Bu işlem sonucundan ortaya çıkan paralel derleme ait örnekleme Şekil 3.8'de sunulmuştur. Şekilde de görülebileceği üzere, karmaşık bir aday seçme yöntemine rağmen hatalı eşleşmeler hala derlemde bulunmaktadır. Bu eşleşmeler, gürültü olarak kabul edilmiş olup, çeviri sistemine olabilecek negatif etkisi, karakter düzeyinde eğitilen dil modelleriyle minumuma indirilmeye çalışılmıştır.

3.3.3 Çok sözcüklü yazım hataları ve diğer hatalar

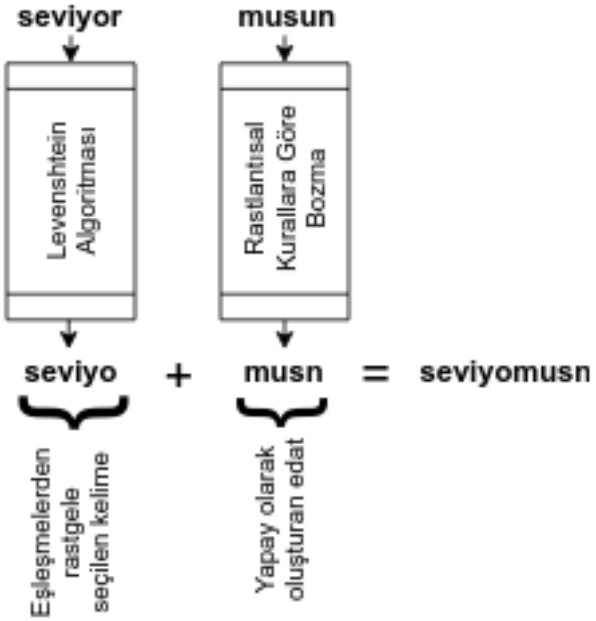
Bazı hata tiplerini Levenshtein algoritmasıyla yakalamak zor veya algoritmanın çalışma süresini oldukça uzatmaktadır. Bu gibi hata tiplerini paralel derlemde temsil etmek için rastlantısal kurallara bağlı veri üretme yöntemi tercih edilmiştir. Bu yöntem kullanılarak, özel isimlere gelen eklerin yanlış yazılması ve soru edatlarının yanlış yazılması ("dğilmydi?" gibi) gibi birden fazla sözcüğü kapsayan hatalar simüle edilmiştir. Bunların yanı sıra, Levenshtein algoritmasıyla yakalanması görece zor olan bazı aksan hataları da bazı kurallarla derleme eklenmiştir.

Öncelikle altyazı derleminde bulunan özel isimler çekim ekleriyle beraber biçimbilimsel çözümleyici kullanılarak tespit edilmiştir. Altyazı derlemi çeviri metinlerden oluştuğu için, bu derlemde varolan özel isimlerin birçoğu Türkçe'de nadir kullanılmaktadır. Bundan dolayı, Türkçe özel isimlerin bulunduğu sözlükçe (lexicon) de yapay veri üretmek için altyazı derleminde tespit edilen özel isimlere ek kaynak olarak kullanılmıştır. Derlemde tespit edilen özel isimlerin kökleri ve çekim ekleri biçimbilimsel çözümleyici ile birbirlerinden ayrılmış, çekim ekleri derlemde geçme sıklıkları ile birlikte bir çekim eki havuzuna aktarılmıştır. Sonrasında, her özel isim için çekim eki havuzundan ağırlıklı rastgele örnekleme yöntemi kullanılarak 4 adet çekim eki seçilmiştir. Biçimbilimsel çözümleyici ters çalıştırılarak özel isimler ile çekim ekleri birleştirilip, her özel isim için 4 adet yüzey formu (surface form) elde edilmiştir. Bu yüzey formları daha sonra rastlantısal kurallara bağlı bozularak, düzgün-bozuk kelime çiftleri elde edilmiştir. Bunlar kabaca, stokastik olarak ünlü silme, harf değişimi yapma (ç→c, ğ→g gibi) ve kesme işaretinin düşürülmesi gibi kurallardan oluşmaktadır. Bu kurallar tek tek veya kombinasyonu yüzey formuna uygulanarak bozuk kelimeler elde edilmiştir. Şekil 3.9'da bu algoritmanın akış şeması ve örnek bir özel ismin nasıl bozulduğu gösterilmiştir. Levenshtein algoritmasıyla oluşturulan verilerin aksine, bu veriler daha yapay bir görünüm sergilemektedirler.

Soru edatlarının yanlış yazılmasından dolayı ortaya çıkan hatalar için, özel isimler için kullanılan yöntemin çok benzeri bir yöntem kullanılmıştır. Öncelikle altyazı derleminde soru edatı olan sözcükler, kendilerinden bir önceki sözcüklerle beraber tespit edilmiştir. Fakat özel isimlerde uygulanan yöntemden farklı olarak,



Şekil 3.9 : Özel isimler için veri üretme şeması



Şekil 3.10 : Soru öbeklerinin oluşturulması

soru sözcüklerinin bozuk karşılıkları, Levenshtein algoritmasının çalıştırılmasının sonucunda ortaya çıkan aday listeden seçilmiştir. Ancak soru edatı kısmı için, özel isimlerde kullanılan rastlantısal kurallara bağlı bozma yöntemi kullanılmıştır. Sonrasında ise, soru sözcükleri ve soru edatları arada boşluk olmayacak şekilde birleştirilmiştir. Bu sürecin akış şeması ve örnek bir soru öbeğinin oluşması Şekil 3.10'da görülmektedir.

4. YÖNTEM

Metin normalleştirme problemi özünde birbiriyle yakından ilişkili iki dil arasında çeviri problemi olarak ele alınabilir. Bu çalışmada, bozuk yazılmış Türkçe metin bir dil olarak kabul edilip bu dilden düzgün yazılmış dile makine çevirisi yapılarak normalleştirme probleminin çözülmesi hedeflendirmiştir. Bu amaçla, 3 bileşenden oluşan bir makine çevirisi işhattı modeli tasarlanmıştır. Türkçe metin normalleştirme problemini daha iyi anlamak için öncelikle daha önce [2]'de kategorize edilmiş hata tipleri ve onlara karşılık gelen örnekler verilecektir. Sonrasında ise bu çalışmada tanıtılan veri odaklı (data-driven) çeviri modelinin detayları paylaşılacaktır.

4.1 Hata Tipleri

4.1.1 Büyük-küçük harf hataları

Bu tarz hatalar genellikle metnin bütünüyle büyük harften oluşmasından, cümleye küçük harf ile başlanmasından veya özel isimlerin küçük harf ile başlamasından kaynaklanan hatalardır. Örnek olarak, "ANLAMADIM?", "emine", "IOS" gibi yazımlar verilebilir.

4.1.2 Ortografik Hatalar ve Kısaltmalar

Başka alfabelerde bulunan fakat Türkçe'de bulunmayan harflerin Türkçe yazılırken kullanılmasına ortografik hatalar denir. Bu harfler genellikle Türkçe'de başka bir harfe benzemekle beraber, resmi olarak Türkçe'de bulunmamaktadır. Bunlara örnek olarak 'ß'→'b', 'ä'→'a' ve 'ø'→'o' verilebilir.

Günlük dilde kullanılan bazı deyişler internet ortamında sıklıkla kısaltıralarak yazılmaktadır. Bu tarz kısaltmalara örnek olarak 'kib'→'Kendine İyi Bak', 'aeo'→'Allah'a Emanet Ol' verilebilir.

4.1.3 Fonetik işaret hataları

Türk Alfabesinde bulunan (çşğöü) fakat diğer Latin tabanlı alfabelerde bulunmayan bazı harfler vardır. Birçok zaman çeşitli nedenlerden ötürü bu harflerin yerine, bu harflere çok benzeyen latin harf karşılıkları çevrimiçi içeriklerde kullanılır. Örnek olarak, "yogurt"→"yoğurt", "Turkce"→"Türkçe" verilebilir.

4.1.4 Eksik ünlü harf yazılması

İnternet sitelerinde bulunan harf kısıtından ötürü veya hızlı yazmak için bazen kelimelerde bulunan ünlü harfler ihmal edilir. Bunlara örnek olarak "çk"→"çok", "gliyrm"→"geliyorum" verilebilir.

4.1.5 Aksan hataları

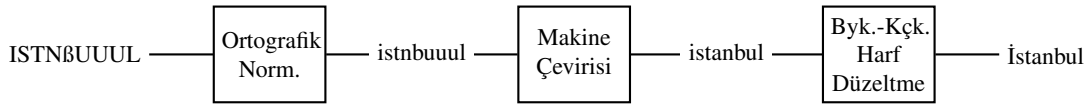
Konuşulduğu gibi yazımdan ötürü ortaya çıkan hatalara aksan hataları denir. Bunlara örnek olarak "yapcam"→"yapacağım", "görüşcez"→"görüşeceğiz" verilebilir.

4.1.6 Yazım hataları

Klavye hatalarından doğan veya kelimenin yazımının yanlış bilinmesinden dolayı ortaya çıkan hatalara yazım hataları denir. Bunlara örnek olarak "gedlim"→"geldim" verilebilir.

4.2 Önerilen Çeviri Sistemi

Önerilen çeviri işhattı 3 ana bileşenden oluşup, çeviri bileşeninin yanısıra bir ön ve bir son işleme bileşeninden oluşmaktadır. Ön işleme bileşeni, harf seviyesinde temel bir normalizasyon sağlamakla beraber (ø→o gibi), aynı zamanda bütün harfleri küçük harfe çevirmektedir. Bu işlem kelime hazinesini (vocabulary) azalttığından dolayı çeviri işleminin karmaşıklığını azaltmaktadır. Bunun yanı sıra, son işleme bileşeni, çeviri bileşeninin sadece küçük harflerden oluşan çıktısını alarak büyük-küçük harf düzenini tekrardan sağlamaktadır. Şekil 4.1'de işhattının bileşenleri ve örnek bir



Şekil 4.1 : Makine çevirisi yönteminin bileşenleri ve akış şeması.

kelimenin hangi aşamalardan geçtiği gösterilmiştir. Bu bileşenlerin detaylarına aşağıda yer verilmiştir.

4.2.1 Ortografik normalleştirme

Bazen sosyal medya kullanıcıları, μ karakteri gibi, Türkçe *u* karakterine benzeyen fakat Türk alfabesinde bulunmayan karakterleri kullanmayı tercih ederler. Kelime haznesini (vocabulary size) azaltmak için, bu bileşen, Türkçe olmayan karakterlerin otomatik olarak normalleştirilmesini sağlar. Aynı zamanda, bunun yanı sıra kelimede bulunan bütün harfleri küçük harfe dönüştürür.

4.2.2 Makine çevirisi

Bu bileşen, bir makine çeviri sistemi kullanarak önceden işlenmiş veri üzerinde normalleştirme işlemini gerçekleştirir. Çeviri bileşeni, diğer bileşenlerden bağımsız olmakla beraber, sistemin performansı tamamen kullanılan çeviri sisteminin türüne bağlı olarak değişiklik gösterebilmektedir.

4.2.3 Büyük-küçük harf kurtarma

Daha önce de vurgulandığı üzere, normalleştirme işlemi küçük harfler üzerinde gerçekleşmektedir. Bu bileşen, çeviri sisteminin küçük harfli kelime çıktısını alarak, bağlamı da göz önünde bulundurarak (cümle başı veya özel isimlerin ilk harfinin büyük olması gibi) küçük-büyük harf kurtarması gerçekleştirmektedir.

5. DENEYSEL SONUÇLAR

Bu çalışma kapsamında Bölüm 3'te makine çevirisi için paralel veri gerekliliğinden bahsedilmiştir. Bu bölümde öncelikle bu üretilen paralel veri, diğer eğitim verileri ve sınama kümelerinden bahsedilecek, sonrasında ise önerilen sistemin parametre detayları ve sınama kümeleri üzerindeki sonuçları verilecektir.

5.1 Veri Kümeleri

5.1.1 Eğitim kümeleri

OpenSubs$_{Filtered}$ OPUS deposunda bulunan, serbestçe ulaşılabilir büyük metin derlemi olan OpenSubtitles2018'den[1] [28] Türkçe altyazılar çekilmiştir. OpenSubtitles verisi görece gürültülü olduğundan, bu veri biçimbilimsel çözümleyici [30] kullanılarak filtre edilmiştir. Bu filtreleme sonucunda, her kelimesi biçimbilimsel çözümleyiciden geçen cümleler tutulmuş, geri kalan cümleler elenmiştir. Sonuç olarak yaklaşık 105 milyon cümle ve 535 milyon sözcük elde edilmiştir. Bu veri, paralel veri üretiminde ve dil modeli eğitiminde kullanılmıştır.

Train$_{ParaSent}$ Bu çalışma kapsamında OpenSubs$_{Filtered}$ ve Train$_{ParaTok}$ kullanılarak üretilen derlemdir. Val$_{Small}$ derleminden elde edilen bilgiye göre derlemde bulunan sözcüklerin %20'si yanlış olarak yazılmışlardır. Bu bilgiden yola çıkılarak OpenSubs$_{Filtered}$ derleminde bulunan düzgün kelimelerin yaklaşık olarak %20'sinin bozuk karşılıkları Train$_{ParaTok}$ derleminden bulunup değiştirilmiştir. Bu işlem sonucunda bağlam korunarak düzgün-bozuk cümle çiftlerini içeren paralel derlem elde edilmiştir.

Train$_{ParaTok}$ Bölüm 3'te anlatılan yöntemlerle üretilen veri kümesidir. Bu veri kümesi kullanılarak makine çevirisi sistemleri eğitilmiştir.

TWNERTC Ek standart metin kaynağı olarak, büyük Türkçe derlemi olan [31] kullanılmıştır. Bu kaynak, 54 milyon cümle ve 968 milyon sözcükten oluşmaktadır. Bu derlem e-kitap, haber siteleri ve film inceleme siteleri gibi birçok farklı kaynaktan

[1] `http://www.opensubtitles.org/`

gelen metinlerden oluşmaktadır. Bu derlemi, dil modeline ek derlem olarak ve büyük-küçük harf kurtarıcısı eğitiminde kullanılmıştır.

5.1.2 Sınama ve doğrulama kümeleri

Test$_{IWT}$ ITU Web Ağaç Yapılı derlemi [32], 4,842 elle düzeltilmiş ve etiketlenmiş cümle ve 38,917 sözcük bulundurmaktadır. Bu derlem, çeşitliliği sağlamak amacıyla birçok internet sitesi içeriğinden oluşturulmuştur. Bu küme, daha önce yapılan çalışmalarla [3] karşılaştırmak için kullanılmıştır.

Çizelge 5.1 : Sınama kümesi büyüklükleri.

Veri Kümeleri	# Sözcükler	# Kuralsız Sözcükler
Test$_{IWT}$	38,917	5,639 (14.5%)
Test$_{2019}$	7,948	2,856 (35.9%)
Test$_{Small}$	6,507	1,171 (17.9%)

Test$_{Small}$ [2] çalışmasında tanıtılan bu veri kümesi, 509 cümle ve 6,507 sözcük içermektedir. Bu küme, daha sonraki çalışmalarda olduğu gibi [3, 4], bu çalışmada da sınama kümesi olarak kullanılmıştır.

Test$_{2019}$ Bu veri kümesi, Twitter'dan elde edilen 713 tweet'ten ve 7,948 sözcükten oluşmaktadır. Bu küme, bu çalışma için elle düzeltilmiş ve etiketlenmiştir.

Val$_{Small}$ [2] çalışmasında tanıtılan, 600 cümle ve 7,061 sözcükten oluşan bu veri kümesi istatiksel makine çevirisi sistemi ve veri üretmek sistemleri için doğrulama kümesi olarak kullanılmıştır.

5.2 Tahminleme Modeli Detayları ve Sonuçları

Bölüm 4'te anlatıldığı üzere, önerilen çeviri iş hatta 3 ana bileşenden oluşmaktadır. Sistemimizin ilk bileşeni, ortografik normalleştirici, basit bir kural tabanlı karakter değiştirme (character replacement) sistemidir. Sistemi oluşturmak için öncelikle Train$_{ParaTok}$ derlemini oluşturmak için elde ettiğimiz Twitter derleminde bulunan bütün tekil karakterler toplanmıştır. Bu karakterler arasında, Türkçe olmayan tweetlerden ötürü, Arapça, Farsça ve Japonca gibi, ortografik olarak Türkçe karakterlere çevrilemeyen karakterler bulunmaktadır. Bu karakterler, *Unicode* veri tabanında bulunan karakter isimleri kullanılarak elenmiştir. Bu eleme sonucunda,

Çizelge 5.2 : Ortografik normalleştiricide yer alan bazı harf çiftleri (*Unicode* isimleriyle beraber).

Orijinal Harf	Normalleştirilmiş Harf
ê (Latin Small Letter E With Circumflex)	e (Latin Small Letter E)
ï (Latin Small Letter I With Diaeresis)	i (Latin Small Letter I)
é (Latin Small Letter E With Acute)	e (Latin Small Letter E)
û (Latin Small Letter U With Circumflex)	u (Latin Small Letter U)
ý (Latin Small Letter Y With Acute)	y (Latin Small Letter Y)
ä (Latin Small Letter A With Diaeresis)	a (Latin Small Letter A)
ș (Latin Small Letter S With Comma Below)	s (Latin Small Letter S)
í (Latin Small Letter I With Acute)	i (Latin Small Letter I)
ã (Latin Small Letter A With Tilde)	a (Latin Small Letter A)
ó (Latin Small Letter O With Acute)	o (Latin Small Letter O)
á (Latin Small Letter A With Acute)	a (Latin Small Letter A)
ì (Latin Small Letter I With Grave)	i (Latin Small Letter I)
ñ (Latin Small Letter N With Tilde)	n (Latin Small Letter N)

yalnızca Latin, Yunan ve Kiril alfabelerine ait karakterler bırakılmıştır. Son olarak, eleme sonucundan geriye kalan karakterler için, ortografik benzerlikleri yakalayan bir Pyhton kütüphanesi yardımıyla latin karakter karşılıkları bulunmuştur. Kütüphanede bulunan bazı hatalar veya istenilen karşı karakterin bulunmamasından ötürü, bu bulunan latin karşılıklar manüel olarak incelenmiş, bazıları silinmiş, bazıları ise düzeltmiştir. Bu işlem sonucunda, 701 adet latin-latin olmayan karakter çifti elde edilmiştir. Bu harf çiftlerinin bazıları çizelge **??**'de görülebilir.

Çeviri bileşeni olarak istatiksel makine çevirisi ve yapay sinir ağı temelli makine çevirisi yöntemleri denenmiştir. Bu çalışmadaki ana amaç, Türkçe metin normalleştirme problemini makine çevirisi yöntemleriyle, yapay üretilen veriyle bile çözülebileceğini göstermek olduğundan, bu iki yöntem için de model parametreleri için ince ayar yapılmamış, daha önceden bilinen ayarlar veya varsayılan ayarlar kullanılmıştır. Bu bilgiler ışığında, istatiksel makine çevirisi işhattı için literatürde halihazırda bilinen, standartlaşmış bir dizi araç kullanılmıştır. Bu iş hattında yer alan araçların parametreleri [33] ve [25] çalışmalarında bulunan değerlere benzer şekilde atanmıştır. Hizalama (alignment) için MGIZA [34] aracı kullanılmış, simetri yöntemi olarak *grow-diag-final-and symmetrization* seçilmiştir. Dil modellemesi için, KenLM [35] dil modelleyici aracı, karakter seviyesinde 6-gram dil modeli eğitiminde kullanılmıştır. Bu eğitimde $\text{OpenSubs}_{Filtered}$ ve TWNERTC derlemleri kullanılmıştır.

Çizelge 5.3 : Büyük/küçük harf duyarlı ölçümler (altta), Büyük/küçük harf duyarsız ölçümler (üstte) (Bütün sözcükler üzerinde).

Model	Test_{IWT}	Test_{2019}	Test_{Small}
Eryiğit et al. (2017)	95.78%	80.25%	92.97%
	93.57%	75.39%	86.20%
SMT	**96.98**%	**85.23**%	**93.52**%
	95.21%	**78.10**%	**89.59**%
NMT	93.90%	74.04%	89.52%
	92.20%	67.87%	85.77%

Çizelge 5.4 : Büyük/küçük harf duyarlı ölçümler (altta), Büyük/küçük harf duyarsız ölçümler (üstte) (Muğlak sözcükler üzerinde)

Model	Test_{IWT}	Test_{2019}	Test_{Small}
Eryiğit et al. (2017)	79.16%	66.18%	74.72%
	70.54%	56.44%	53.80%
SMT	**87.43**%	**74.02**%	**76.00**%
	84.70%	**66.35**%	**68.40**%
NMT	71.34%	50.84%	58.67%
	68.91%	45.03%	51.84%

Öbek çıkarımı (phrase extraction) ve kodlama (decoding) için, Moses [36] aracı kullanılmıştır. Bu araçla beraber $\text{Train}_{ParaTok}$ derlemi üzerinde eğitimler yapılmıştır. Çeviri parametresi olarak distorsiyon iptal edilmiştir. Asgari hata oranı eğitimi için (minimum error rate training), Val_{Small} derlemi kullanılmıştır.

Yapay sinir ağı temelli makine çeviri modeli, *OpenNMT* aracı kullanılarak $\text{Train}_{ParaTok}$ üzerinde herhangi bir parametre ayarlaması yapılmadan varsayılan ayarlarla eğitilmiştir. Model *attentional* kodlayıcı-kod çözücü yapısında olup, çift katmanlı LSTM kodlayıcısı ve koz çözücüsünden oluşmaktadır. Girdi embeddingleri, kodlayıcı LSTM'in katmanları ve kod çözücü LSTM'in iç katmanı 500 birim boyuttundadır. Kod çözücü LSTM'in dış katmanı ise 1000 birim boyutundadır. Kodlayıcı ve kod çözücü LSTM'ler 0.3 oranında seyreltmeye (dropout) sahiptir.

Metin normalleştirme iş hattının son bileşeni için, büyük-küçük harf kurtarma modeli eğitilmiştir. Bu model, Moses [36] aracında bulunan *recaser* bileşeni kullanılarak TWNERTC üzerinde eğitilmiştir. Eğitim verisi olarak $\text{OpenSubs}_{Filtered}$ yerine, TWNERTC verisinin kullanılması daha önce de belirtilen bir sorundan dolayı kaynaklanmaktadır. $\text{OpenSubs}_{Filtered}$ çeviri bazlı bir derlem olduğunda çoğunlukla yabancı yer isimleri ve özel isimler içermektedir. Fakat TWNERTC çeşitli web

Çizelge 5.5 : MN modellerinin sınama kümeleri üzerinde yaptığı tahminler (Düzgün sözcükler üzerinde üstte, bozuk sözcükler üzerinde altta)

Model	Veri Kümesi	#Doğru Tahmin	#Yanlış Tahmin	Oran
SMT	Test_{IWT}	26,962	732	97.36%
		4,776	863	84.70%
	Test_{Small}	3,951	182	95.60%
		801	370	68.40%
	Test_{2019}	2,966	402	88.06%
		1,895	961	66.35%
NMT	Test_{IWT}	26,842	847	96.94%
		3,886	1753	68.91%
	Test_{Small}	3,942	191	95.38%
		607	564	51.84%
	Test_{2019}	2,938	430	87.23%
		1,286	1570	45.03%

sitelerinden oluşturulduğu için hem yabancı hem de yer isimleri yeterince temsil edebilmektedir. Bu da doğal olarak özel isimleri yakalayıp, ilk harfini büyük yapmadaki oranı artıracaktır. Bu bileşenin başarımı diğer bileşenlerden bağımsız olarak ölçülmemiştir. Küçük harfli ve tam harfli (fully-case) çeviri çıktıları, bu çıktılara karşılık gelen kesin referanslarla (ground truth) ile karşılaştırılmıştır. Bu karşılaştırma sonucunda, büyük/küçük harf duyarlı ve büyük/küçük harf duyarsız sonuçlar elde edilmiştir. Bu sonuçlar çizelge 5.3 ve 5.4'te görülebilir[2]. Bunlara ek olarak modellerin düzgün kelimeler üzerinde (düzeltme gerektirmeyen kelimeler) ve bozuk kelimeler üzerindeki (muğlak kelimeler) sonuçları da çizelge 5.5'te verilmiştir. Modellerin ürettiği bazı örnek hatalı tahminler de çizelge 5.6'da görülebilir.

URL, sosyal medya etiketleri (hashtags), bahsetmeler (mentions), anahtar kelimeler, emoji gibi özel sözcükler doğruluk hesabına katılmamıştır[3]. Tablolarda gözlenen sonuçlar sözcük tabanlı sonuçlar olup, şu ana kadar ki en iyi sonucu üreten sistem olan [3] ile karşılaştırılmıştır. Karşılaştırmada, önceki bölümde bahsedilen sınama kümeleri kullanılmıştır. [3] çalışmasında kullanılan *Big Twitter Set (BTS)* veri kümesi açık erişimli olmadığı için bu çalışmada kullanılamamıştır.

[2] [3] çalışmasında Test_{IWT} ve Test_{Small} veri kümelerinde 67.37% ve 81.76% olarak verilen doğruluk oranları bu çalışmada sırasıyla 70.54% ve 53.80% olarak hesaplanmıştır.

[3] Daha önce [3] çalışmasında raporlanan sonuçlar ile tekrardan üretilen sonuçlar arasındaki farklılığın büyük bir kısmı buna dayanmaktadır. Ayrıca bu çalışmada bütün harfleri büyük olan sözcükler de (önceki çalışmadan farklı olarak) doğruluk hesabına dahil edilmiştir.

Çizelge 5.6 : Bazı tahmin hataları.

Orijinal	Yanlış Tahmin	Doğrusu
hd	hadi	HD
hz	hız	Hz.
allahı	Allahı	Allah'ı
nagihanı	nagihanı	Nagihan'ı
Türkçeye	Türkçeye	Türkçe'ye
öpücem	ölücem	öpeceğim
görücez	görücez	göreceğiz
denicem	denicem	deneyeceğim

Sonuçlara göre, bu tez kapsamında önerilen karakter tabanlı istatiksel makine çevirisi yöntemi, bu konudaki mevcut en iyi çalışma olan [3]'yi yüksek farklarla geçtiği görülmüştür. Bu sonuçlar göstermektedir ki gözetimsiz paralel veri üretme yöntemlerimiz ile makine çevirisi yöntemi Türkçe metin normalleştirme problemini çözmede oldukça yüksek başarım göstermektedir.

6. SONUÇ VE ÖNERİLER

Bu tez çalışması kapsamında, Türkçe metin normalleştirme probleminin makine çevirisi yaklaşımları kullanırak çözülmesi önerilmiştir. Bu konudaki önceki en iyi çalışmaya göre, önerdiğimiz sistem daha az bileşen ile, bağlamı da işin içine katarak problemi çözebilmektedir. Ayrıca sistem kural tabanlı bileşen içermediğinden insan hatalarından daha az etkilenmektedir. Bunların yanı sıra önerdiğimiz sistem harici dil araçlarına gerek duymamaktadır.

Makine çevirisi sistemlerini eğitmek için paralel veriye ihtiyaç vardır. Metin normalleştirme problemi için yeterli işaretlenmiş veri bulunmamasından ötürü yapay olarak veri oluşturulmuştur. Yapay veri oluşturmak için Levenshtein algoritması ve rastlantısal kurallara göre kelime bozma yöntemleri kullanılmıştır. Levenshtein algoritmasının bazı yazım hatalarını daha iyi yakalayabilmesi için bu çalışma kapsamında algoritmaya eklemeler yapılmış, hız optimizasyonları uygulanmıştır. Bu modifiye edilmiş algoritma kullanılarak bozuk kelimeler ile düzgün kelimeler eşleştirilmiştir. Bu sayede gerçekçi yapay veri oluşturulması hedeflenmiştir. Eşleştirmeler sonucunda ortaya çıkan aday listesinden akıllı seçim yöntemleri kullanılarak nihai yapay paralel veri ortaya çıkarılmıştır.

Karakter tabanlı istatiksel makine çevirisi yöntemi ve ona ek iki bileşen ile yapay olarak oluşturulan verinin üzerinde yapılan eğitimler göstermiştir ki, bu tez kapsamında önerilen sistem önceki en iyi çalışmayı yüksek farklarla geçmiştir. Sistemin ilk bileşeni kural tabanlı karakter normalizasyonu yapıp kelime hazinesini daraltırken, son bileşen bağlamı da gözeterek büyük-küçük harf kurtarması yapmaktadır. Yapılan incelemelerde önerilen sisteminin çoğunlukla kısaltmalarda, bazı aksan hatalarında ve özel isimlere gelen eklerde hata yaptığı görülmüştür.

Gelecek çalışmalarda, yapay veri oluşturulurken kelime vektörlerinin de kullanılarak paralel verinin kalitesinin artması hedeflenmektedir. Buna ek olarak, yapay sinir

ağı temelli makine çevirisi yönteminde karakterden üretilen kelime vektörlerinin kullanılmasının performansa olabilecek pozitif etkisi incelenebilir.

KAYNAKLAR

[1] **Xie, Z.**, **Genthial, G.**, **Xie, S.**, **Ng, A. ve Jurafsky, D.** (2018). Noising and denoising natural language: Diverse backtranslation for grammar correction, *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, cilt 1, s.619–628.

[2] **Torunoğlu, D. ve Eryiğit, G.** (2014). A cascaded approach for social media text normalization of Turkish, *Proceedings of the 5th Workshop on Language Analysis for Social Media (LASM)*, s.62–70.

[3] **Eryiğit, G. ve Torunoğlu-Selamet, D.** (2017). Social media text normalization for Turkish, *Natural Language Engineering*, *23*(6), 835–875.

[4] **Göker, S. ve Can, B.** (2018). Neural Text Normalization for Turkish Social Media, *2018 3rd International Conference on Computer Science and Engineering (UBMK)*, IEEE, s.161–166.

[5] **Choudhury, M.**, **Saraf, R.**, **Jain, V.**, **Mukherjee, A.**, **Sarkar, S. ve Basu, A.** (2007). Investigation and modeling of the structure of texting language, *International Journal of Document Analysis and Recognition (IJDAR)*, *10*(3-4), 157–174.

[6] **Aw, A.**, **Zhang, M.**, **Xiao, J. ve Su, J.** (2006). A phrase-based statistical model for SMS text normalization, *Proceedings of the COLING/ACL on Main conference poster sessions*, Association for Computational Linguistics, s.33–40.

[7] **Cook, P. ve Stevenson, S.** (2009). An unsupervised model for text message normalization, *Proceedings of the workshop on computational approaches to linguistic creativity*, Association for Computational Linguistics, s.71–78.

[8] **Han, B. ve Baldwin, T.** (2011). Lexical normalisation of short text messages: Makn sens a# twitter, *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, Association for Computational Linguistics, s.368–378.

[9] **Liu, F.**, **Weng, F.**, **Wang, B. ve Liu, Y.** (2011). Insertion, deletion, or substitution?: normalizing text messages without pre-categorization nor supervision, *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers-Volume 2*, Association for Computational Linguistics, s.71–76.

[10] **Kobus, C.**, **Yvon, F. ve Damnati, G.** (2008). Normalizing SMS: are two metaphors better than one?, *Proceedings of the 22nd International Conference on Computational Linguistics-Volume 1*, Association for Computational Linguistics, s.441–448.

[11] **Beaufort, R.**, **Roekhaut, S.**, **Cougnon, L.A. ve Fairon, C.** (2010). A hybrid rule/model-based finite-state framework for normalizing SMS messages, *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, Association for Computational Linguistics, s.770–779.

[12] **Clark, E. ve Araki, K.** (2011). Text normalization in social media: progress, problems and applications for a pre-processing system of casual English, *Procedia-Social and Behavioral Sciences*, *27*, 2–11.

[13] **Liu, F.**, **Weng, F. ve Jiang, X.** (2012). A broad-coverage normalization system for social media language, *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*, Association for Computational Linguistics, s.1035–1044.

[14] **Han, B.**, **Cook, P. ve Baldwin, T.** (2013). Lexical normalization for social media text, *ACM Transactions on Intelligent Systems and Technology (TIST)*, *4*(1), 5.

[15] **Hassan, H. ve Menezes, A.** (2013). Social text normalization using contextual graph random walks, *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, cilt 1, s.1577–1586.

[16] **Sidarenka, U.**, **Scheffler, T. ve Stede, M.** (2013). Rule-based normalization of German Twitter messages, *Proc. of the GSCL Workshop Verarbeitung und Annotation von Sprachdaten aus Genres internetbasierter Kommunikation*.

[17] **Xu, K.**, **Xia, Y. ve Lee, C.H.** (2015). Tweet normalization with syllables, *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, s.920–928.

[18] **Saloot, M.A.**, **Idris, N. ve Aw, A.** (2014). Noisy text normalization using an enhanced language model, *Proceedings of the International Conference on Artificial Intelligence and Pattern Recognition*, SDIWC Kuala Lumpur, Malaysia, s.111–122.

[19] **van der Goot, R. ve van Noord, G.** (2017). Monoise: Modeling noise using a modular normalization system, *arXiv preprint arXiv:1710.03476*.

[20] **Tursun, O. ve Cakici, R.** (2017). Noisy uyghur text normalization, *Proceedings of the 3rd Workshop on Noisy User-generated Text*, s.85–93.

[21] **Matthews, D.** (2007). Machine transliteration of proper names, *Master's Thesis, University of Edinburgh, Edinburgh, United Kingdom*.

[22] **Pennell, D. ve Liu, Y.** (2011). A character-level machine translation approach for normalization of SMS abbreviations, *Proceedings of 5th International Joint Conference on Natural Language Processing*, s.974–982.

[23] **Ljubešic, N.**, **Zupan, K.**, **Fišer, D. ve Erjavec, T.** (2016). Normalising Slovene data: historical texts vs. user-generated content, *Proceedings of the Conference on Natural Language Processing*, s.146–155.

[24] **Ikeda, T.**, **Shindo, H. ve Matsumoto, Y.** (2016). Japanese text normalization with encoder-decoder model, *Proceedings of the 2nd Workshop on Noisy User-generated Text (WNUT)*, s.129–137.

[25] **Scherrer, Y. ve Ljubešić, N.** (2016). Automatic normalisation of the Swiss German ArchiMob corpus using character-level machine translation, *Proceedings of the 13th Conference on Natural Language Processing (KONVENS 2016)*, s.248–255.

[26] **Li, C. ve Liu, Y.** (2012). Normalization of text messages using character- and phone-based machine translation approaches, *Thirteenth Annual Conference of the International Speech Communication Association*.

[27] **Lopez Ludeña, V.**, **San Segundo Hernández, R.**, **Montero Martínez, J.M.**, **Barra Chicote, R. ve Lorenzo Trueba, J.** (2012). Architecture for text normalization using statistical machine translation techniques.

[28] **Lison, P. ve Tiedemann, J.** (2016). OpenSubtitles2016: Extracting large parallel corpora from movie and TV subtitles.

[29] **Levenshtein, V.I.** (1966). Binary codes capable of correcting deletions, insertions, and reversals, *Soviet physics doklady*, cilt 10, s.707–710.

[30] **Oflazer, K.** (1994). Two-level description of Turkish morphology, *Literary and linguistic computing*, *9*(2), 137–148.

[31] **Yildiz, E.**, **Tirkaz, C.**, **Sahin, H.B.**, **Eren, M.T. ve Sonmez, O.O.** (2016). A morphology-aware network for morphological disambiguation, *Thirtieth AAAI Conference on Artificial Intelligence*.

[32] **Pamay, T.**, **Sulubacak, U.**, **Torunoğlu-Selamet, D. ve Eryiğit, G.** (2015). The annotation process of the ITU Web Treebank, *Proceedings of the 9th Linguistic Annotation Workshop*, s.95–101.

[33] **Scherrer, Y. ve Erjavec, T.** (2013). Modernizing historical Slovene words with character-based SMT, *BSNLP 2013-4th Biennial Workshop on Balto-Slavic Natural Language Processing*.

[34] **Gao, Q. ve Vogel, S.** (2008). Parallel implementations of word alignment tool, *Software engineering, testing, and quality assurance for natural language processing*, 49–57.

[35] **Heafield, K.** (2011). KenLM: Faster and smaller language model queries, *Proceedings of the sixth workshop on statistical machine translation*, Association for Computational Linguistics, s.187–197.

[36] **Koehn, P.**, **Hoang, H.**, **Birch, A.**, **Callison-Burch, C.**, **Federico, M.**, **Bertoldi, N.**, **Cowan, B.**, **Shen, W.**, **Moran, C.**, **Zens, R.** *ve diğerleri* (2007). Moses: Open source toolkit for statistical machine translation, *Proceedings of the 45th annual meeting of the association for computational linguistics companion volume proceedings of the demo and poster sessions*, s.177–180.

EKLER

EK A.1 : Levenhstein Ağırlıkları

EK A.1

Bu bölümde yer alan Levenshtein ağırlıkları, yapay paralel veri üretmek için kullanılmıştır. Yapay paralel veri üretimi oldukça zor bir süreç olduğundan ağırlıklar çoğu kez sonuçlara göre tekrar gözden geçirilmiştir ve yeniden ayarlanmıştır. Dolayısıyla bu çizelgede bulunan ağırlıklar değerleri, bu çalışmada kullanılan ağırlık değerlerinin tümünü doğru olarak yansıtmayabilir.

Çizelge A.1 : Levenshtein ağırlıkları (Parça 1)

Ağırlık Tipi	**Harfler Harf Çiftleri**	**Değer**	**Anlam**
Ekleme (Sadece Fiiler)	s, k, r, y, ğ	0.1	
Ekleme	a, e, o, ö, ’, h	0.1	
Ekleme	ı, i, u, ü	0.1	
Ekleme	r, y, ğ, h	0.1	Bazı ünsüzleri yakalama
Değiştirme	i→ı, u→ü, o→ö s→ş, c→ç, g→ğ	0.0	Fonetik Hataları Yakalama
Değiştirme (Fiil)	k→z	0.3	yapıyok →yapıyoruz
Değiştirme (Fiil)	i→a	0.3	anliyim →anlayayım
Değiştirme (Fiil)	ı→a	0.3	anlıyım →anlayayım
Değiştirme (Fiil)	i→e	0.3	izlice →izleyeceğiz
Değiştirme (Fiil)	ç→c	0.3	yapçek →yapacak
Değiştirme (Fiil)	d→t	0.3	bitdi →bitti
Değiştirme (Fiil)	z→c	0.3	yapazak →yapacak
Değiştirme	0→o	0.4	
Değiştirme	0→ö	0.4	
Değiştirme	q→g	0.4	
Değiştirme	q→ğ	0.4	
Değiştirme	q→k	0.4	yoq →yok
Değiştirme	w→v	0.4	yawrum →yavrum
Değiştirme	3→e	0.4	
Değiştirme	$→ş	0.4	$arkı →şarkı
Değiştirme	$→s	0.4	
Değiştirme	j→c	0.4	abijim →abicim
Değiştirme	h→k	0.8	yoh →yok
Değiştirme	v→u	0.8	

Çizelge A.2 : Levenshtein ağırlıkları (Parça 2)

Ağırlık Tipi	**Harfler Harf Çiftleri**	**Değer**	**Anlam**
Değiştirme	z→s	0.8	herkez →herkes
Değiştirme ve Koş. Silme	q →(w, a)	0.9	Komşu Harfler
Değiştirme ve Koş. Silme	w →(q, e, s, a)	0.9	Komşu Harfler
Değiştirme ve Koş. Silme	e →(w, s, d, r)	0.9	Komşu Harfler
Değiştirme ve Koş. Silme	r →(e, d, f, t)	0.9	Komşu Harfler
Değiştirme ve Koş. Silme	t →(r, f, g, y)	0.9	Komşu Harfler
Değiştirme ve Koş. Silme	y →(t, g, h, u)	0.9	Komşu Harfler
Değiştirme ve Koş. Silme	u →(y, h, j, ı)	0.9	Komşu Harfler
Değiştirme ve Koş. Silme	ı →(u, j, k, o)	0.9	Komşu Harfler
Değiştirme ve Koş. Silme	o →(ı, k, l, p)	0.9	Komşu Harfler
Değiştirme ve Koş. Silme	p →(o, l, ş, ğ)	0.9	Komşu Harfler
Değiştirme ve Koş. Silme	ğ →(p, ş, i, ü)	0.9	Komşu Harfler
Değiştirme ve Koş. Silme	ü →(ğ, i)	0.9	Komşu Harfler
Değiştirme ve Koş. Silme	a →(q, w, s, z)	0.9	Komşu Harfler
Değiştirme ve Koş. Silme	s →(w, e, d, x, z, a)	0.9	Komşu Harfler
Değiştirme ve Koş. Silme	d →(e, r, f, c, x, s)	0.9	Komşu Harfler
Değiştirme ve Koş. Silme	f →(r, t, g, v, c, d)	0.9	Komşu Harfler
Değiştirme ve Koş. Silme	g →(t, y, h, b, v, f)	0.9	Komşu Harfler
Değiştirme ve Koş. Silme	h →(y, u, j, n, b, g)	0.9	Komşu Harfler
Değiştirme ve Koş. Silme	j →(u, ı, k, m, n, h)	0.9	Komşu Harfler
Değiştirme ve Koş. Silme	k →(ı, o, l, ö, m, j)	0.9	Komşu Harfler
Değiştirme ve Koş. Silme	l →(o, p, ş, ç, ö, k)	0.9	Komşu Harfler
Değiştirme ve Koş. Silme	ş →(p, ğ, i, ç, l)	0.9	Komşu Harfler
Değiştirme ve Koş. Silme	i →(ğ, ü, ş)	0.9	Komşu Harfler

Çizelge A.3 : Levenshtein ağırlıkları (Parça 3)

Ağırlık Tipi	**Harfler Harf Çiftleri**	**Değer**	**Anlam**
Değiştirme ve Koş. Silme	z →(a, s, x)	0.9	Komşu Harfler
Değiştirme ve Koş. Silme	x →(z, s, d, c)	0.9	Komşu Harfler
Değiştirme ve Koş. Silme	c →(x, d, f, v)	0.9	Komşu Harfler
Değiştirme ve Koş. Silme	v →(c, f, g, b)	0.9	Komşu Harfler
Değiştirme ve Koş. Silme	b →(v, g, h, n)	0.9	Komşu Harfler
Değiştirme ve Koş. Silme	n →(b, h, j, m)	0.9	Komşu Harfler
Değiştirme ve Koş. Silme	m →(n, j, k, ö)	0.9	Komşu Harfler
Değiştirme ve Koş. Silme	ö →(m, k, l, ç)	0.9	Komşu Harfler
Değiştirme ve Koş. Silme	ç →(ö, l, ş)	0.9	Komşu Harfler
Silme	ı, i	0.8	“sıtandart”, “tiren”

ÖZGEÇMİŞ

Ad Soyad: Talha Çolakoğlu

Doğum Tarihi ve Yeri: 06.01.1993, Gölcük

E-Posta: colakoglut@itu.edu.tr

ÖĞRENİM DURUMU:

- **Lisans:** 2016, İstanbul Teknik Üniversitesi, Bilgisayar Mühendisliği Anabilim Dalı, Bilgisayar Mühendisliği Programı.
- **Y. Lisans:** 2019, İstanbul Teknik Üniversitesi, Bilgisayar Mühendisliği Anabilim Dalı, Bilgisayar Mühendisliği Programı.

MESLEKİ DENEYİMLER VE ÖDÜLLER:

- Araştırma Görevlisi, 2017 - 2019. İTÜ Bilgisayar Bilişim Fakültesi.

YÜKSEK LİSANS TEZİNDEN TÜRETİLEN YAYINLAR, SUNUMLAR VE PATENTLER:

- Colakoglu T., Sulubacak U., Tantuğ A. C. Normalizing Non-canonical Turkish Texts Using Machine Translation Approaches *Association for Computational Linguistics*, Student Research Workshop, 28 July - 2 August, 2019 Florence, Italy.