HACETTEPE ÜNİVERSİTESİ
EĞİTİM BİLİMLERİ ENSTİTÜSÜ

Türkçe ve Sosyal Bilimler Eğitimi Ana Bilim Dalı

Türkçe Eğitimi Programı

DİZİLERDEN BİRİMLERE:

BİLİŞİMSEL DİLBİLİM ÇERÇEVESİNDE BİR BİRİMLENDİRİCİ TASARIMI

Taner SEZER

Doktora Tezi

Ankara, 2025

Liderlik, araştırma, inovasyon, kaliteli eğitim ve değişim ile

Daha ileriye... En İyiye...

HACETTEPE ÜNİVERSİTESİ
EĞİTİM BİLİMLERİ ENSTİTÜSÜ

Türkçe ve Sosyal Bilimler Eğitimi Ana Bilim Dalı

Türkçe Eğitimi Programı

DİZİLERDEN BİRİMLERE:

BİLİŞİMSEL DİLBİLİM ÇERÇEVESİNDE BİR BİRİMLENDİRİCİ TASARIMI

STRINGS TO TOKENS:

DESIGNING A TOKENIZER WITHIN A COMPUTATIONAL LINGUISTICS FRAMEWORK

Taner SEZER

Doktora Tezi

Ankara, 2025

Kabul ve Onay

Eğitim Bilimleri Enstitüsü Müdürlüğüne,

Taner SEZER'in hazırladığı "Dizilerden Birimlere: Bilişimsel Dilbilim Perspektifiyle bir Birimlendirici Tasarımı" başlıklı bu çalışma jürimiz tarafından **Türkçe ve Sosyal Bilimler Eğitimi Ana Bilim Dalı, Türkçe Eğitimi Bilim Dalında Doktora Tezi** olarak kabul edilmiştir.

Jüri Başkanı	Prof. Dr. Bilginer ONAN	İmza
Jüri Üyesi (Danışman)	Prof. Dr. Özay KARADAĞ	İmza
Jüri Üyesi	Prof. Dr. Mehmet KURUDAYIOĞLU	İmza
Jüri Üyesi	Prof. Dr. Bayram BAŞ	İmza
Jüri Üyesi	Doç. Dr. Zafer KIYAN	İmza

Bu tez Hacettepe Üniversitesi Lisansüstü Eğitim, Öğretim ve Sınav Yönetmeliği'nin ilgili maddeleri uyarınca yukarıdaki jüri üyeleri tarafından 27 / 06 / 2025 tarihinde uygun görülmüş ve Enstitü Yönetim Kurulunca ..... / ..... / ..... tarihi itibarıyla kabul edilmiştir.

Prof. Dr. İsmail Hakkı MİRİCİ

Eğitim Bilimleri Enstitüsü Müdürü

Öz

Birimlendirme, doğal dilin bilgisayar tarafından işlenmesinde temel bir adımdır; bu işlem karakter dizilerinin anlamlı birimlere dönüştürülmesini sağlar. Çalışmada, bir metni oluşturan her karakter dizisinin bir birim olabileceği, ancak her birimin mutlaka bir sözcük olması gerekmediği savından hareketle özgün bir birimlendirici tasarlanmıştır. TS Tokenizer adıyla geliştirilen araç, düzenli ifadeler ve sözcük listelerini birleştiren hibrit bir yapı ile tasarlanmıştır. Böylelikle hem sözcükler hem de rakamlar, noktalama işaretleri, sosyal medyaya özgü kullanımlar gibi sözcük dışı birimler ilgili karakter dizisinin metin içindeki işlevi temel alınarak birimlendirme sürecine dahil edilmiştir. TS Tokenizer, Python diliyle hazırlanmış, hem bir Python kütüphanesi olarak hem de komut satırı üstünden kullanılabilecek şekilde tasarlanmış ve sunulmuştur.

Çalışmada, geleneksel birimlendirme yaklaşımları ile güncel sözcük-altı birimlendirme yöntemleri karşılaştırmalı olarak ele alınmış; bu yöntemlerin sözcük bütünlüğünü gözetmemesi nedeniyle sosyal bilimlerin dil çözümleme beklentilerini karşılamada yetersiz kaldığı eleştirel bir bakışla tartışılmıştır.

Uygulama bölümünde NLTK kütüphanesinin sunduğu birimlendiriciler ile TS Tokenizer karşılaştırılmış, ders kitaplarından ve sosyal medya verilerinden oluşan iki farklı derlem analiz edilmiştir. Her iki derlemde de TS Tokenizer'ın, daha verimli sonuç ürettiği, sözcük bütünlüğünü koruma ve noktalama işaretlerinin ayrıştırılması gibi ölçütlerde daha yüksek başarı sağladığı gözlenmiştir.

Sonuç olarak, TS Tokenizer; dil eğitimi, uygulamalı dilbilim, derlem dilbilim ve dijital beşeri bilimler gibi alanlarda özgün birimlendirme ihtiyaçlarına yanıt veren, erişilebilir ve açık kaynaklı bir birimlendirici olarak sunulmuştur.

Anahtar sözcükler: birim, birimlendirme, birimlendirici, bilişimsel dilbilim, TS Tokenizer

Abstract

Tokenization is a fundamental step in the computational processing of natural language, enabling the conversion of character sequences into meaningful tokens. This study is based on the premise that any sequence of characters in a text can be considered as a token, but not every token must necessarily be a word. Upon this idea, a novel tokenizer named TS Tokenizer has been developed. This tool employs a hybrid architecture that combines using both regular expressions and lexicons. In this way, both words and non-word elements, such as numbers, punctuation marks, and social media-specific expressions, are included in the tokenization process based on their function within the text. TS Tokenizer is implemented in Python and is designed to function both as a Python library and a command-line tool.

The study provides a comparative analysis of traditional tokenization methods and recent subword-level approaches. These newer methods are critically examined for their inability to preserve word integrity, a limitation that reduces their usefulness for linguistic analysis in the social sciences.

In the experimental section, TS Tokenizer is evaluated against existing tokenizers provided by the NLTK library using two different corpora: one derived from textbooks and the other from social media data. In both corpora, TS Tokenizer demonstrated better performance, particularly in preserving word boundaries and correctly segmenting punctuation.

In conclusion, TS Tokenizer is presented as an accessible and open-source tokenizer that responds to the specific tokenization needs of fields such as language education, applied linguistics, corpus linguistics, and digital humanities.

Keywords: token, tokenization, tokenizer, computational linguistics, TS Tokenizer

Teşekkür

Bu doktora çalışmasının ilk adımları, 2010 yılında yüksek lisans diplomamı aldıktan hemen sonra atıldı. O günden bu yana geçen on beş yıl üreterek öğrendiğim uzun bir yolculuk oldu. Bu yolculuğun her durağında yanımda olan, emeğiyle, sözüyle, yüreğiyle katkı sunan herkese gönülden teşekkür ederim.

Bu çalışmanın şekillenmesinde büyük paya sahip olan, değerli tez danışmanım Prof. Dr. Özay Karadağ'a düşünsel ufkumu genişleten bakış açısıyla katkı sunduğu için şükranlarımı sunuyorum.

Tez izleme komitemde yer alarak, fikirleriyle çalışmama yeni yönler kazandıran ve beni her zaman cesaretlendiren Prof. Dr. Bayram Baş ve Prof. Dr. Mehmet Kurudayıoğlu'na en içten teşekkürlerimi sunuyorum. Ayrıca tez savunma jürimde yer almayı kabul ederek bu çalışmayı kıymetlendiren Prof. Dr. Bilginer Onan ve Doç. Dr. Zafer Kıyan hocalarıma da bu çalışmayı zenginleştirdikleri için minnettarım.

Akademik yolculuğumun pek çok durağında birlikte çalışmaktan büyük mutluluk duyduğum ve ilham verici katkılarına her zaman kıymet verdiğim sevgili hocalarım Berna Arslan ve Erhan Arslan'a teşekkür ederim. Bilgiyle olduğu kadar dostlukla da yanımda oldular.

Bu sürecin her anında koşulsuz destekleriyle yanımda olan aileme olan borcumu kelimelerle ifade etmek güç. Sevgili annem Serpil Çağalıoğlu'na desteği ve canım kardeşim Türker Sezer'e ne zaman ihtiyaç duysam orada olduğu ve tüm geniş aileme bana inandıkları için gönülden teşekkür ederim.

Ve elbette, bu sürecin her anında sevgiyle, sabırla, anlayışla yanımda olan iki canıma… Hayat arkadaşım Bengü Sezer'e, varlığıyla bana yalnız olmadığımı her daim hissettirdiği için; ve biricik kızım Defne Sezer'e, her gülüşüyle bu hayatın ne kadar güzel olduğunu bana tekrar tekrar hatırlattığı için sonsuz sevgimle teşekkür ediyorum. Siz olmasaydınız hayat bu kadar anlamlı olmazdı.

İçindekiler

Tablolar Dizini

Şekiller Dizini

Simgeler ve Kısaltmalar Dizini

BD: Bilişimsel Dilbilim (*Computational Linguistics*)

DDİ: Doğal Dil İşleme (*Natural Language Processing*)

BPE: Bayt-Çift Kodlama (*Byte-Pair Encoding*)

Bölüm 1

Giriş

Dil, insanın bilişsel gelişiminin en aktif ve en etkin bileşenlerinden biridir. Pagel dünyayı paylaşan yaşamsal çeşitlilik içinde insanın en başarılı tür olmasını dille ilişkilendirmektedir (Pagel, 2017). İnsan, dile sahip olması sayesinde üzerinde var olduğu gezegenin yaşanabilir her noktasını fethetmeyi başarmıştır. Dil insana "bilgiyi verimli şekilde aktaracak" bir kod sağlamıştır. İnsanlık tarihi boyunca başlangıçta sözle gerçekleşen bilgi aktarımı yazının icadıyla birlikte kalıcılık ve yayılma gücü kazanmıştır.

Yazılı bilgi yüzyıllar boyunca el yazması eserlerle çoğaltılmıştır. Makineler kullanılarak yapılan basım önce uzak doğuda ortaya çıkmış, Avrupa'ya 1438'de Gutenberg tarafından geliştirilerek taşınmıştır (Forrester, 2020; Moran, 1971). Matbaanın bulunuşu, basılı eserlerin daha hızlı ve daha çok sayıda basılmasını ve dağıtılmasını sağlamıştır. Hausser, matbaanın bulunuşunu Birinci Gutenberg Devrimi, 20. yüzyılın ortalarında başlayan bilgi çağıyla birlikte metinlerin sayısal ortama aktarılmasını ise İkinci Gutenberg Devrimi olarak adlandırmaktadır (Hausser, 2013). Bilgi çağıyla birlikte insanların bilgi aktarımı mecraları dönüşmüş ve gelişmiştir.

Bilgi çağının en önemli gelişmelerinden biri internettir. İnternetin yaygınlaşması dil çalışmalarını da farklı bir boyuta taşımış, birbiriyle ilişkili iki temel eksende geliştirmiştir. Birinci eksen internetin yaygınlaşmasıyla "erişilebilir" veri hacminin katlanarak artması olmuştur. Bu durum dilbilim çalışmaları için eşsiz bir fırsat sunmaktadır. İkinci eksense internet öncesi dönemde sınırlı kalmış olan "yazma ve yayma" fırsatının günümüzde hemen herkes tarafından erişilebilir duruma gelmiş olmasıdır. Böylece toplumun her kesiminden bireylerin ürettiği metinler dilsel çalışmaların nesnesi olarak kullanılabilir hale gelmiştir. Üretici sayısındaki artış verinin hacimsel artışını da beraberinde getirmiştir.

İnsan zihni dili hem üretmek hem de işlemek konusunda oldukça beceriklidir. Bağlama, zamana, konuya, konuma, aktörlere vb. bir çok başka üst bilgiye göre insan

zihni *(yazılı veya sözlü)* dilsel çıktıyı eş zamanlı olarak işleyebilmektedir. Bu işlemleme becerisinde şüphesiz insanın dünya bilgisi ve sağduyusu aktif birer etkendir. Dilin bilgisayar ile işlenmesi, çeşitli örüntülerin ve istatistiki sonuçların çıkarılması bilişimsel dilbilimin (BD), insan ve bilgisayar arasında doğal dil ile etkileşim sağlanması doğal dil işleme (DDİ) alanlarının konusudur. Ancak, aynı dil verisini bilgisayar ile işleyebilmek için tüm bu ön bilginin eksiksiz olarak sayısal ortamda tanımlanması *(henüz)* mümkün değildir.

Sayısal platformdaki metinler harf, noktalama işareti, rakamlar, boşluk karakteri, satır sonu işareti vb. karakterlerden oluşan dizilerdir (Mikheev, 2005). Bu diziler bilgisayarlar tarafından ardışık bir karakterler dizisi (string) olarak algılanır (Vijayarani & R.Janani, 2016). Metinlerin bilgisayar aracılığıyla işlenmesi/incelenmesi için ilk adım verinin "kullanılabilir birimlere" ayrıştırılmasıdır (Barcala, 2002; Webster & Kit, 1992). Bu işlem "birimlendirme" (*tokenization*) olarak tanımlanmaktadır. Rychlý, birimlendirmenin ardışık işlemlerden oluşan bir DDİ sürecinin ilk adımı olduğunu söylemektedir (Rychlý & Špalek, 2022). Birimlendirme işleminin başarımı ardından gelecek işlemlerin veya işlem adımlarının başarımı üzerinde de doğrudan etkilidir (Rust ve diğerleri., 2020).

Bu çalışmanın amacı bakımından, söz konusu iki alanın, BD ve DDİ perspektifinden bakarak dil için şu tanımı yapabiliriz: "*Doğal dil, insan zihninin çok katmanlı ve kaotik bir ürünüdür*". Dolayısıyla, insan zihninden farklı bir araç kullanarak doğal dili işlemek çok sayıda katmana yüklenmiş bilgi parçacıklarını bir araya getirerek anlamak, başlangıcı ve bitişi belirsiz dizilerle zorlu bir mücadeleye girişmek anlamına gelmektedir. Bu açıdan, yukarıdaki tanım, bu çalışmaya konu olan ve doğal dilin bilgisayar tarafından işlenmesi sürecinin ilk adımı olarak kabul edilen *birimlendirme* işleminin hem önemini hem de zorluğunu açıklamak için uygun olacaktır.

Problem Durumu

Bu çalışmada, temel olarak dil eğitimi, uygulamalı dilbilim, derlem dilbilim, dijital insani bilimler ve söz varlığı çalışmaları gibi sosyal bilimler alanlarında kullanılmak üzere,

Türkçe metinleri işlenebilir ve analiz edilebilir hale getirecek bir birimlendiricinin tasarlanması amaçlanmıştır. Bu perspektifle, hedeflenen birimlendiricinin tasarlanması sürecinde birbiriyle ilişkili iki önemli problem olduğu görülmektedir. Bunlar birimlendirici tasarımının hangi amaçla yapıldığının belirlenmesi ve neyin "birim" olarak kabul edileceğinin tanımlanmasıdır. Bu çalışmada her iki problem de sosyal bilimler perspektifinden, yukarıda anılan disiplinler ışığında ele alınmıştır.

Zaman içinde metinlerin birimlendirilmesi için farklı yöntemler kullanılmıştır. Ancak, sosyal bilimleri ilgilendiren en belirgin ayrışma "sözcük vektör temsilleri" (*Word2Vec*) yöntemlerinin ortaya çıkışıyla birlikte gözlenmiştir. 2013'te Mikolov tarafından geliştirilen "sözcük vektör temsilleri" yöntemi büyük hacimli ve *etiketlenmemiş* dil verisi kullanılarak sözcükler arasındaki ilişkileri belirlemeye ve dil modelleri üretmeye yarayan bir yöntemdir (Mikolov ve diğerleri, 2013). Bu yöntemin ortaya çıkışından önce DDİ çalışmalarında iyi etiketlenmiş veri setlerine ihtiyaç duyulmaktaydı (Ploeger ve diğerleri, 2025). Bu dönemde dilsel veri çoğunlukla sözcük türü, sözcüklerin biçimbirimsel özellikleri, sözdizimsel yapı etiketleri vb. etiketlemeler ile zenginleştirilmiş veri setlerini, ağaç yapılı derlemleri (*treebank*) işaret etmektedir. Çünkü üretilen dil modelleri etiketlenmiş bu veriden edindikleri bilgiyle, bir başka deyişle her bir sözcüğün dilbilgisel rolü, cümledeki konumu, sözcüğün taşıdığı biçimbirimsel özellikleri gibi özniteliklerin analiziyle hazırlanmaktaydı. Dolayısıyla bu veri setleri ve bu veri setlerinin üretimi için DDİ alanında kullanılan araçlar büyük oranda sosyal bilimlerin ilgi alanına giren çalışmalarda da kullanılabilmekteydi. 2013 sonrasındaysa Word2Vec modellerle başlayan ve zamanla büyük dil modellerinin (LLM) üretimine evrilen süreçte verinin her satırda bir cümle olacak şekilde ayrıştırılması temel veriyi oluşturabilmekteydi. Bu sebeple birimlendirme DDİ alanında farklı bir yaklaşım ve ihtiyaçla kullanılmaya başlanmıştır.

İkinci problemse bir metni oluşturan hangi dilsel unsurların "birim" olarak kabul edileceğinin belirlenmesidir. Grefensette'nin belirttiği gibi doğal dilde neyin birim olarak kabul edileceğine karar vermenin birçok yolu vardır (Grefensette & Tapanainen, 1994).

Öte yandan, dilin durağan olmaması sebebiyle “birim” olarak ele alınabilecek unsurlar da kapalı, değişmez bir küme olarak düşünülemez. Örneğin, e-posta adresleri, URL adresleri gibi internet sonrası döneme ilişkin yapılar ile, mention ve hashtag gibi sosyal medyaya ilişkin kullanımlar zamanla DDİ’nin üstünde durduğu yapılar olmuş, dolayısıyla birimlendiricilerin ilgi alanına da girmişlerdir.

DDİ birimlendirme işleminden, hedeflenen amaca göre verinin hazırlanması için bir araç olarak faydalanır. Daha açık bir ifadeyle, DDİ’nin temelde yapay zekâ çalışmalarının bir alt disiplini olması sebebiyle bu alanda birimlendirme her zaman ölçünlü dil ile tam uyum aramaz. Özellikle son on yılda makine öğrenme, yapay sinir ağları ve büyük dil modelleri (*Large Language Models*) üretmek için geniş hacimli veri setleri “sözcük-altı birimler” ve “bayt-çiftli kodlama” gibi yöntemlerle birimlendirilmektedir. Bu yöntemlerde sözcük bütünlüğünün korunması bir hedef değildir. Örneğin önemli sözcüğü önem+*li* biçiminde, etkilidir sözcüğü etkili+*dir* şeklinde iki ayrı parça olarak birimlere ayrıştırılabilir. Birimlendirme işleminin dil modelleri üzerindeki etkisini incelediği çalışmasında Toraman, sözcük bazında birimlendirmenin “metin sınıflandırma, nefret söylemi tespiti, duygu durumu analizi, varlık adı tanıma” gibi DDİ konularında daha düşük performans gösterdiğini belirtirken bunu “bayt-çiftli kodlama” ve “sözcük-altı birimler” yaklaşımlarının sözlük dışı sözcükleri (*Out-Of-Vocabulary*, eğitim sırasında modelin görmediği sözcükler) bertaraf etme konusundaki yetkinliği ile açıklamaktadır (Toraman ve diğerleri, 2022). Bu tarz birimlendirme yaklaşımları bilgisayar bilimleri açısından kullanılabilir bir çıktı sağlamakta ve dil modellerinin eğitilmesinde verimli şekilde kullanılmaktadır.

Toraman’ın çalışmasında “toplumsal barış sağlanır” cümlesinin farklı yöntemlerle birimlendirilmiş çıktıları Şekil 1’de verilmektedir. Şekil 1 incelendiğinde, örnek olarak verilen 5 ayrı yaklaşımın birbirine benzemeyen çıktılar ürettiği görülmektedir.

Şekil 1

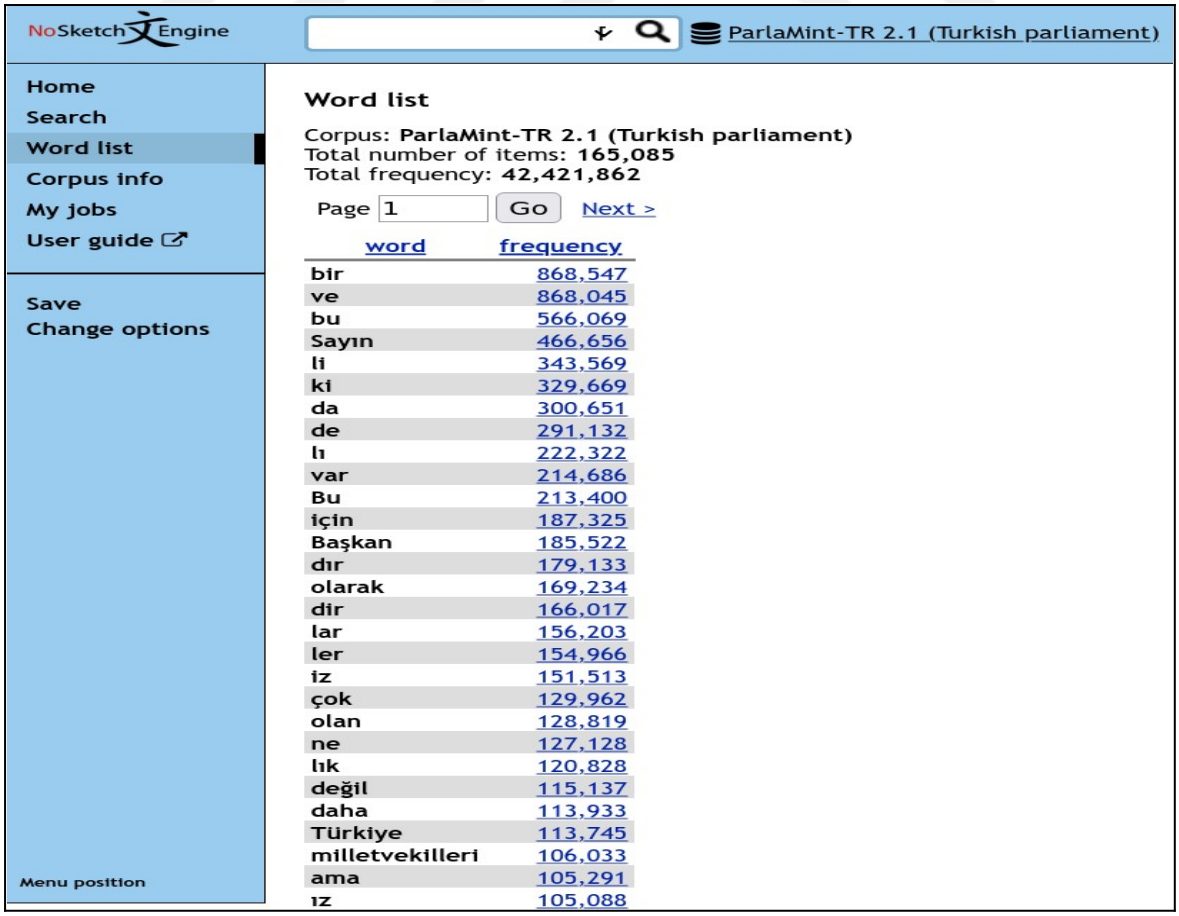
Farklı birimlendirici çıktıları. (Toraman ve diğerleri, 2022: s.6)

Method	Tokenized text
Character-level	"t", "o", "p", "l", "u", "m", "s", "a", "l", " ", "b", "a", "r", "ı", "ş", " ", "s", "a", "ğ", "l", "a", "n", "ı", "r"
BPE	"[CLS]", "toplumsal", "barış", "sağ", "##lanır", "[SEP]"
WordPiece	"[CLS]", "toplumsal", "barış", "sağlan", "##ır", "[SEP]"
Morphological-level	"[CLS]", "toplum", "##sal", "barış", "sağ", "##lanır", "[SEP]"
Word-level	"[CLS]", "[UNK]", "barış", "[UNK]", "[SEP]"

Birimlendiricinin, yapılan çalışmanın hedefine uygunluğunun önemini açıklayabilmek adına Clarin tarafından yayınlanan, Türkiye Büyük Millet Meclisi tutanaklarından oluşan ve açık erişimle sunulan Clarin ParlaMint-TR 2.1 derlemi ilgi çekici bir örnektir. Aşağıdaki ekran görüntüsünde (CLARIN.SI, 2025a) bu derlem tarafından üretilen sözcük sıklık listesi görülmektedir.

Şekil 2

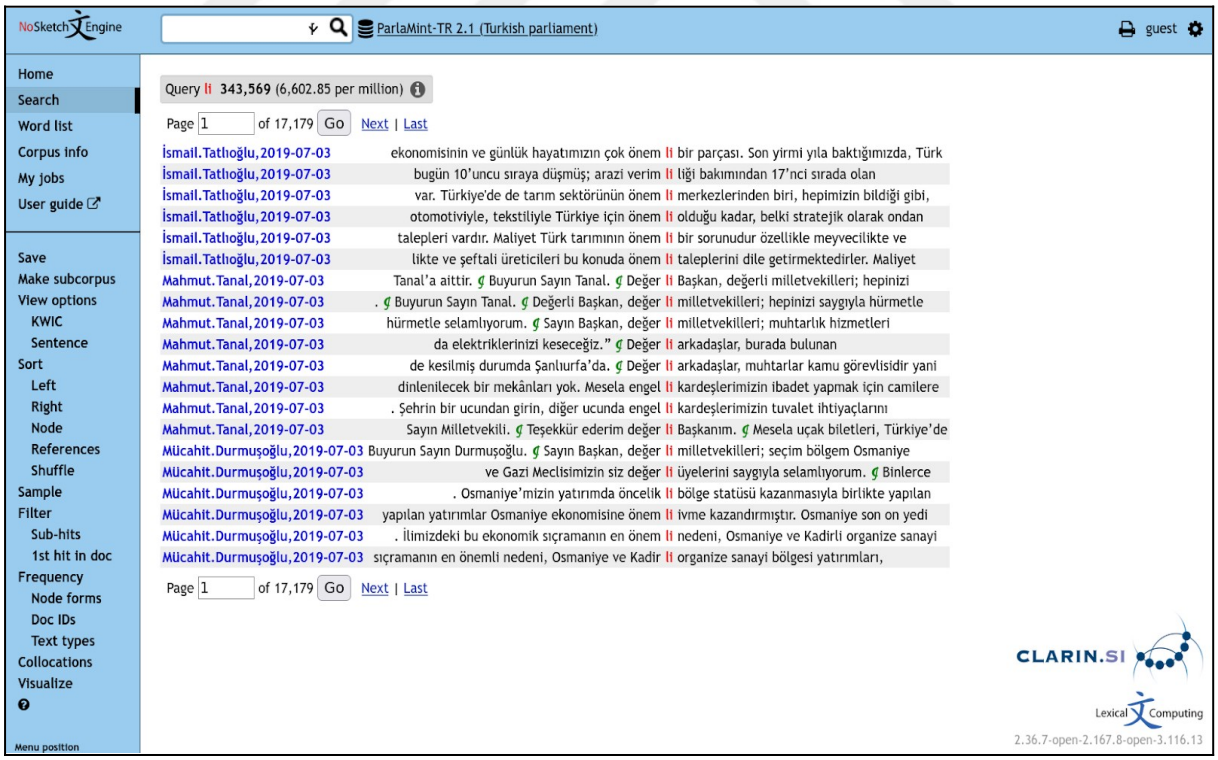
Clarin ParlaMint-TR 2.1 derlemi sözcük sıklık listesi (CLARIN.SI, 2025a)



Şekil 2’de verilen sözcük sıklık listesi incelendiğinde, bu derlemi oluşturan sözcük kadrosunda en sık gözlenen 5. sözcüğün “li”, 9. sözcüğün “lı”, 15. sözcüğün “dır”, 17. sözcüğün “dir”, 18. sözcüğün “lar”, 19. sözcüğün “ler”, 20. sözcüğün “iz” olduğu görülmektedir. Ekran görüntüsünde, derlemin 42 milyon 421 bin 862 sözcükten oluştuğu bilgisi verilirken bu derlemde toplam 165 bin 085 farklı sözcük olduğu belirtilmiştir. Ancak bu listeye göre en sık gözlenen 20 sözcüğün 7’si Türkçe bir sözlükte madde başı olarak bulunmayan veya madde başı bir sözcüğün çekimli bir biçimi olan birer sözcük değildir. Nitekim, ilgili sözcüklere tıklanarak bu sözcüklerin bulunduğu bağlam incelendiğinde de bu sonuç gözlenmektedir. Aşağıda, Clarin ParlaMint-TR 2.1 (CLARIN.SI, 2025b) derleminden alınmış ekran görüntüsünde “li” sözcüğünün bulunduğu bağlamlar listelenmektedir.

Şekil 3

Clarin ParlaMint-TR 2.1 derlemi üzerinde “li” sözcüğü için yapılan sorgunun ekran görüntüsü (CLARIN.SI, 2025b).



Şekil 3 incelendiğinde sıklık listesinde 5. sırada sözcük olarak verilen “li” yapısının aslında bir sözcük değil, isimden sıfat üreten bir ek olduğu görülmektedir. Son sırada yer

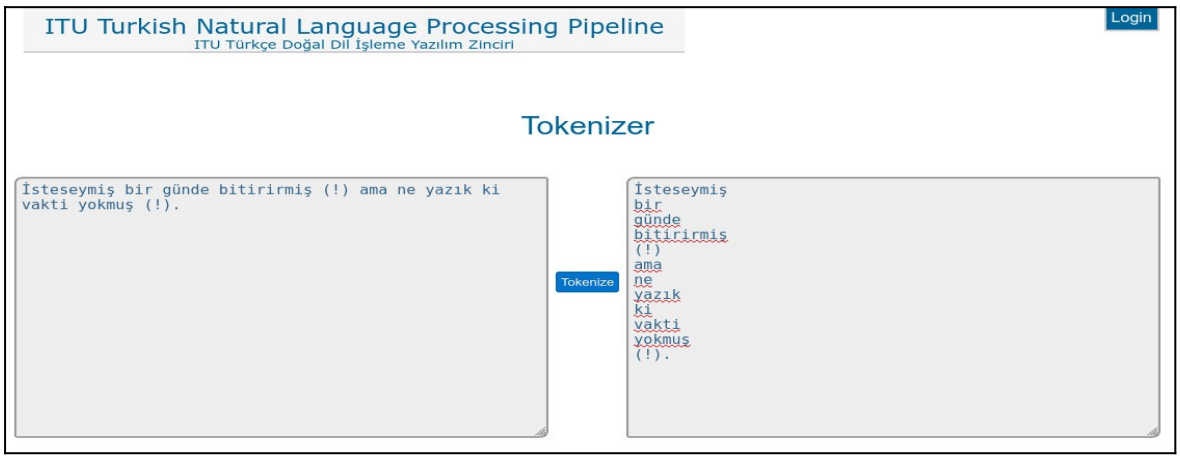
alan “Kadirli” (Adana iline bağlı bir ilçe) sözcüğü ise tamamen hatalı olarak “Kadir+li” şeklinde ayrıştırılmıştır. Derlem içinde söz konusu yapı bir sözcük olarak verilerek listede bulunan “ve”, “Sayın”, “Türkiye” sözcükleri ile eşdeğer biçimde Türkçenin söz varlığında yer alan bir sözcük olarak sunulmuştur. Bu derlemden elde edilecek verinin söz varlığı, eğitim bilimleri ve dilbilim gibi alanlarda doğrudan kullanılması mümkün görünmemektedir.

Sözcük sıklığı çözümlemesi konusunu ele aldığı çalışmasında Pilten-Ufuk iki çevrim içi (ITU Türkçe Metin Sıklık Çözümleyicisi ve TS Corpus Frequency Calculator) ve bir çevrim dışı aracı (AntConc 3.5.8) karşılaştırmış, bu araçların ürettiği sonuçlar arasındaki farklılığın araçların birimlendirme ölçütlerinin farklılığından kaynaklandığını belirtmiştir (Pilten-Ufuk, 2021).

Çevrim içi olarak erişilebilen İTÜ Türkçe Doğal Dil İşleme Yazılım Zinciri ve Hugging Face platformunda yayınlanan ve Stanford Üniversitesi bünyesinde geliştirilen bir DDİ yazılımı olan “Stanza’nın” Türkçe kütüphanesi incelendiğinde de benzer bir farklılaşma gözlenmektedir.

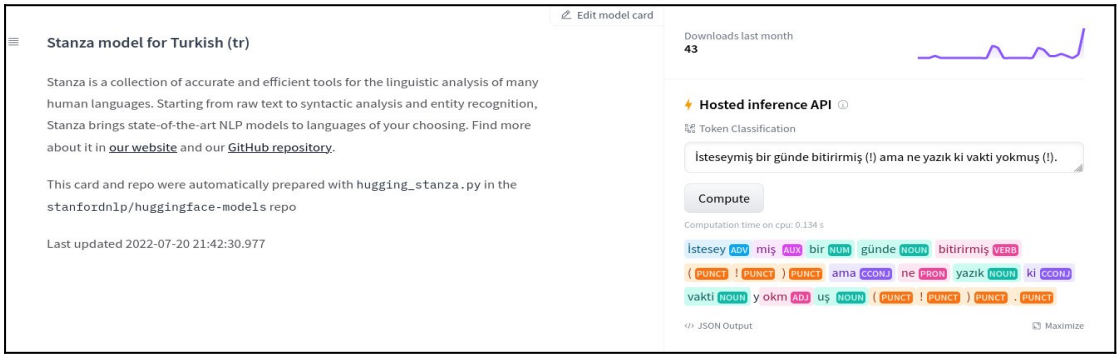
Şekil 4

İTÜ Doğal Dil İşleme Zinciri birimlendirici çıktısı



Şekil 5

Stanza Türkçe birimlendirici çıktısı

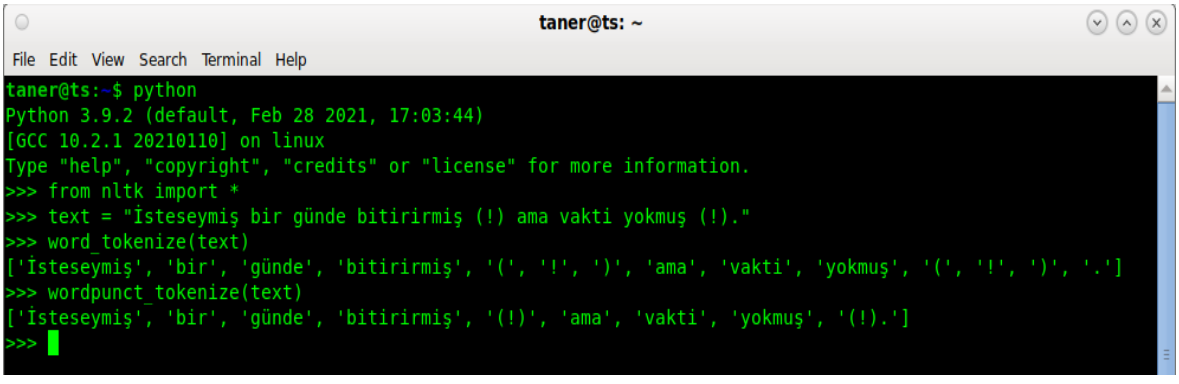


Şekil 4 ve Şekil 5 “İsteseymiş bir günde bitirirmiş (!) ama vakti yokmuş (!).” cümlesinin her iki yazılım tarafından nasıl birimlendirildiğini göstermektedir. İTÜ Türkçe DDİ Yazılım Zinciri sözcük formlarının korurken, sözcük-altı birimlere odaklanan Stanza *İsteseymiş* sözcüğünü istesey+miş olarak, *yokmuş* sözcüğünü y+okm+uş olarak birimlendirmiş, (!) dizisini ise üç ayrı noktalama işareti olarak sunmuştur.

Çevrimiçi araçlar donanım ve bağlantı hızı gibi limitler sebebiyle sınırlı miktarda veriyi işlemek veya aracın bir gösterimini yapmak amacıyla hazırlanmaktadır. Daha büyük verileri yerelde işlemek için BD ve DDİ alanında sıklıkla kullanılan bir kütüphane olan Python NLTK (Natural Language Toolkit) kütüphanesinde de çeşitli birimlendiriciler sunulmaktadır. Aşağıdaki ekran görüntüsünde NLTK kütüphanesininin sunduğu iki farklı metodla (word_tokenize, wordpunct_tokenize) aynı cümlenin işlenmesi sonucunda elde edilen çıktı verilmektedir.

Şekil 6

Python yorumlayıcısı üstünde üretilen NLTK kütüphanesi birimlendirici çıktıları



NLTK kütüphanesinde sunulan birimlendiricilerin yukarıda da dile getirildiği şekilde farklı çıktılar sunduğu gözlenmektedir. Evert'e göre bu tarz ayrışmalar üretilen yazılımların hedef farklılığından kaynaklanmaktadır (Evert, 2005).

Özetle, yukarıda anılan iki problem Türkçe metinlerin sosyal bilimleri temel alan bir amaç doğrultusunda verimli bir şekilde işlenmesi için çözülmesi gereken temel zorluklardır ve çalışmanın odak noktasını oluşturmaktadır.

Araştırmanın Amacı ve Önemi

Bu çalışmada, sayısal platformda bulunan Türkçe metinlerin verimli şekilde işlenmesi amacıyla, özellikle sözcük bütünlüğünü hedefleyen dil eğitimi, derlem dilbilim, uygulamalı dilbilim, eğitim bilimleri, dijital insanî bilimler vb. alanlar için tasarlanmış bir birimlendiricinin üretilmesi amaçlanmaktadır. Bu kapsamda ilgili alanlarda birim olarak kullanılabilecek dilsel unsurlar belirlenmiş ve hibrit bir yaklaşımla bir birimlendirici hazırlanmıştır. Hazırlanan birimlendirici "TS Tokenizer" olarak adlandırılmış, Python dili için bir kütüphane olarak sunulmuştur. Hazırlanan birimlendiricinin aşağıdaki konularda alana katkı sağlayacağı düşünülmektedir.

Birinci konu, sözcük bütünlüğüne ihtiyaç duyulan çalışma alanlarında bir standardın sağlanmasıdır. Örneğin, dil eğitimi ve edebiyat alanlarında popüler bir konu olan söz varlığı çalışmalarında alanyazında mutabık kalınmış bir birimlendirme yaklaşımının olmadığı görülmektedir. Ancak Baş'ın (2011) da belirttiği gibi söz varlığı çalışmaları emek yoğun çalışmalar olup, bu çalışmalar sonucunda elde edilen sonuçların yaygınlaşmaması veya karşılaştırılabilir olmaması ciddi bir problemdir.

Bir diğer önemli konu kolay kullanılır ve erişilebilir bir birimlendiricinin kullanıcılara sunulmasıdır. TS Tokenizer bir Python betiği içine çağrılarak kullanılabileceği gibi Linux platformunda doğrudan Linux kabuğu üzerinde, Windows işletim sisteminde Komut İstemcisi veya PowerShell üstünde kullanılabilir. Bu yönüyle, yüksek seviyede kod okuryazarlığı gerektirmeden kullanıcıların birimlendiriciyi kullanması amaçlanmıştır.

Son olarak birimlendiriciye bir metin temizleme sınıfı eklenerek, özellikle karakter kodlaması sorunlarının bertaraf edilmesi sağlanmıştır.

Araştırma Problemi

Araştırma problemi “Türkçe bir metnin Türkçenin yapısına uygun bir şekilde birimlendirilmesi için kullanılabilecek verimli bir işlemler dizisi nasıl tasarlanabilir?” olarak belirlenmiştir.

Alt Problemler

Çalışmanın alt problemleri şu şekildedir:

1- Bir metnin birimlendirilmesinde işlem basamakları arasında nasıl bir hiyerarşi oluşturulmalıdır?

2- Türkçe söz varlığı araştırmalarının standartlaştırılmasına birimlendiricinin katkısı nasıldır?

Sayıltılar

Çalışmada kullanılan veri setinin (bkz. Veri Toplama Süreci) Türkçenin sözcük çeşitliliğini yansıttığı kabul edilmiştir.

Sınırlılıklar

Bu çalışmada birimlendirme işlemine alınacak girdi iki boşluk karakteri arasında kalan diziler olarak belirlenmiş, boşluk karakterini de içerecek şekilde birden fazla birimden oluşan ve alan yazında “çok birimli sözcükler” (multiword expressions) olarak adlandırılan yapılar çalışma kapsamına alınmamıştır.

Tanımlar

Birim: Bir metnin tanımlanan hedef uyarınca ayrıştırılmış ve işlenmeye/kullanıma hazır hale getirilmiş parçacığı.

Birimlendirme: Metnin hedeflenen amaçlar doğrultusunda birimlere ayrıştırılması işidir.

Birimlendirici: Birimlendirme işlemi gerçekleştirmek üzere tasarlanmış betik veya uygulama.

Bilişimsel (Hesaplamalı) Dilbilim: Bilgisayar ve istatistiki yöntemler kullanılarak yapılan, doğal dile ilişkin örüntüleri keşfetmeyi, dilin biçim, sözdizim vb. katmanlarında çözümlemeler yapmayı konu alan çalışma alanı.

Doğal Dil İşleme: Yapay zekâ çalışmalarının bir alt disiplinidir. Bilgisayarlar tarafından insan dillerinin anlaşılabilmesini ve üretilebilmesini konu alır. Temel hedefi insan bilgisayar arasında doğal dil ile iletişim sağlanmasıdır.

Derlem: Belirli ölçütlerle bir araya getirilmiş, sayısallaştırılmış, görece büyük hacimli dil verisi bütünleridir.

Derlem Dilbilim: Dil derlemleri kullanılarak yapılan dilbilimsel çalışmaları konu alan disiplindir.

Betik: Bir programlama dili ile hazırlanmış, grafik arayüzü olmayan, ilgili programla dilinin yorumlayıcısı ile doğrudan çalıştırılabilen yazılımlar.

Veri Madenciliği: Veri içinde yer alan anlamlı örüntülerin belirlenmesini hedefleyen çalışmalar.

Python: Guido Van Rossum tarafından, 1991 yılında geliştirilmiş, nesne yönelimli programlama dilidir.

Bash: Unix, GNU/Linux ve benzeri sistemlerde kullanılan komut satırı kabuğudur.

Bölüm 2

Araştırmanın Kuramsal Temeli ve İlgili Araştırmalar

İlk adımlar: Makine Çevirisi

Mekanik sözlükler kullanılarak insanlar arası dil bariyerinin aşılmasına ilişkin ilk fikirler 17. yüzyılda ortaya atılmış ancak kayda değer ilk ürünler 20. yüzyıla kadar ortaya çıkmamıştır (Hutchins, 1986). "Çeviri makinesi" terimi ilk olarak, 1933 yılında George Artsrouni'nin ve 1939 yılında Petr Smirnov Troyanski'nin birbirlerinden bağımsız olarak aldıkları patent adlarında görülmüştür (Hutchins & Lovtskii, 2000; Johri ve diğerleri, 2021). Artsrouni'nin 1937 yılında sunduğu prototip makine, bir çeviri cihazından çok diller arası sözcük karşılıklarını veren mekanik bir sözlük olarak tanımlanırken Troyanski'nin[1] önerdiği makine ise çağının ötesinde bir yapıda olan ve patent adından da anlaşılabileceği üzere çok dilli tercüme yapısını hedefleyen bir cihazdır (Hutchins, 2004; Korkmaz, 2019).

Bilgisayar ile insan dillerini işleyen çalışmaların geçmişi, 1950'lere kadar uzamaktadır (Ferrari, 2004). Alandaki ilk çalışmalar makine çevirisi üzerine yapılmıştır (Jones, 1994). 1967 tarihli çalışmasında 1940'ların başını işaret eden Melchuck, o dönemde otomatik çeviri ifadesinin kimse için anlamlı bir karşılığının olmadığını, ancak konunun özellikle son 10 yılda popüler hale geldiğini dile getirmektedir (Melchuck, 1967).

Almanlar tarafından İkinci Dünya Savaşı sırasında kullanılan bir şifreleme makinesi olan Enigma'yı kırmayı başaran Alan Turing ve ekibi, birincil amaçları bu olmasa da, makine çevirisinin temellerini oluşturmuşlardır (Stein, 2018). Turing'in bu başarısı alanda bir dinamizm yaratmıştır. 1948 tarihli "Matematiksel İletişim Teorisi" (A mathematical theory of communication) başlıklı çalışmasında Shannon, iletişimin sinyallerin istatistiksel

[1] Troyanski'nin aldığı patent adı şöyledir: "Bir dilden diğerine veya aynı anda birkaç başka dile çeviri yaparken kelimeleri seçmek ve yazmak için kullanılan bir makine" (*A machine for selecting and typing words when translating from one language into another or several others simultaneously*). Hutchins'in 2004 tarihli "Makine çevirisinin iki öncüsü: Artsrouni ve Troyanski" başlıklı makalesinde özellikle Troyanski'nin önerdiği cihaza ilişkin detaylı bilgi verilmektedir.

olarak modellenerek çözümlenebileceğini öne sürmüştür (Shannon, 1948). Shannon'ın kısa sürede popülerleşen savı makine çevirisi konusunda ilham verici olmuştur (Nwagwu, 2022).

Farklı kaynaklarda alanın başlangıcı olarak Warren Weaver'in 1949 tarihli makine tabanlı çevirinin olasılığını öneren bildirisi verilmektedir (King, 1984; Kay, 2003; Camburn, 2013). Warren Weaver ve Andrew Booth 1949 yılında, yalnızca üç yıl önce bulunmuş olan bilgisayarların, çeviri yapmak için kullanılabileceğini önermişlerdir (Melby, 2019). Weaver, 2. Dünya savaşı döneminde kullanılan kriptografi teknikleri ve istatistiki metodların kullanılmasıyla dilin altında yatan işleyişin keşfedilebileceğini ve bu yolla makineler tarafından çeviri yapılabileceğini öne sürmüştür (Weaver, 1949).

Makine çevirisi konulu ilk konferans 1952 yılında yapılmış ve alana ilişkin olumlu bir izlenim oluşmasını sağlamıştır (King, 1984). 1954 yılında "Makine Çevirisi" (Mechanical Translation) adlı bilimsel bir dergi yayınlanmaya başlanmış ve 1965 tarihinde bu derginin adına "bilişimsel dilbilim" alt başlığı eklenerek "Makine Çevirisi ve Bilişimsel Dilbilim" (Mechanical Translation and Computational Linguistics) olarak yayına devam etmiştir (Muslim, 2007). Bu dergi, 1980 yılından itibaren de "*Computational Linguistics*" adıyla aktif olarak yayınlanmaya devam etmektedir.

Ancak, makineler ile çeviri yapmanın başlangıçta sanıldığından daha zorlu bir çalışma konusu olduğu kısa sürede ortaya çıkmıştır. Pérusse, yıllar sonra, Georgetown Üniversitesinde 12 yıl boyunca devam eden, yirmi milyon Amerikan Dolar harcama yapılan ve sonuç alınamayan çalışmaları kastederek konunun nasıl ele alındığını şu sözlerle ifade eder: "Sonuçta diferansiyel denklemle karşılaştırıldığında metafor nedir?" (Pérusse, 1983, s.6).

Josselson, makine çevirisi çalışmalarının ilk dönemlerde dili yalnızca bir "sözcük demeti" olarak gördüğünü ve başlangıç döneminde yapılan çalışmaların temel amacının yeteri büyüklükte bir sözcük dağarcığını tutacak bir yapı kurmak olduğunu söylemektedir (Josselson, 1971). Ancak hem kendi bütünselliği içinde dilin hem de bulunduğu bağlamda

sözcük ve sözcük öbeklerinin oluşturduğu yapıların yüzeyde görünenden daha fazla bilgi içerdikleri bilinmektedir. Firth'ün meşhur cümlesinde de belirttiği gibi, bir sözcük ancak içinde bulunduğu bağlamda ve birliktelik kurduğu diğer sözcüklerle birlikte bilinebilir (Firth, 1957). Bu konuya dikkat çeken Manning, o dönemde karşılaşılan zorlukları değerlendirirken, kullanılan veri setlerindeki cümlelerin kendi sınırları içinde kalması (ön/art gönderimlerin işlem dışında kalması), her dil çifti için özgün veri ihtiyacı ile eş seslilik ve çok anlamlılık gibi sözcük bazındaki problemlere işaret etmektedirler (Manning & Schütze, 1999).

1960 tarihli çalışmasında makine öğrenmesinde gelinen durumu değerlendiren ve geleceğe yönelik bir tahminde bulunan Bar-Hillel birkaç yıl içinde ticari bir ürün olarak kısmen mekanize çeviri merkezlerinin pratik bir gerçeklik haline gelme şansının ciddi olduğu bir noktaya geldiğini dile getirmiştir. Ancak, henüz doğrudan kağıttaki metni bilgisayar ortamına aktaran "mekanik yazı okuyucuların" olmadığı, (makine çevirisine) en uygun biçimde kullanılabilir sözlük/sözlüklerin hazırlanmasının emek yoğun bir iş yükü olduğunu da not düşmüştür (Bar-Hilel, 1960).

Bilgisayarlar ile dilin işlenmesi ve çeviri yapılması konusunda esmekte olan olumlu hava ALPAC raporu ile sekteye uğramıştır. Amerika Birleşik Devletleri Ulusal Bilimler Akademisi tarafından desteklenen "Otomatik Dil İşleme Danışma Komitesi" (Automatic Language Processing Advisory Committe) tarafından yazılan 1966 tarihli ALPAC raporu alanın gelişim sürecindeki önemli noktalardan biri olmuştur (Akbari, 2014; Garje & Kharate, 2013; Hutchins, 2003). Raporda, o döneme kadar yapılan ve devlet tarafından da finansal olarak desteklenen "makine çevirisi" çalışmalarının kaydettiği ilerleme değerlendirilmiş ve gelinen noktada yapılan çalışmaların beklenen sonucu üretememesi sebebiyle finansal desteğin kesilmesi önerilmiştir (Manaris, 1998). ALPAC raporunda geçen şu cümleler dikkat çekicidir: "Elektrikli kaldırımlar ve uzay yolculuğu gibi makine çevirisi de hemen köşede değil. Ve aynı fikirde olmayanlar elektronik 'beyni' başka yöne çevirebilirler." (ALPAC, 1966). Rapor her ne kadar makine çevirisi çalışmalarına aktarılan

kaynağın büyük oranda kesilmesine sebep olmuşsa da bilgisayar ile dil işleme çalışmalarına farklı açılardan devam edilmesini de önermiştir. Raporun yayınlanmasından sonra makine çevirisi alanında yapılan çalışmalar akademi tarafından büyük oranda terk edilmiş olsa da bazı gruplar bu konudaki çalışmalarına devam etmiştir (Pierce & Carroll, 1966).

Elbette dönemin teknolojik sınırlılıklarının da başarım üstünde etkisi olmuştur. Plath, 1950'lerde en gelişmiş makineler ve en yeni algoritmalarla bile uzun bir cümlenin işlenmesinin 7 dakikaya kadar çıkabildiğini belirtmektedir (Plath, 1967). Hays raporu değerlendirirken, bir zamanlar üyesi olduğu komitenin yazdığı ALPAC raporunun, dönemin dile ilişkin bilgisi ve hesaplama yetkinlikleri göz önünde bulundurulduğunda ağır bir fatura kesmek olduğunu dile getirmiştir (Hays, 1969). Jones, o dönemde yapılan makine çevirisi çalışmaların beklenen başarıyı gösterememesinin temel nedeni olarak sözlük tabanlı sözcükten sözcüğe çeviri yöntemlerin yetersiz kalmasını ileri sürmektedir (Jones, 1994).

Bilişimsel Dilbilim ve Doğal Dil İşleme

İngilizce "Computational Linguistics" terimi Türkçe literatürde "hesaplamalı dilbilim", "bilgisayarlı dilbilim", "berimsel dilbilim" veya "bilişimsel dilbilim" karşılıkları ile kullanılmaktadır. Bu çalışmada hem Sankur'un İngilizce-Türkçe Ansiklopedik Bilişim Sözlüğünde (Sankur, 2004) hem de Türkiye Bilişim Derneğinin *Bilişim Sözlüğünde*[2] "computational linguistics" karşılığı olarak verilen "bilişimsel dilbilim" kullanımı benimsenmiştir.

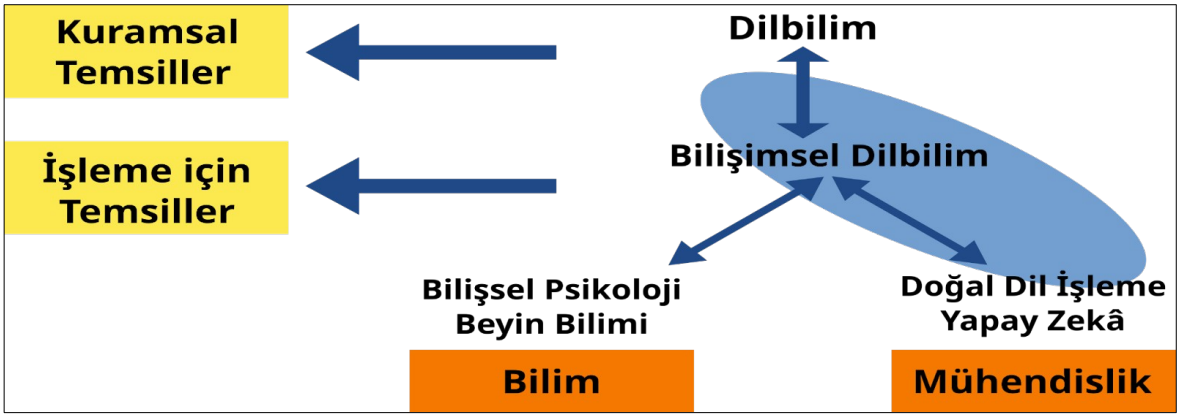
BD ve DDİ çoğu zaman birbiriyle yakından ilişkili ve belirli noktalarda örtüşen çalışma alanlarıdır. Çalışma konuları ve çıktıları bakımından da benzer odaklara yönelmektedirler. İki kavram arasındaki ayrım özellikle günümüzde iyice bulanıklaşmış olsa da tarihsel bağlamda çok da keskin olmayan bazı farklılıkları olmuştur.

[2] Türkiye Bilişim Derneği İngilizce-Türkçe bilişim terimleri sözlüğü. (Erişim: 23 Mart Mayıs 2024, https://eski.tbd.org.tr/index.php?sayfa=sozluk)

Grishman (1986) genel bir bakışla BD'yi doğal dili anlama ve üretme çalışmaları olarak tanımlar. Hausser (2013), BD'yi daha somut bir şekilde ele alarak insan-makine arasında doğal dilde iletişimi mümkün kılacak sistemlerin geliştirilmesine olanak tanıyan bilişsel makinelerin kurulmasına yönelik işlevsel bir dil kuramı olarak ele alır. Schubert BD'nin yazılı ve sözlü dili bilişimsel bir bakış açısından anlamakla ve dili etkili bir şekilde işleyip üretebilen ürünler geliştirmekle ilgilenen bir mühendislik disiplini olduğunu söyler (Scuhebert, 2020). Tsujii (2021) ise BD'yi, dilsel kuralların biçimsel ve bilişimsel olarak tanımlanması olarak değerlendirir ancak bu alanı dilbilimin bir alt disiplini olarak konumlandırır. DDİ ise alanyazında yapay zekâ çalışmalarının bir alt alanı olarak tanımlanmaktadır. Jurafsky (Jurafsky & Martin, 2025) tanımı yaparken DDİ'nin insan dillerinin bilgisayarlar tarafından anlaşılması, analiz edilmesi, üretilmesi ve dönüştürülmesine yönelik algoritmaların geliştirilmesini konu alan yapay zekâ alt dalı olduğunu söyler. Nitekim Doğal Dil Anlama (Natural Language Understanding, NLU), Doğal Dil Üretme (Natural Language Generation, NLG) gibi çalışma alanları verilen tanımla birebir örtüşmektedir.

Şekil 7

Bilişimsel dilbilim ve doğal dil işleme ilişkisi (Tsuji, 2021'den uyarlanmıştır.)



Şekil 7'de verilen ilişkisel yapı incelendiğinde, BD'in daha çok doğal dilin yapısına ve işlevine ilişkin bilimsel sorulara, yani "neden?" sorusuna odaklanırken, DDİ elde edilen kuramsal bilgiyi uygulamaya geçirmeye, yani "nasıl?" sorusuna yanıt bulmaya yönelen bir

mühendislik yaklaşımıdır (Luz, 2022; Tsujii, 2021). Başka bir deyişle, BD'de dil olgusunu anlama ve açıklama ön plandayken, DDİ'de bu bilgiyi kullanarak belirli görevleri yerine getiren sistemlerin tasarlanması ön plandadır.

Doğal dil işlemeye yönelik çalışmalarda genellikle belirli bir göreve odaklanan uygulamalar veya uçtan uca tasarlanmış yazılım zincirleri geliştirilir. Bu sistemlerin başarımı da yine mühendislik bakış açısıyla örneğin doğruluk (accuracy), F1 skoru veya hata oranı (error rate) gibi nicel ölçütlerle deneysel olarak test edilir (Jurafsky & Martin, 2025). Buna karşılık, BD araştırmalarında belirli bir dilsel olgunun derinlemesine incelenmesine yoğunlaşılır ve elde edilen kuramsal modellerin dile uygunluğu ve dili ne ölçüde açıklayabildiğinin değerlendirilmesi öne çıkar.

Kısacası, BD, doğal dilin kural ve yapılarının bilgisayar ortamında modellenmesi yoluyla bilgisayar bilimleri ile dilbilim arasında bir köprü kuran; aynı zamanda dilin yazılı ve sözlü biçimlerinin işlenmesine dair kuramsal bir altyapıyı da içeren bir disiplindir. Sonuçta, Steedman'ın (2008) da belirttiği gibi dil insanlığın sahip olduğu bilgi dil ile ifade edilir ve bu sebeple bilişimsel dilbilim (ve DDİ) önemli çalışma alanlarıdır.

Metin İşlemenin İlk Adımı

Bu bölümde sözcük ve birim kavramları çalışmanın hedefleri çerçevesinde tartışılmıştır.

Sözcük ve Birim

Sözcük kavramı, sezgisel olarak kolaylıkla tanımlanabilir gibi görünse de kuramsal bağlamda kesin ve evrensel bir tanım geliştirmek oldukça güçtür. Yalnızca sezgisel bilgiye dayanarak "herkes bir sözcüğün ne olduğunu bilir" denilemeyeceği araştırmacılar tarafından dile getirilmiştir (Carter, 2012, s.20; Hockett, 1987, s.3). Nitekim Haspelmath (2023) "sözcük" kavramının tüm dillerde evrensel bir biçimde tanımlanamayacağını, bu kavramın farklı dillerde çok çeşitli yapılarda ortaya çıktığını ifade eder ve bu sebeple

sözcüğü dilbilimsel evrensellikten çok işlevsel uygunluk üzerinden ele alır. Sözcük tanımının yapılabilmesi için dilin bu yapısal ve işlevsel özelliklerinin dikkate alınması gereklidir. Alanyazında yapılan tanımlar yaklaşıma göre farklılık göstermekle birlikte sözcüğü temelde biçimsel (*şekilsel*) ve anlamsal olarak iki düzlemde ele alır (Kurudayıoğlu & Karadağ, 2005).

Uzun (2004) yazılı dile odaklanıldığında sözcüğün her iki ucunda bir boşluk karakteri içeren dil birimleri olarak tanımlanabileceğini ancak bu tanımın sözlü dili kapsayamacağını söylemektedir. Yine yazılı dile atıfla Carter (2012) sözcüğü bir boşluk ya da noktalama işaretiyle sınırlandırılmış, harflerden oluşan bir dizi olarak tanımlamıştır. Carter'ın tanıma eklediği noktalama işaretlerine özellikle bu çalışmanın odağı gereğince dikkat etmek gereklidir. Öte yandan birden fazla sözcüğün bir araya gelerek oluşturduğu, tek bir varlığa/kavrama işaret eden sözcükler için "boşluk" üzerinden bir sınır çizmek yeterli değildir. "New York" gibi, alanyazında "çok birimli sözcükler" (multiword expressions, MWE) olarak anılan bu yapılar kendi içlerinde bir boşluk karakteri barındırmaktadır.

Ancak bu çalışma kapsamında amacımız genel bir sözcük tanımına ulaşmak değildir. Çünkü metinlerin birimlendirilmesi yalnızca sözcüklerin değil metni oluşturan her türlü bileşenin ayrıştırılması işidir. Bu bağlamda, Evert'in (Evert 2005, s.18) sözcük tanımı yaparken "temeldeki teoriye veya amaçlanan uygulamaya bağlı olarak her türlü anlamsal (*lexical*) ögeyi işaret edecek genel bir terim" şeklinde belirlediği çerçeve çalışmamız için işlevsel bir zemin sağlamaktadır. Bu çerçeve, bir metni oluşturan her türlü karakter dizisinin (noktalama işaretlerinden oluşan yapılar, rakamlar ya da karışık biçimlerdeki diziler dahil) analiz açısından eşdeğer biçimde ele alınabilmesini mümkün kılmaktadır.

Türkçe alanyazında İngilizce "token" ve "tokenization" karşılığı olarak "bölüt", "bölütleme" (Çetiner ve diğerleri, 2021), "dizge", "dizgeciklere ayırma" (Tunalı & Bilgin, 2012) gibi terimler kullanılmaktadır. Bu çalışmada önerilen ve kullanılan terminoloji ise

“***birim***”, bu işi yapan araç adı olarak “***birimlendirici***” ve birimlere ayırma işi için de “***birimlendirme***” olacaktır.

Çalışmanın kapsamı doğrultusunda, özellikle BD ve sosyal bilimler bağlamında sözcük ve birim ayrımının daha yakından ele alınması gerekmektedir. Alanyazında kullanılan “dilin yapı taşı sözcüklerdir” ifadesi (Delioğlu & Şen, 2022; Erat, 2022), dilin işleyişine ilişkin genel bir sezgiyi yansıtmak ile birlikte, BD açısından sınırlı ve geçerli olmayan bir çerçeve sunmaktadır. Bu sezgisel yaklaşım, her sözcüğün doğrudan olarak bir birim kabul edileceği varsayımına dayanır. Yalnızca sözcük temelli bir birim anlayışı, doğal dilin bilgisayar ortamdaki temsilini ve işlenmesini tam olarak yansıtamamaktadır. Bu nedenle, birim kavramının tanımı daha kapsayıcı bir çerçevede ele alınmalı ve sözcük dışındaki unsurların da metnin kurgusundaki işlevleri göz önünde bulundurulmalıdır. Pratikte sözcük eşittir birim eşitliği tersten de çalışmak zorunda olduğu için, yani birim eşittir sözcük denkleminin de doğru olmasını gerektirdiği için çalışmayacaktır.

Örneğin, Türkçe sözlükte “arabacı” sözcüğü madde başı olarak yer almaktadır. Türkçe konuşan biri bu sözcüğün farklı çekim durumlarını (*arabacının*, *arabacılar*) sahip olduğu dilbilgisine başvurarak kolaylıkla anlayacaktır. Burada dikkat edilmesi gereken nokta, sözcüğün çekimli hallerinin bu kökle ilişkili olan anlamını koruduğudur. Dolayısıyla sözcüğün türleri (*type*) metin içinde ama yine birer sözcük olarak var olmaktadırlar. Öte yandan, @kullanici_adi gibi sosyal medyaya özgü bir kullanım olan mention (bahsetme/kullanıcıyı işaret etme) @ işaretinden sonra (bazı teknik sınırlılıklarla) gelebilecek her türlü karakter dizisi ile oluşturulabilmektedir. Bu tarz bir kullanımın bir sözlükte listelenmesi farklı türlerinin (type) ancak işlevsel bakımdan tanımı yapılabileceği için mümkün değildir. Örneğin @kullanici_bir ve @kullanici_iki aynı işlevi yerine getiren dizilerdir fakat bu diziler dilin söz varlığının birer ögesi olamaz. Fakat, sosyal medyanın bir parçası olan ve milyarlarca kullanıcı tarafından genel bir uzlaşı ile kullanılan bu dizilerin yok sayılması da mümkün değildir. Kendi bağlamında metinsel bütünlüğün oluşturulmasına katkı sağlayan ve bir işlevi yerine getiren bu yapıların, eğer sosyal

medyayı hedef alan bir birimlendirici tasarlanıyorsa tasnif dışı bırakılmaması gerekmektedir. Birimlendiricilerin bu tarz birim olabilen ancak sözcük olmayan rakamlar, noktalama işaretleri, smiley işaretleri, emojiler, satır sonu ve paragraf boşluğu gibi karakterleri de işlemesi gereklidir.

Sonuç olarak, bu çalışmada geliştirilen birimlendirici, aşağıda belirtilen iki temel ilkeye dayalı olarak yapılandırılmıştır:

1- Her sözcük bir birimdir ancak her birim bir sözcük değildir.

2- Analize uygun hale getirilmiş ve dilsel işlevi/değeri olan her karakter dizisi, bu çalışma kapsamında bir "birim" olarak değerlendirilecektir.

Birimlendirme

Erken dönemde, özellikle metinden bilgi çıkarımı bağlamında, birimlendirme bir metnin sözcüklerden oluşan indeks terimlerine bölünmesi ve noktalama işaretlerinin kaldırılması işlemi olarak değerlendirilmiştir (van Rijsbergen, 1979; Salton, 1989). Bu dönemde birimlendirme metnin analizi sürecinden önce yerine getirilmesi gereken zorunlu bir aşama olarak kabul edilmiştir.

Webster ve Kit (1992), birimlendirmenin DDİ sürecinin ilk adımı olduğunu belirtir ve birimleri, daha sonraki aşamalarda daha fazla bölünmesi gerekmeyen temel atomik diziler olarak tanımlarlar. Birimlendirmeyi, karakter dizilerinden oluşan cümleleri sözcük dizilerine dönüştüren bir eşleştirme işlemi olarak tanımlayan Guo (1997), sözcük sınırlarının belirlenmesinin DDİ'nin ilk adımı olduğunu vurgular. Grefenstette ve Tapanainen (1994) birimlendirmeyi *sözcük-benzeri* dizilerin metinden ayrıştırılması süreci olarak tanımlar ve birimlendirme sonrasında **(i)** noktalama işaretleri, tarihler, sayılar gibi anlaşılabilir bütünler oluşturan diziler ve **(ii)** biçimbirimsel analize gönderilebilir yapıların elde edildiğini söyler. Kaplan'a (2005) göre birimlendirme, bir metnin anlam taşıyan ve belirgin parçalarına ayrıştırılması işlemidir.

Görüldüğü üzere tanımlar ve birimlendiricilerin işlevi üstünde genel bir uzlaşı olmakla birlikte, birimlendirme konusundaki farklılaşmalar **(i)** hangi dizilerin **(ii)** hangi amaçla “birim” olarak kabul edileceği çerçevesinde şekillenmiştir.

Ancak, yukarıda da belirtildiği gibi (bkz. *Problem Durumu*) özellikle 2010 ve sonrasında yapay zekâ çalışmalarının hız kazanmasıyla birlikte DDİ alanında birimlendirme metinlerin analize, işlemeye gönderilmesinden daha çok dil modelleri eğitmek için ham metnin ön işleme sürecine atıfla ele alınmıştır (Budur ve diğerleri, 2020; Tokgöz ve diğerleri., 2021). Hopton’un birimlerin işlevsel ve içeriksel ayrımının nöral ağların yaygın kullanımı öncesinde DDİ için önemli olduğunu tespiti de bu görüşü desteklemektedir (Hopton, ve diğerleri, 2025).

Bu durumun bir sonucu olarak, bu amaçla hazırlanan birimlendiricilerin ürettiği çıktılar eğitim bilimleri, dilbilim, edebiyat çalışmaları, uygulamalı dilbilim, derlem dilbilim vb. sosyal bilimler disiplinleri için kullanılabilirliklerini yitirmiştir. Dolayısıyla DDİ aşağıda inceleyeceğimiz geleneksel birimlendirme yaklaşımlarını büyük oranda terk etmiş, dil modellerinin eğitiminde kullanılacak biçimde en yüksek verimi üretecek birimlendirme algoritmaları ve yaklaşımlarını kullanmaya yönelmiştir.

Birimlendirme Yaklaşımları

Birimlendiriciler, verilen metin üzerinde belirli bir hiyerarşiyle uygulanan ardışık işlemler dizilerini işleterek çalışır. Bu işlemler metindeki yapısal ipuçlarını (örneğin boşluklar, noktalama işaretleri), sözlük kaynaklarını, düzenli ifadeleri ya da istatistiksel modelleri kullanarak metni birimlendirmeyi hedefler. Söz konusu işlemler kullanılan yaklaşıma göre farklılaşmaktadır. Üretilen çıktılar da uygulama amacına göre çeşitlilik göstermektedir.

Boşluk Tabanlı Birimlendirme

Boşluk tabanlı birimlendirme (whitespace tokenization), doğal dil işleme sistemlerinde en basit ve en yaygın kullanılan birimlendirme yöntemidir. Bu yaklaşım, sözcükleri boşluk karakteriyle ayrılan diziler olarak ele alır ve metni boşluklardan bölerek birimlendirir. Özellikle sözcük sınırlarının boşluklarla belirginleşmiş olduğu dillerde bu yöntem kolaylıkla uygulanabilir. Ancak bu yaklaşım, çok birimli ifadeler (örn. “New York”), noktalama işaretleriyle biten sözcükler ya da tireleme içeren dizilerin birimlendirilmesinde yetersiz kalabilmektedir.

Bu yaklaşımın geliştirilmiş bir biçimi noktalama işaretlerinin önüne ve ardına birer boşluk eklemek, birden fazla ardışık boşluk karakterlerini bir boşluk karakteri ile değiştirmek ve sonrasında boşluk karakterini baz alarak girdiyi birimlendirmek şeklinde kullanılmaktadır. Bu yaklaşımla dizinin sınırında bulunan noktalama işaretleri ayrı birer birim olarak ayrıştırılabilirken “e-posta” gibi bünyesinde noktalama işareti içeren diziler “e”, “-”, “posta” olarak parçalanmakta, @mention ve #hashtag benzeri yapılar ise bütünlüklerini kaybetmektedir. Zeman (2018) bu yaklaşımı alt-düzey (low-level) olarak tanımlamaktadır.

Aşağıda boşluk tabanlı birimlendirme için üç ayrı örnek verilmiştir.

Kodların ürettiği çıktı sırasıyla kod1, kod2 ve kod3 olarak Tablo 1’de verilmiştir.

Kod 1 hiçbir ek kütüphane kullanmadan yalnızca boşluk karakterinin yeni satır karakteri ile değiştirilmesi prensibiyle çalışmaktadır. Kod 2 ve Kod 3 yerleşik Python kütüphanelerinden faydalanmaktadır.

Şekil 8

Boşluk karakterinin değiştirilmesi ilkesiyle çalışan kod örneği (kod 1)

```
cumle = "Defne kahvaltısını yaptı, sonra okula gitti."
ws_tokenized = cumle.replace(" ", "\n")
print(ws_tokenized)
```

Şekil 9

String kütüphanesinden yararlanarak çalışan kod örneği (kod 2)

```
    import string
    cumle = "Defne kahvaltısını yaptı, sonra okula gitti."
    birimler = []
    birim = ""
    for karakter in cumle:
       if karakter in string.punctuation:
           if birim:
               birimler.append(birim)
               birim = ""
# Noktalama işaretini ayrı birer token olarak listeye ekiyoruz
           birimler.append(karakter)
       elif karakter.isspace():
           if birim:
               birimler.append(birim)
               birim = ""
       else:
           birim += karakter
    if birim:
       birimler.append(birim)
    print("\n".join(birimler))
```

Tablo 1

Örnek kodların ürettiği çıktılar

Kod 1	**Kod 2**
	Defne
Defne	kahvaltısını
kahvaltısını	yaptı
yaptı,	,
sonra	sonra
okula	okula
gitti.	gitti
	.

Tabloda görüldüğü gibi kod 1 “yaptı,” ve “gitti.” dizilerindeki noktalama işaretlerini dizinin bir parçası olarak döndürmektedir. Kod 2’de kullanılan *string.punctuation* tüm temel

noktalama işaretlerini içerir[3]. Her dizi sırasıyla bir noktalama işaretiyle veya boşluk karakteriyle karşılaşıncaya kadar analiz edilir, bu karakterlerden birini bulduğunda ise *tokens* listesine bu dizi eklenir. Görüldüğü üzere kod 2 noktalama işaretlerini dizilerden ayrıştırılmıştır. Boşluk tabanlı yöntemler, yüksek hata toleransı gerektirmeyen uygulamalarda ön işleme adımı olarak tercih edilse de, daha karmaşık birimlendirme görevleri için tek başına yeterli değildir.

Sözlük Tabanlı Birimlendirme

Sözlük tabanlı birimlendirme (lexicon-based tokenization), metin içerisindeki dizilerin önceden tanımlanmış bir sözcük listesi ile karşılaştırılması esasına dayanan bir birimlendirme yaklaşımıdır (Mills, 1998). Bu yaklaşımda, verilen metin karakter karakter veya sözcük sözcük taranarak, her bir dizinin sözlükte yer alıp almadığı kontrol edilir. Bulunan eşleşmeler bir birim olarak kabul edilir. Bu yaklaşım, özellikle kelime sınırlarının açık olmadığı ve boşluk karakterinin sözcük ayracı olarak işlev görmediği dillerde (örneğin Çince, Japonca gibi) sözcüklerin ayrıştırılması amacıyla kullanılmıştır (Wong & Chan, 1996).

Sözlük tabanlı yöntemlerde başarım, büyük ölçüde kullanılan sözlüğün kapsamına ve güncelliğine bağlıdır. Nitekim sözlükte yer almayan sözcükler, özel adlar veya yazım yanlışları söz konusu olduğunda yöntem yetersiz kalmaktadır. Harflerden oluşmamış diziler söz konusu olduğunda ise çeşitli kurallar tanımlanarak başarım arttırılabilmektedir.

Maksimum eşleştirme (maximum matching) bu yaklaşımda en sık kullanılan algoritmadır. Bu algoritma, metnin ilk karakterinden başlayarak adım adım ilerler ve ilgili dizi için en uzun eşleşmeyi arar. Bulunan en uzun eşleşme bir birim olarak kabul edilir. Aynı işlem metin sonuna kadar devam eder. Bu algoritmanın türevleri olan ters maksimum eşleştirme (reverse maximum matching) veya çift yönlü eşleştirme gibi yöntemler de literatürde yer almaktadır (Almaaytah, 2024).

---

[3] !"#$%&'()*+,-./:;<=>?@[\]^_`{|}~

Kural Tabanlı Birimlendirme

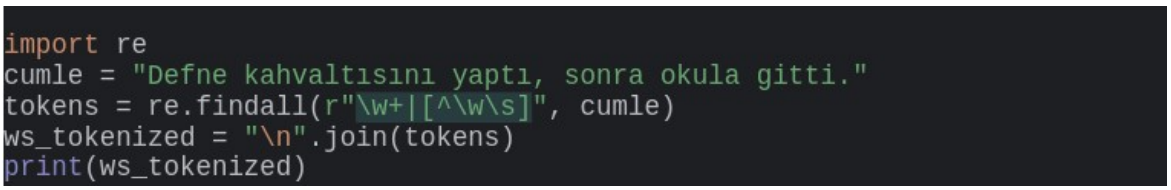
Kural tabanlı birimlendirme (rule-based tokenization), belirli dil kuralları ve yapısal örüntüler doğrultusunda oluşturulmuş düzenli ifadelerin kullanılması ile metnin birimlendirilmesi yaklaşımıdır. Düzenli ifadeler, belirli bir karakter örüntüsünü tanımlamak için kullanılan özel söz dizimine sahip yapılardır. Veri madenciliği, bilgi çıkarımı, doğal dil işleme ve arama motorları gibi pek çok alanda, belirli yapıları tanımak ve işlemek amacıyla kullanılırlar. Kural tabanlı birimlendirme yakalanması hedeflenen diziler için bir düzenli ifade/ifadeler oluşturularak bu yapıların, metindeki noktalama işaretleri, rakamlar, özel karakterler, tarih biçimleri, e-posta adresleri, URL'ler, @mention ya da #hashtag gibi yapıları tanımlaması beklenir. Özellikle yapı bakımından tutarlı olan biçimler üretilen kurallar ile verimli şekilde tespit edilir.

Bu yaklaşımın avantajlı yönlerinden biri de denetlenebilir oluşudur. Kuralların tanımlanmış olması sayesinde geliştirici veya dilbilimci, hangi girdinin neden belirli bir şekilde birimlendirildiğini kolaylıkla anlayabilir. Bu şeffaflık, kural tabanlı sistemleri özellikle dilsel analiz ve eğitim amaçlı uygulamalarda değerli kılar.

Ancak kural tabanlı sistemlerin belirli sınırlılıkları da vardır. Her dilin yapısal karmaşıklığı, istisnaları ve sürekli değişen kullanımı göz önüne alındığında, tüm olası durumları kapsayan kuralları önceden tanımlamak zordur. Bu nedenle kural tabanlı yaklaşımlar da genellikle diğer yaklaşımlar gibi başka yaklaşımlarla bir arada kullanılmaktadır.

Şekil 10

Düzenli ifadeler kütüphanesinden yararlanarak çalışan kod örneği (kod 3)



Tablo 2

Örnek kodun ürettiği çıktı (kod 3)

Kod 3
Defne kahvaltısını yaptı , sonra okula gitti .

Kod 3'ün ürettiği çıktıda da noktalama işaretleri ayrı birer birim olarak döndürülmektedir. Kod 3 düzenli ifadelerden yararlanılarak hazırlanmıştır. \w+ ifadesiyle harf ve rakam içeren diziler, [^\w\s] ifadesi ile de harf, rakam ve boşluk karakteri dışında kalan tüm diziler yakalanarak birimler üretilmiştir.

Sözcük Altı Birimlendirme

Çalışmanın önceki bölümlerinde de ifade ettiğimiz gibi, günümüzde DDİ ve yapay zekâ alanındaki çalışmaların ağırlıkla büyük dil modellerinin eğitilmesi, dili anlama ve dili üretme çalışmalarına yoğunlaşmıştır. Bu sebeple birimlendiricilerden daha çok modelin eğitiminde kullanılacak sözlüklerin hazırlanması amacıyla yararlanılmaktadır. Dil modellerinin eğitiminde kullanılan *birimlendirilmiş veri*, modelin ilişkili parametreleri belirleyeceği en verimli parçacıkları elde etmek için kullanılır. Bu sebeple, boşluk ile sınırlandırılmış diziler, bu modeller için verimli birimler sunmamaktadır. Dolayısıyla

Mielke'nin (DDİ açısından değerlendirerek) birimlendirmenin artık yalnızca metni sözcüklere bölme olmadığı tespiti yerinde ve önemlidir (Mielke ve diğerleri, 2021).

Günümüzde bu alanda en sık kullanılan iki birimlendirme yöntemi Byte-Pair Encoding (Bayt-Çift Kodlama, BPE) (Sennrich ve diğerleri, 2016) ve WordPiece (Sözcük Parçası) (Devlin ve diğerleri, 2019) yöntemleridir (Schmidt, ve diğerleri, 2024).

BPE modeli, ilk olarak Gage (1994) tarafından önerilen karakter çiftlerinin sıklık temelli birleştirilmesine dayanırken, WordPiece modeli Wu vd. (2016) tarafından Google'ın nöral makine çevirisi (Neural Machine Translation) sistemleri için geliştirilmiştir

Bu algoritmalar var olan sözcüklerin daha küçük diziler olarak ifade edilmesi mantığı üstüne inşa edilmiştir. Temel amaç, sınırlı bir birim sayısını kullanarak dilde düşük sıklıkla gözlenen sözcükleri de temsil edebilmek, böylelikle veri seyrekliğinin (data sparsity) önüne geçmek ve daha verimli modelleme olanakları sunmaktır (Sennrich ve diğerleri, 2016).

Byte-Pair Encoding veri sıkıştırma amacıyla geliştirilmiş bir algoritmanın doğal dile uygulanmasıdır (Gage, 1994; Sennrich ve diğerleri, 2016). BPE, karakter düzeyinde başlayarak girdideki dizileri analiz eder ve en sık gözlenen karakter çiftlerini birleştirerek yeni birimler oluşturur. Bu süreç önceden belirlenmiş bir sözlük hacmine ulaşıncaya kadar sürdürülür ve sonunda oluşturulan birimler, modelin eğitim sırasında kullanacağı sözlüğü oluşturur. BPE yaklaşımı GPT4, Llama3, DeepSeek v3 ve Mixtral gibi büyük dil modellerinin eğitiminde de kullanılmış birimlendirici algoritmasıdır (Wegmann ve diğerleri, 2025).

Sözcük parçası (WordPiece), sözcükleri daha küçük alt birimlere ayıran bir sözcük-altı birimleme (subword tokenization) yöntemidir. WordPiece algoritması, ilk olarak konuşma tanıma sistemleri için geliştirilmiştir (Schuster & Nakajima, 2012). BERT (Devlin ve diğerleri, 2019) gibi modern dil modellerinin temel bileşeni haline gelmiştir. WordPiece, sıklık bilgisine dayalı olarak karakter ve karakter dizilerini iteratif biçimde analiz eder. Bu

işlem sırasında her adımda en yüksek birleşme olasılığına sahip olan çift seçilir ve bu çift yeni bir birim olarak sözlüğe eklenir.

Sözcük altı birimlendirmenin dil modellerinin eğitilmesine olan olumlu katkısı, yalnızca sık kullanılan sözcüklerin temsilini değil aynı zamanda modelin daha önce karşılaşmadığı ya da türetilmiş kelimelerin de model tarafından işlenebilmesini mümkün kılmasında yatmaktadır. Sözcük altı birimlendirme yaklaşımının özellikle metin üretimi içeren görevlerde modellerinin performansını olumlu yönde etkilediği gösterilmiştir (Gallé, 2019; Zouhar ve diğerleri, 2023; Goldman ve diğerleri, 2024).

Bu yaklaşımlar, sabit büyüklükte bir sözlük kullanarak açık kelime dağarcıklı, morfolojik olarak çekimli yapıların daha fazla gözlendiği dillerde bile yüksek performans sunulabilmektedir. Mermer (2010), Türkçe gibi morfolojik olarak zengin dillerde sözcük altı birimlerin kullanımının sözcük temelli sistemlere kıyasla makine çeviri performansını arttırdığını söylemektedir. Mielke vd. (2021) tarafından da belirtildiği üzere, sözcük altı birimlendirme, düşük kaynaklı dillerde bile daha yüksek başarım sağlamış, açık kelime dağarcığını sabit sayıda birimden oluşan sözlükle temsil etme yetenekleri sayesinde DDİ alanında baskın yöntem haline gelmiştir.

Sözcük altı birimlendirme yöntemlerinin yukarıda anılan yaklaşımlardan farklılığını daha iyi açıklayabilmek için Hugging Face tarafından geliştirilen *tokenizers* kütüphanesiyle BPE ve WordPiece yöntemleri kullanarak farklı sözlük büyüklüklerinde hazırlanan birimlendiriciler aşağıda karşılaştırılmıştır.

BPE yaklaşımı için hazırlanan kod örneği Şekil 11’de verilmiştir.

Şekil 11

Hugging Face kütüphanesiyle hazırlanmış BPE birimlendirici eğitim kodu

```
import sys
from tokenizers import Tokenizer, models, trainers, pre_tokenizers,
normalizers
from tokenizers.normalizers import NFC, Lowercase
vocab_size = int(input("Vocabulary Size: \n"))

# Load and normalize the corpus
corpus = []
with open(sys.argv[1], encoding="utf-8") as f:
    for line in f:
        corpus.append(line.strip())

# Initialize BPE tokenizer
tokenizer = Tokenizer(models.BPE())
tokenizer.normalizer = normalizers.Sequence([NFC(), Lowercase()])
tokenizer.pre_tokenizer = pre_tokenizers.Whitespace()

# Train tokenizer
trainer = trainers.BpeTrainer(vocab_size=vocab_size, show_progress=True)
tokenizer.train_from_iterator(corpus, trainer)

# Save tokenizer to JSON
tokenizer.save(f"turkish_bpe_tokenizer_{vocab_size}.json")

# Export vocabulary
vocab = tokenizer.get_vocab()
sorted_vocab = sorted(vocab.items(), key=lambda x: x[1])  # sort by token
ID

with open(f"turkish_bpe_{vocab_size}.txt", "w", encoding="utf-8") as f:
    for token, idx in sorted_vocab:
        f.write(f"{token}\t{idx}\n")
```

Örnek kod eğitim için kullanılacak derlemi sistem argümanı olarak almakta, ardından kullanıcıdan hedeflenen sözlük büyüklüğünü istemektedir. Kod sağlanan derlem ve değer ile birimlendiriciyi üreterek çıktıyı sözlük büyüklüğü değerini dosya adında belirterek kaydetmektedir.

Aynı prensiple WordPiece yaklaşımı için hazırlanan kod Şekil 12'de verilmiştir.

Şekil 12

Hugging Face kütüphanesiyle hazırlanmış WordPiece birimlendirici eğitim kodu

```
import sys
from tokenizers import Tokenizer, models, trainers, pre_tokenizers,
normalizers
from tokenizers.normalizers import NFC, Lowercase

# Get vocabulary size from user
vocab_size = int(input("Vocabulary Size: \n"))

# Load and normalize the corpus
corpus = []
with open(sys.argv[1], encoding="utf-8") as f:
    for line in f:
        corpus.append(line.strip())

# Initialize WordPiece tokenizer
tokenizer = Tokenizer(models.WordPiece(unk_token="[UNK]"))
tokenizer.normalizer = normalizers.Sequence([NFC(), Lowercase()])
tokenizer.pre_tokenizer = pre_tokenizers.Whitespace()

# Train tokenizer
trainer = trainers.WordPieceTrainer(
    vocab_size=vocab_size,
    show_progress=True,
    special_tokens=["[UNK]", "[CLS]", "[SEP]", "[PAD]", "[MASK]"]
)
tokenizer.train_from_iterator(corpus, trainer)

# Save tokenizer to JSON
tokenizer.save(f"turkish_wordpiece_tokenizer_{vocab_size}.json")

# Export vocabulary
vocab = tokenizer.get_vocab()
sorted_vocab = sorted(vocab.items(), key=lambda x: x[1])
with open(f"turkish_wordpiece_{vocab_size}.txt", "w", encoding="utf-8")
as f:
    for token, idx in sorted_vocab:
        f.write(f"{token}\t{idx}\n")
```

Birimlendiricilere verilen örnek 1 “Defne kahvaltısını yaptı, sonra okula gitti.”, örnek 2 “sarsılıp durur şimşek çakınca camlar” cümleleridir.

Tablo 3

Farklı sözlük büyüklükleriyle hazırlanmış BPE tokenizer çıktıları

Sözlük Boyutu	Model	Örnek 1	Örnek 2
16000	BPE	defne kahvaltı sını yaptı , sonra okula gitti .	sars ılıp dur ur şimşek çak ınca

			cam lar
32000	BPE	defne kahvaltı sını yaptı , sonra okula gitti .	sars ılıp durur şimşek çak ınca cam lar
64000	BPE	defne kahvaltı sını yaptı , sonra okula gitti .	sars ılıp durur şimşek çak ınca camlar
128000	BPE	defne kahvaltısını yaptı , sonra okula gitti .	sars ılıp durur şimşek çak ınca camlar

Tablo 3 farklı sözlük büyükleri belirlenerek hazırlanan BPE tabanlı birimlendiricilerin örnek cümleler için ürettiği çıktıları göstermektedir. Daha küçük boyutlu olan 16k ve 32k modelleri gözlendiği üzere sözcükleri daha fazla parçalama eğilimi gösterirken, 64k ve 128k modelleri sözcük bütünlüğünü korumaya daha yakındır.

Tablo 4

Farklı sözlük büyüklükleriyle hazırlanmış WordPiece tokenizer çıktıları

Sözlük Boyutu	**Model**	**Örnek 1**	**Örnek 2**
16000	WordPiece	def ##ne kahvaltı ##sını yaptı , sonra okula gitti .	sars ##ıl ##ıp ##dur ##ur şimşek ##çak ##ınca cam ##lar
32000	WordPiece	def ##ne kahvaltı ##sını yaptı , sonra okula gitti .	sarsı ##lıp ##dur ##ur şimşek ##çak ##ınca cam ##lar
64000	WordPiece	defne kahvaltısı ##nı yaptı , sonra okula gitti .	sarsı ##lıp ##dur ##ur şimşek ##çak ##ınca camlar
128000	WordPiece	defne kahvaltısını yaptı , sonra okula gitti .	sarsı ##lıp ##duru ##r şimşek ##çak ##ınca camlar

WordPiece ile hazırlanan çıktıda ilk gözlenen “##” sembolünün kullanımıdır. Bu sembol, başına geldiği dizinin tek başına bir dizi olmadığını, yüksek olasılıkla bir başka birimin ardına geleceğini belirtir.

Tablo 4’de üretilen çıktılarda da küçük sözlük boyutlarında daha fazla parçalanma olduğu gözlenmektedir. Her iki tabloda dikkat edilmesi gereken önemli bir nokta da 128k modellerinde örnek 1’in doğrudan sözcüklerden oluşurken Örnek 2’nin bu sözlük boyutunda bile parçalanmış olmasıdır. Bu durum, Örnek 1’i oluşturan sözcüklerin dilde daha sık karşılaşılan, Örnek 2’yi oluşturan sözcükleri ise daha düşük sıklıkla karşılaşılan sözcüklerden oluşmuş olmasıdır. Aynı zamanda Örnek 2’yi oluşturan sözcükler Türkçenin biçimbirim özellikleri sebebiyle sözcük sonunda daha sık yer bulan ekler olmasıdır.

Bölüm 3

Yöntem

Çalışmanın temel hedefi Türkçe metinlerin işlenmesinde kullanılacak bir birimlendiricinin hazırlanmasıdır. Birimlendirici “TS Tokenizer” olarak adlandırılmıştır. Bu amaçla,

- birimlendiricinin hazırlanması için kullanılacak kurallar setinin oluşturulması ve
- bu kurallar setinin işletilmesinde izlenecek hiyerarşinin belirlenmesi gerekmektedir.

Bu amaca ulaşmak için çalışmada bilişimsel dilbilim ve veri madenciliği yöntemleri kullanılmıştır.

Birimlendirici algoritması sözcük listeleri ve düzenli ifadelerden yararlanarak işleyecek bir şekilde hibrit bir yaklaşımla tasarlanmıştır. Birimlendirici algoritmasında kullanılan kurallar listesi, iki boşluk karakterinin arasında kalan karakter dizilerine ilişkin özelliklerinin tanımlanmasına dayanmaktadır. Hedeflenen birimlendiricinin temel kaynağı olan verinin hacimsel büyüklüğü sebebiyle elle yapılması mümkün olmayan bu işlemler için en uygun yöntemler olarak bilişimsel dilbilim ve veri madenciliği yöntemleri belirlenmiştir.

Araştırma Yöntemi

Bu çalışmada kullanılan araştırma yöntemi, bilişimsel dilbilim ve veri madenciliği yaklaşımlarının bütünleştirilmesine dayanmaktadır. Bilişimsel dilbilim, bilgisayar teknolojileri kullanılarak dilin sistematik olarak incelenmesidir. Bilişimsel dilbilim, doğal dil veri kümelerini analiz ederek, veri içinde bulunan sayısallaştırılabilir (formalleştirilebilir) bilgiyi ortaya çıkarmayı ve bu çıkarımlar sonucunda kullanılabilir bilgi üretmeyi hedeflemektedir. Veri madenciliği ise özellikle büyük hacimli veri setlerinden işlenebilir, analiz edilebilir bilginin damıtılması olarak tanımlanabilir. Bu iki disiplinin kesişiminde yer

alan kümeleme, sınıflandırma ve örüntü tanıma teknikleri, çalışmanın kuramsal ve uygulamalı temelini oluşturmaktadır.

Witten (Witten ve diğerleri, 2017), veri madenciliğini anlamlı örüntüleri ve ilişkileri ortaya çıkarmak amacıyla bilgisayar algoritmalarının kullanılmasına dayalı bir süreç olarak tanımlar. Elde edilen örüntüler, verinin daha iyi anlaşılması ve işlenmesi sürecinde verilecek kararlara rehberlik etme fırsatı sunmaktadır (Fayyad ve diğerleri, 1996).

Tunalı (2011) veri madenciliğinin 1960'lardan itibaren teknolojideki gelişmeyle birlikte, hızla artan veri hacmini yönetme ihtiyacından doğduğunu söylemektedir. Dolgun ve arkadaşları (2009), veri madenciliğini veriden anlamlı bilgileri ve ilişkileri çıkarmada kullanılan tekniklerin bütünü olarak tanımlamaktadır. Weiss (Weiss ve diğerleri, 2010) veri madenciliğinin büyük hacimli verinin oluşması ve depolanmasına bir yanıt olarak ortaya çıktığını söylemektedir. Han ise daha geniş bir bakışla veri madenciliğinin "bilgi teknolojilerindeki doğal evrimin" bir sonucu olarak ortaya çıktığını söyleyerek, kavramın yalnızca veriden bilgi çıkarımı şeklinde tanımlanamayacağını, internet, veri tabanları, veri hangarı gibi verinin akış halinde olduğu sistemlerin de veri madenciliğinin ilgi alanına girdiğini dolayısıyla veri madenciliğinin büyük hacimli veri içindeki ilgi çekici örüntü ve bilginin keşfedilmesi olarak tanımlanabileceğini söylemektedir (Han ve diğerleri, 2012).

Bu kuramsal çerçeve doğrultusunda birimlendirici, aşağıda, "*Birimlendirici Tasarımı*" başlığı altında detaylarını ele aldığımız şekilde, sözcük listeleri ve düzenli ifadeler kullanarak çalışmak üzere hibrit bir yaklaşımla tasarlanmıştır. Bununla birlikte, özellikle karmaşık ve gürültülü dizilerde görülen örüntüleri tanımlamak için K-ortalamalar (K-means) algoritmasından da yararlanılmıştır. Wu (2012) tarafından tanımlandığı üzere, bu algoritma kullanıcı tarafından belirlenen K sayıda küme oluşturur ve verileri bu kümelere atar.

Bu çalışmada K-ortalamalar algoritması, gürültülü dizilerdeki "noktalama işaretlerinin sayısını", "hangi noktalama işaretleri olduklarını", "konumlarını", "dizinin uzunluğunu" gibi öznitelikler (feature) tanımlanarak kullanılmıştır. Bu sayede, düzenli

ifadelerin yapılandırılmasında kullanılmak üzere, gürültülü dizilerin örüntüsel özelliklerine göre sınıflandırılması sağlanmış ve birimlendiricinin kapsamı genişletilmiştir.

Veri Toplama Süreci

Çalışmada açık erişimli üç veri setinden yararlanılmıştır. Veri kaynakları, farklı yıllarda yayınlanmış ve içerik çeşitliliği sağlayacak biçimde seçilmiştir. Her üç veri seti de dilsel işaretleme içermeyen, ham metinlerden oluşmaktadır.

En büyük hacimli veri kümesi Common Crawl (CC) 2017[4] açık web taraması kaynaklı veri kümesidir. Bu veri seti Ocak 2017 tarihinde yayınlanmıştır. CC 2017 veri kümesi 4 milyar 250 milyon 365 bin 257 satır ve 21 milyar 882 milyon 861 bin 021 birimden oluşmaktadır. Bu veri seti internet kaynaklarından elde edilmiş doğal ve heterojen içerikleri barındırması nedeniyle, birimlendiricinin genellenebilirliğini test etmek için verimli bir kaynak sunmaktadır.

İkinci veri kümesi, Avrupa dil kaynaklarını bir araya getiren MaCoCu[5] (Massive Collection of Comparable Corpora) adlı çok dilli derlemin Türkçe bölümüdür. Bu veri seti Mayıs 2022'de yayınlanmıştır. Özellikle bilgisayar destekli çeviri sistemlerinin hazırlanması amacıyla oluşturulmuş ve Mayıs 2022 tarihinde yayınlanmış olan MaCoCu derlemidir. Bu veri seti 131 milyon 090 bin 305 satır ve 4 milyar 346 milyon 270 bin 326 birimden oluşmaktadır. Bu veri seti, görece daha temiz ve düzenli bir kaynak sunmaktadır.

Çalışmada kullanılan üçüncü veri kümesi ise Haziran 2019 tarihinde yayınlanmış olan TS TimeLine[6] derlemidir. Bu veri kümesi 2 milyon 586 bin 976 satır ve 719 milyon 192 bin 477 birimden oluşmaktadır. Bu veri seti Türkçe haber ve köşe yazılarından oluşmaktadır. Diğer iki veri setine kıyasla daha temiz bir veri kaynağıdır.

---

[4] https://commoncrawl.org/the-data/get-started/

[5] https://macocu.eu/

[6] https://tscorpus.com

Veri kümeleriyle ilgili satır ve birim sayıları verinin yayınlandığı kaynaklardan aktarılmaktadır. Çalışmada kullanılan toplam veri hacmi büyüklük sırasına göre aşağıdaki tabloda verilmektedir.

Tablo 5

Kullanılan veri setleri ve hacimleri

Derlem	Satır Sayısı	Sözcük Sayısı	Yayın Tarihi
CC 2017	4.250.365.257	21.882.861.021	2017
MaCoCu Corpus	131.090.305	4.346.270.326	2022
TS TimeLine	2.586.976	719.192.477	2019
Toplam	4.384.042.538	26.948.323.824	---

Bu üç veri seti toplamda 26.94 milyar sözcükten oluşan bir hacim oluşturmakta ve dilsel çeşitlilik olarak kapsayıcılık sağlamaktadır. Çalışmanın sayıltılarında da belirtildiği üzere bu veri setinin Türkçenin sözcük çeşitliliğini yansıttığı kabul edilmiştir.

Verinin Ön İşleme Süreci

Bu basamakta, çalışmaya konu olan veri setleri bir araya getirilerek tek bir ham metin dosyası olarak birleştirilmiştir. Ardından veri, tüm boşluk karakterleri satır sonu işareti ile değiştirilerek veri her satırda bir “*dizi*” olacak şekilde biçimlendirilmiştir. Bu işlem sonucunda ortaya çıkan her bir satır, bir karakter dizisi olarak ele alınmış ve çalışmada “dizi” olarak tanımlanmıştır. Dizi kavramı, ham veriyi oluşturan her satırın hem birimlendiriciden elde edilmesi beklenen çıktı hem de dilsel olarak geçerli bir birime karşılık gelmemesi sebebiyle farklılığı betimlemek amacıyla tercih edilmiştir. Örneğin “araba”, “;)”, “-gel”, “okul/kurumlarında”, “...Devam?...”, “[1.9.22_1171]Rating:”, “ptt+90”, “a@b.com”, “###__..”, “okul,öğrenci” gibi örnekler birer dizidir. Birim kavramı ise, bu dizilerin birimlendiriciden döndürüldüğü halini ifade etmektedir. “-gel” dizisinin birimlendirici

çıktısının “-” ve “gel” biçiminde, ““...Devam?...” dizisinin “...”, “Devam”, “?”, “...” biçiminde olması beklenmektedir. Bu nihai çıktıyı oluşturan parçalar “birim” olarak kabul edilmiştir.

Toplam veri hacminin ~27 milyar sözcük olması sebebiyle, işlem sürecinin hızlandırılması adına veri Türkçe alfabedeki “ğ” dışında kalan 28 harf ile başlayanlar ve bu 28 harf dışında kalan herhangi bir karakterle başlayanlar olmak üzere toplam 29 parçaya bölünmüştür. Her parça Linux kabuğu (bash) üstünde çalışan uniq[7] yazılımıyla benzersiz (tekil) dizilerden oluşacak biçime getirilmiştir. Bu süreçte büyük/küçük harf duyarlılığı korunmuştur.

Böylelikle,

- Aynı dizinin tekrar işlemlenmesinin önüne geçilmiş,
- Daha az donanım kaynağı kullanılarak birimlendirici tasarım ve sınama sürecinin yürütülmesi sağlanmıştır.

Ön işleme sonrasında elde edilen tekil diziler ve hacimleri aşağıdaki tabloda verilmektedir.

Tablo 6

Başlangıç karakterine göre tekil dizi sayıları

Harf	Dizi Sayısı
a	11.288.400
b	1.416.597
c	3.530.074
ç	2.993.209
d	9.338.133
e	6.689.562

[7] https://man7.org/linux/man-pages/man1/uniq.1.html

f	2.955.585
g	8.239.449
h	6.589.220
ı	5.329.177
i	1.072.572
j	571.550
k	12.889.291
l	1.942.352
m	7.189.689
n	2.411.941
o	5.432.569
ö	2.090.292
p	4.318.407
r	2.365.927
s	11.717.815
ş	1.577.978
t	7.504.504
u	2.143.651
ü	897.569
v	3.069.773
y	8.636.529
z	1.216.129

Diğer	41.784.288
Toplam	187.202.232

Tabloda diğer olarak verilen dizilerin toplam içinde %22.32'luk bir oran oluşturduğu gözlenmektedir. Bu diziler,

- x, w, q harfleriyle başlayan dizileri,
- noktalama işaretiyle başlayan dizileri,
- emoji ve smiley işaretlerini,
- farklı alfabelerde (Kiril, Arapça vb.) bulunan karakterler ile başlayan dizileri

içermektedir. Bu dizilerin tüm veri içinde ⅕ oranından fazla yer tutması, çalışmanın öneminde de vurgulandığı üzere, ham verinin verimli şekilde birimlendirilmesinin değerini ortaya koymaktadır.

Verilerin Analizi

Veriler **(i)** Linux kabuğu (bash) üzerinde çalışan araçlar ve **(ii)** Python (3.11.2) programlama dili kullanılarak hazırlanmış betikler ile işlenmiştir. Linux kabuğu sed, awk, sort, uniq vb. araçların doğrudan çalıştırılabileceği bir ortam sunmaktadır. Python, DDİ, makine öğrenimi ve yapay zekâ uygulamaları başta olmak üzere birçok çalışmada tercih edilen, yüksek seviyeli ve esnek bir dildir. Python, yorumlayıcı (interpreter) tabanlı bir yüksek seviye programlama dili olup, derlenmeye (compile) gereksinim duymadan çalıştırılabilmektedir.

Analiz süreci, birimlendiricinin tasarımı ve test edilmesi için gerekli olan kuralların belirlenmesi, özniteliklerin çıkarılması, gürültülü dizilerin filtrelenmesi ve veri kümelerinin ayrıştırılması adımlarını kapsamaktadır.

Ayrıca bir Python kütüphanesi olan NLTK (Natural Language Toolkit, Doğal Dil İşleme Araç Seti) altında tanımlı olan dört ayrı birimlendirici ile ham veri, bir karşılaştırma (benchmark) yapılabilmesi amacıyla birimlendirilmiştir. Bunun için hazırlanan betik (nltk_tokenizers.py) ve elde edilen skorlar birimlendiricinin sunulduğu kod deposunda paylaşılmıştır[8]. Bu betik, NLTK kütüphanesinin *word_tokenize*, *wordpunct_tokenize*, *WhitespaceTokenizer* ve *TreebankWordTokenizer* metotlarını kullanarak verilen bir metin dosyasını birimlendirmeyi sağlamaktadır. Betik, biçimlendirilmesi istenen metni bir sistem argümanı olarak alırken --tok parametresiyle wt, wpt, ws, tb seçeneklerini betiğe verilmektedir.

word_tokenize (wt) noktalama işaretlerini diziden ayıran, ondalık sayıları tanıyabilen, virgül ve nokta gibi noktalama işaretlerini bağımsız birer birim olarak kabul eden bir birimlendiricidir. İngilizce temelli olarak hazırlanmış olup Türkçe için başarımı düşüktür.

wordpunct_tokenize (wpt) her türlü noktalama işaretini diziden ayıran basit bir yaklaşım sunmaktadır. Dilden bağımsız olmakla birlikte dizileri daha fazla parçalama eğilimindedir.

WhitespaceTokenizer (ws) sadece boşluk karakterine göre birimlendirme yapan basit prensipli bir birimlendiricidir. Doğal dilde verimli değildir.

TreebankWordTokenizer (tb) Penn Ağaç Yapılı Derlemine dayalı olarak geliştirilmiş bir birimlendiricidir. İngilizce odaklı olarak hazırlanmış, sözcük listesi temelli bir birimlendiricidir.

[8] https://github.com/tanerim/ts_tokenizer/tree/main/nltk_benchmark

Tablo 7

Python NLTK birimlendiricileri

Tokenizer	Açıklama	Dil Bağımlılığı	Türkçe Başarımı
word_tokenize (wt)	Noktalama duyarlı	İngilizce	Düşük
wordpunct_tokenize (wpt)	Noktalama duyarlı	Bağımsız	Orta
WhitespaceTokenizer (ws)	Boşluk tabanlı birimlendirme	Bağımsız	Çok Düşük
TreebankWordTokenizer (tb)	Sözcük listesi temelli	İngilizce	Düşük

Tablo 7'de gösterildiği üzere NLTK içindeki birimlendiriciler İngilizce için veya basit prensiplere göre hazırlanmıştır. Ancak kolay kullanılabilir ve hızlı olmaları sebebiyle Türkçe DDİ çalışmalarında da sıklıkla kullanılmaktadır (Aksoy, 2022; Baykara & Güngör, 2022; Görgün ve diğerleri, 2023; Özdemir & Yeniterzi, 2020; Yaşar ve diğerleri, 2019; Yazar ve diğerleri, 2024).

Şekil 13

Python NLTK kütüphanesi birimlendiricileri örnek kodları

```
      # ------------------------------------------|
      # Use Python NLTK Tokenizers to create a    |
      # Benchmark for comparison - ts -           |
      # ------------------------------------------|
      # w_tokenized_tokens = word_tokenize(text)  |
      # w_punct_tokens = wordpunct_tokenize(text) |
      # ------------------------------------------|
      # wht_spc_tokenizer = WhitespaceTokenizer() |
      # tokenizer = TreebankWordTokenizer()       |
      # ------------------------------------------|
      import argparse
      import multiprocessing
      from tqdm import tqdm
      from nltk.tokenize import word_tokenize, wordpunct_tokenize,
WhitespaceTokenizer, TreebankWordTokenizer
      parser = argparse.ArgumentParser(description='Tokenization with
different built-in methods of NLTK.')
      parser.add_argument('filename', type=str, help='Input file')
      parser.add_argument('--tok', type=str, choices=['wt', 'wpt', 'ws',
'tb'], default='ws',
      help='Tokenization methods are as follows'
      '(wt: word_tokenize,'
```

```
      'pt: wordpunct_tokenize,'
      'ws: WhitespaceTokenizer,'
      'tb: TreebankWordTokenizer)')
      args = parser.parse_args()
      with open(args.filename, encoding='utf-8') as f:
           in_file = f.read().split("\n")
      if args.tok == 'wt':
           tokenizer = word_tokenize
      elif args.tok == 'wpt':
           tokenizer = wordpunct_tokenize
      elif args.tok == 'ws':
           tokenizer = WhitespaceTokenizer().tokenize
      elif args.tok == 'tb':
           tokenizer = TreebankWordTokenizer().tokenize
      # Function to process each line using the chosen tokenizer
      def process_line(line):
           tokens = tokenizer(line)
           return tokens
      # Main processing using multiprocessing
      def main():
           try:
                with open(args.filename, encoding='utf-8') as file:
                     infile = file.read().split("\n")
                num_workers = multiprocessing.cpu_count() - 1
                with multiprocessing.Pool(processes=num_workers) as pool:
                     results = pool.imap(process_line, infile,
chunksize=100)
                          for tokens in tqdm(results, total=len(infile),
desc="Processing lines", unit="lines"):
                     for token in tokens:
                          print(token)
           except FileNotFoundError:
                print(f"Error: File {args.filename} not found.")
           except Exception as e:
                print(f"An error occurred: {e}")
      if __name__ == '__main__':
           main()
```

Betik Bash üstünde,

$ python nltk_tokenizers.py [girdi_ham_metin_dosyası] --tok [birimlendirici]

komutuyla çalıştırılmaktadır. Ayrıca, çok çekirdekli işlemcilerden yararlanmak üzere *multiprocessing* kütüphanesi betiğe entegre edilerek büyük veri dosyalarının paralel işlenmesi sağlanmış, işlem süresi anlamlı biçimde azaltılmıştır.

Veri analizinin son adımı, birimlendiricinin geliştirilmesine paralel olarak yürütülmüştür. Bu süreç boyunca birimlendiricinin kullandığı sözcük listelerinin güncellenmesi, yeni sözcük listelerinin eklenmesi, yeni düzenli ifadelerin eklenmesi, düzenli ifadelerin güncellenmesi veya işlem akışının değiştirilmesi sonrasında Tablo 6'da Diğer olarak tanımlanmış diziler birimlendiriciye verilmiştir. Tüm veri seti içinde en kirli veriyi içeren bu listeden OOV (Out-of-Vocabulary) olarak dönen diziler toplanarak K-Means yöntemiyle analiz edilmiştir. Böylelikle bu dizilerin karakter temelli örüntüler açısından kümelenmesi sağlanmıştır. K-Means, veri noktalarını benzerliklerine göre gruplara ayıran ve her bir grubu temsil eden bir merkez etrafında kümeler oluşturan bir yöntemdir (Syakur ve diğerleri, 2018). Bu çalışmada, her bir dizinin;

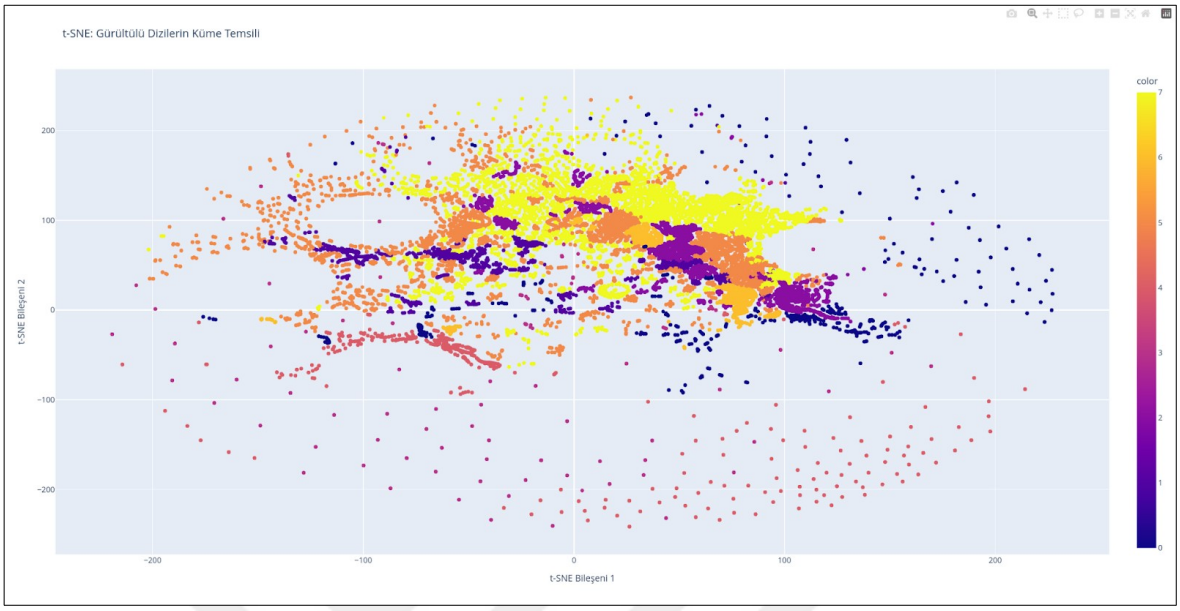
- uzunluğu,
- içerdiği noktalama işareti sayısı,
- noktalama işaretlerinin dizideki konumu,
- büyük harf içerip içermediği,
- rakam barındırıp barındırmadığı

gibi biçimsel özellikleri çıkarılmıştır.

Elde edilen kümelerin dağılımını görsel olarak sunmak amacıyla t-dağılımlı stokastik komşuluk yerleştirmesi (t-SNE, t-Distributed Stochastic Neighbor Embedding) yöntemiyle iki boyutlu temsil oluşturulmuştur. Aşağıda yer alan görselde, her bir küme farklı renklerle temsil edilmekte; biçimsel örüntüleri benzer olan OOV dizilerinin kümeler etrafında yoğunlaştığı Şekil 14'de açık biçimde gözlemlenebilmektedir.

Şekil 14

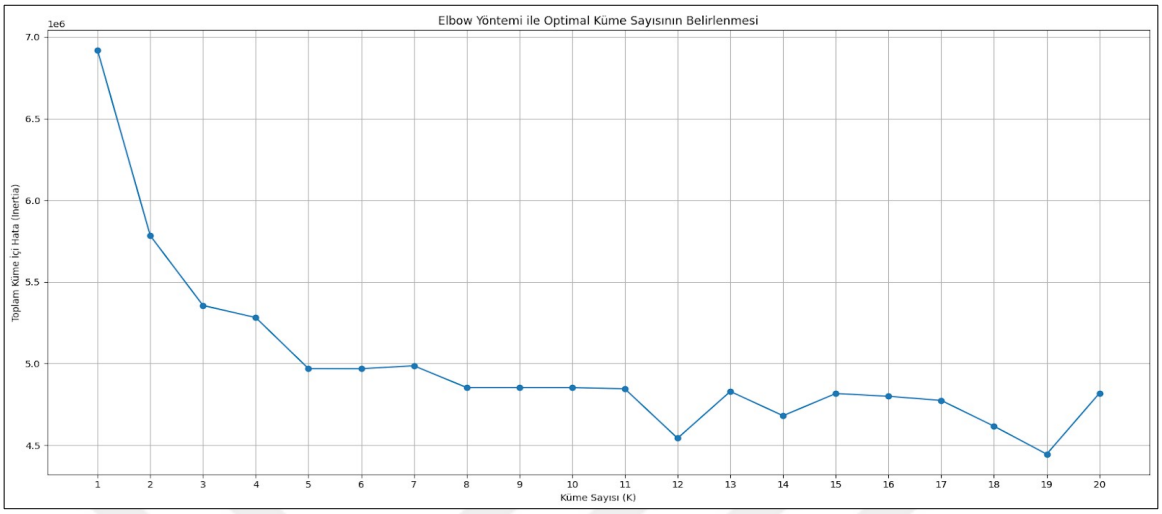
t-SNE Küme Görselleştirmesi



Kümeleme sürecinde karşılaşılan temel sorunlardan biri, kaç adet küme oluşturulacağının önceden bilinmemesidir. Bu nedenle, Elbow yöntemi kullanılarak optimal K değerine erişilmiştir. Elbow yöntemi, farklı K değerleri için hesaplanan hata oranlarını görselleştirerek, eğrinin kırılma (dirsek) gösterdiği noktayı en uygun K değeri olarak kabul eder (Ketchen & Shook, 1996). Bu yaklaşım, örüntüsel benzerliklerin yoğunlaştığı ve fazla bölünmenin faydalı bilgi sağlayabilmekten uzaklaştığı noktayı belirlemeye yardımcı olur. Böylelikle, kümeler arası farkların belirgin olduğu noktaya kadar olan kümeler dikkate alınmış, anlamlı bilgi sunmadığı kabul edilen kırılımlar dikkate alınmamıştır.

Şekil 15

Elbow kırılımı



Bu süreç sonucunda elde edilen kümeler, biçimsel örüntüleri benzer OOV dizileri bir araya getirerek, hem yeni düzenli ifadelerin tanımlanmasında rehberlik etmiş hem de birimlendiricinin başarımını yükseltecek yeni kuralların oluşturulmasına katkı sunmuştur.

Bölüm 4

Birimlendirici Tasarımı

Bu bölümde TS Tokenizer birimlendiricisinin tasarım ilkeleri ve kullanım özellikleri ayrıntılı biçimde sunulmaktadır. Birimlendirici, Türkçe metinlerin doğru ve işlevsel biçimde birimlendirilmesini sağlamak amacıyla geliştirilmiştir. Bu bağlamda çok katmanlı bir işleyiş modeli benimsenmiştir. TS Tokenizer mimarisi, ardışık işlemlerden oluşan ve her bir işlem adımında belirli bir yapısal niteliğe odaklanan bir algoritma kullanmaktadır.

Algoritma verilen metindeki her satırı boşluk karakterinden bölerek ilerler. İki boşluk karakteri arasında kalan her dizi bir birim adayı olarak kabul edilir. Her birim adayı, TokenProcessor sınıfı içerisinde tanımlı olan process_token() metodu aracılığıyla çok katmanlı bir denetim sürecinden geçmektedir. Bu denetim süreci, birimlendiricinin temel algoritmik mantığını oluşturan dört aşamalı bir sıralı yapı üzerine kurulmuştur:

1- sözlük tabanlı kontroller,

2- düzenli ifade (regex) tabanlı kontroller,

3- çoklu noktalama yapıları içeren diziler

4- tekli noktalama yapıları içeren diziler.

Her bir kontrol için tanımlanmış bir fonksiyon vardır ve bu fonksiyonlar TokenPreProcess sınıfı içinde tanımlanmıştır.

Sözcük Listeleri

Yukarıda anıldığı şekilde (bkz. *Sözlük Tabanlı Birimlendirme*), bir sözlükten yararlanmak da sıklıkla kullanılan bir birimlendirme yaklaşımıdır. Birimlendirici öncelikle çeşitli listelerden yararlanarak girdi dizinin bu listelerde bulunup bulunmadığını kontrol etmektedir. Bu listelerden birinde yer alan ve tam eşleşme sağlayan diziye ilgili bir etiket atanarak birimlendirme süreci tamamlanmaktadır. Bu işlem basamağında aşağıdaki tabloda verilen 7 farklı sözcük listesi kullanılmıştır.

Tablo 8

Birimlendiricinin faydalandığı sözcük listeleri

Sözcük Listesi	İçerik	Sözcük Sayısı
TS Corpus Sözcük Listesi	Bu sözcük listesi, çekimli yapılar da dahil olmak üzere TS Corpus altında bulunan derlemler, farklı formatlardan metin haline getirilmiş elektronik kitaplar ve çeşitli internet kaynaklarından oluşmaktadır.	3.280.440
İngilizce Sözlükler Listesi	Bu liste İngilizce sözcüklerden oluşmaktadır.	272698
Kısaltmalar Listesi	Türkçe kısaltmalar listesi.	1023
Emoji LİstesiBu liste emoji işaretlerinden oluşmaktadır.		926
Smiley Listesi	Bu liste Smiley işaretlerini barındırmaktadır.	202
İstisnalar Listesi	e-posta, e-kitap gibi, veri setinde sıklıkla gözlenmiş ve internet üstünde sıklıkla kullanılan sözcükler listesi	131
Alan Adı Listesi	İnternet adreslerinde en fazla kullanılan alan adları listesi	28

Kontrol *lexicon_based* listesinde bulunan fonksiyonlar ile ve bu listede tanımlı sırayla yapılmaktadır. Liste dizinin sırasıyla şu listelerde bulunup bulunmadığını kontrol eder:

1. İstisnalar listesi,
2. emojiler listesi,
3. smiley listesi,
4. kısaltmalar listesi,
5. geçerli sözcükler listesi,
6. İngilizce sözcükler listesi.

Son olarak dizi eğer tek bir noktalama işaretinden oluşuyorsa yine bu seviyede işlenir. Böylelikle tek bir noktalama işaretinden oluşan ve her iki yanında bir boşluk karakteri barındıran dizinin tanımlanma süreci tamamlanmış olacaktır.

Geçerli sözcüklerin kontrolünü sağlayan liste 3 milyon 280 bin 440 sözcükten oluşmaktadır. Bu listede yalnızca kök durumundaki sözcükler değil, aynı zamanda sözcüklerin çekimli yapıları da yer almaktadır. Bu bakımdan liste, kapsayıcılığı ve hacmi bakımından dikkat çekicidir. Aynı zamanda listenin gürültüden arındırılmış olması çalışmada kullanılması için tercih sebebi olmuştur.

Birimlendirici eğer girdi dizinin bu listelerden birinde yer alan bir diziyle *tam eşleşmesini* yakalarsa bu diziyi doğrudan bir birim olarak kabul etmektedir. Kullanılan listeler arasına bir istisnalar listesinin eklenmesi, farklı çalışmalarda bu birimlendiriciyi kullanmak isteyenler için bir genişleme ve esneklik alanı oluşturulması amacıyla tercih edilmiştir. Bu listeye eklenen her dizi, birimlendiricide başka bir işleme tabi tutulmayacaktır.

Kısaltmalar listesi TrMorph (Çöltekin, 2010) tarafından kullanılan kısaltmalar listesini temel almaktadır[9]. Smiley listesi Wikipedia'da bulunan smiley listesi girdisinden[10], emoji listesi ise Unicode konsorsiyumunun ilgili güncel listesinden[11] alınmıştır.

Eşleşme durumunda dizi doğrudan bir birim olarak kabul edilir ve döndürülür. Geçerli sözcükler "*Valid_Word*", İngilizce sözcükler "*English_Word*, kısaltmalar "*Abbr*", emojiler "*Emoticon*", smiley ifadeleri "*Smiley*", istisna sözcükler "*Exception*" etiketiyle işaretlenir.

Hem geçerli sözcük listesinde hem İngilizce sözcükler listesinde bulunan tüm ortak diziler birimlendiricinin hedef dilinin Türkçe olması sebebiyle İngilizce sözcükler listesinden silinmiştir. Örneğin "but" dizisi "Valid_Word" olarak döndürülmektedir.

---

[9] https://github.com/coltekin/TRmorph/blob/master/lexicon/abbreviation

[10] https://en.wikipedia.org/wiki/List_of_emoticons

[11] https://unicode.org/emoji/charts/full-emoji-list.html

Düzenli İfadeler İle Tanımlama

İkinci kontrol adımı düzenli ifadelerden yararlanarak yapılmaktadır. Düzenli ifadeler bir karakter dizisinin belirli bir biçimini veya örüntüsünü tanımlamak üzere kullanılan bir ifade biçimidir (Goyvaerts, 2006). Bir metin içinde yer alan ve bir kural çerçevesinde tanımlanabilen dizileri belirlemek amacıyla veri madenciliği, DDİ vb. alanlarda sıklıkla kullanılırlar. Friedl (2006), düzenli ifadelerin kendine özgü üst karakterler (*, +, ?, vb.) ve gerçek karakterlerden (a, 1, b, vb.) oluştuğunu söyler. Üst karakterler özel işlevleri yerine getiren karakterlerdir. Düzenli ifadelerde "*" sembolü herhangi bir sayıda herhangi bir karakter, "^" sembolü satırın başlangıç noktası anlamı taşır. Örneğin "^a.*$" düzenli ifadesi sırasıyla,

- **^** satırın başlangıcından itibaren başlanırsa ifadenin geçerli olacağını,
- **a** Küçük harf a karakterinin satırın ilk karakteri olması gerektiğini,
- **.*** Küçük harf "a" karakterinden sonra, "." satır sonu işareti hariç, sıfır veya daha fazla karakterin gelebileceğini,
- **$** ise satırın sonunu belirtir.

Bu ifade ile "a123", "algoritma" "a bendinde belirlenmiş" gibi diziler yakalanır.

Birimlendirici, aday dizilerin tanımlanması için tanımlanmış 30 ayrı düzenli ifadeden yararlanmaktadır. Birimlendiricinin tasarımı sırasında düzenli ifadelerin işleme alınma sırasının başarım üstünde doğrudan etkili olduğu gözlenmiştir. Bu sebeple birimlendiricinin hazırlanmasında şu prensipler benimsenmiştir.

- **Özelden Genele:** Daha özel olan yapılarla eşleşen düzenli ifadeler öncelikle ele alınmıştır. Bu yöntem, belirli örüntülerin daha geniş ve genel örüntülerle karışmasını önler ve düzenli ifadelerden daha verimli faydalanılmasını sağlar. Örneğin hashtag veya mention için kullanılan düzenli ifadeler işlem sırasına daha önce alınmaktadır. Böylelikle bu ifadelerin daha keskin olan örüntüsü verimli şekilde yakalanmaktadır.

- **Karmaşıklık ve Örtüşme:** Daha karmaşık olan ve diğer düzenli ifadelerle örtüşme ihtimali olan desenler, çakışmaların önlenmesi için özel desenlerin ardından gelmelidir. Bu, karmaşık yapıların doğru bir şekilde tanımlanmasını ve işlenmesini sağlar.

Kullanılan düzenli ifadelerin process_token() metoduna aktarım sırasıyla aşağıda verilmiştir.

Tam URL (Web Adresi) Dizileri: Bu düzenli ifade, http:// veya https:// ile başlayan ve ardından gelen alan adı, ya da bileşenlerini içeren tam web adreslerini tanımlamak üzere tasarlanmıştır.

- "full_url":re.compile(r'^((http|https)\:\/\/)[a-zA-ZıiİüÜçÇöÖşŞğĞ0-9_\ uFE0F\.\/\?\:@\-=#]+\.([a-zA-ZıiİüÜçÇöÖşŞğĞ0-9_\uFE0F\&\/\?\:@\-=#])+')

Bu tür yapılar birimlendirici tarafından tekil ve bölünmeden korunması gereken yapılar olarak değerlendirilmiştir. Hem “https://ornekwebsitesi.com” hem de “https://ornekwebsitesi.com/page_1” gibi diziler yakalanabilmektedir.

www ile Başlayan Web Adresleri: Bu düzenli ifade, protokol belirtmeden doğrudan www. ile başlayan web adreslerini tanımlamak üzere geliştirilmiştir. Bu tür yapılar, kullanıcı tarafından kopyalanan veya bağlam içinde kısaltılmış bağlantılar şeklinde metinlerde sıkça yer alır. Tam URL adreslerinden farklı olarak alınmaları, başarımı arttırmak için iki ayrı düzenli ifadeden yararlanılabilmesini sağlamıştır.

- "web_url":re.compile(r'^((www)\.)[a-zA-ZıiİüÜçÇöÖşŞğĞ0-9_\uFE0F\.\/\?\:@\-=#] +\.([a-zA-ZıiİüÜçÇöÖşŞğĞ0-9_\uFE0F\&\/\?\:@\-=#])+')

Bu ifade ile hem “www.ornekwebsitesi.com” hem de “www.ornekwebsitesi.com/urun?id=25” dizileri yakalanabilmektedir.

E-posta: Birimlendirici e-postalarla ilgili olarak iki ayrı düzenli ifade kullanmaktadır. İlk düzenli ifade “email_punc” olarak adlandırılmıştır. Bu ifade bir veya birden fazla noktalama işareti ile biten ve e-posta yapısına uygun olan dizileri yakalamaktadır. “email” düzenli ifadesi ise dizinin bir bütün olarak bir e-postadan oluştuğu dizileri yakalamaktadır.

email düzenli ifadesi yakaladığı diziyi doğrudan döndürmektedir. email_punc ifadesi ise e-posta kısmını döndürmekte, diziyi takip eden noktalama işaretini/işaretlerini işlenmek üzere döngüye almaktadır.

- "email_punc": re.compile(r'\b[' + re.escape(string.punctuation) + r']*[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+[' + re.escape(string.punctuation) + r']*\b')

- "email": re.compile(rf'^[^{puncs}][a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+(?<![{puncs}])$')

Para Birimi Sembolü İçeren Diziler: Bu düzenli ifade, önünde veya arkasında bir para birimi sembolü içeren dizileri yakalamaktadır. Bu tür yapılar hem metinsel hem de sayısal verilerde sıkça yer almakta ve özellikle ekonomik, finansal ve ticari bağlamlarda önem taşımaktadır.

- "currency":re.compile(rf"^(?:[{re.escape(''.join(LocalData.currency_symbols()))}]\d{{1,3}}(?:[.,]\d{{3}})*([.,]\d+)?"r"|\d{{1,3}}(?:[.,]\d{{3}})*([.,]\d+)?[{re.escape(''.join(LocalData.currency_symbols()))}])$")

İfadede yer alan LocalData.currency_symbols() fonksiyonunda 20 farklı para birimi sembolü tanımlıdır. “₺500”, “250₺”, “1.000,00€”, “$19.99”, “99.5£” gibi rakam ve para biriminden oluşan diziler yakalanmaktadır. Bu diziler “Currency” etiketiyle döndürülür.

Tarih Aralığı: Bu düzenli ifade, yalnızca Türkçede kabul edilmiş olan gün/ay/yıl yapısına uyan ve tire ile bağlamış tarih aralıklarını ifade eden dizileri yakalar.

- "date_range": re.compile(r'^(?:(?:0[1-9]|[1-2][0-9]|3[0-1])\.(?:0[1-9]|1[0-2])\.\d{4})-(?:(?:0[1-9]|[1-2][0-9]|3[0-1])\.(?:0[1-9]|1[0-2])\.\d{4})$')

Örneğin “01.01.2022-31.12.2022” veya “15.03.2020-20.03.2020” gibi diziler, zaman aralığı belirten tekil bir anlam birimi taşıdığından doğrudan birim olarak işlenir.

Yıl Aralığı: Bu düzenli ifade dört basamaklı, sıfırla başlamayan ve tire ile bağlanmış iki yıl ifadesini tanımlar.

- "year_range": re.compile(r'^(?:[1-9]\d{3})-(?:[1-9]\d{3})$')

Yıl aralıkları anlamsal bütünlüğü korumak adına birim olarak kabul edilmiştir.

Tarih için hem gün/ay/yıl hem de ay/gün/yıl biçiminde verilmiş dizileri “-”, “.” ve “/” karakterleriyle yazılmış olarak yakalayan ayrı bir fonksiyon tanımlanmıştır (bkz. *Birimlendirici Fonksiyonları*).

Saat: Saatlerle ilgili üç düzenli ifade kullanılmıştır. İlk düzenli ifade “hour_suffix” olarak adlandırılmıştır. Bu ifade saat bildiren bir ifade ve takip eden bir ekten oluşmaktadır. İkinci düzenli ifade olan “hour” ise saat bildiren bir diziyi yakalamaktadır. Her iki ifadenin yakaladığı diziler doğrudan döndürülmektedir. Bu düzenli ifade iki nokta (:) veya nokta (.) ile ayrılmış, saat kısmı 00 ile 23, dakika kısmı 00 ile 59 arasında olabilen dizileri yakalar. Örneğin 24:00 bu ifade tarafından göz ardı edilirken 19:45 Hour etiketiyle döndürülecektir. Bu iki ifade 24 saatlik zaman biçiminde yazılmış dizileri hedeflemektedir.

- "hour_suffix": re.compile(r"^(0[0-9]|1[0-9]|2[0-3])[:.][0-5][0-9](?:'te|'de|'da|'den|'dan|'ten|'tan|'deki|'daki)$")
- "hour": re.compile(r"^(0[0-9]|1[0-9]|2[0-3])[:.][0-5][0-9]$")

Kullanılan diğer düzenli ifade ise 12 saatlik zaman formatında yazılmış saatleri hedefler. Bu dizilerin AM veya PM ile bitmesi beklenir. Mevcut birimlendirici sürümünde bu ifade kapsam dışı tutulmaktadır.

"hour_12": re.compile(r"^(0[0-9]|1[0-9])[:.][0-5][0-9]([AP]M)$")

Hashtag: Hashtag sosyal medyada bir konu veya olaya ilişkin etiketleme yapmak için kullanılan # işaretiyle başlayan, dizilerdir. 2007 yılında Chris Messian tarafından önerilmiş ancak başlangıçta fazla ilgi görmemiştir (Piatek, 2021). Aynı yılın sonbaharında San Diego’da başlayan orman yangınlarıyla ilgili olarak Nate Ritter tarafından hashtag’ler kullanılarak atılan tweetlerin olayların takibini kolaylaştırdığı görülerek kullanımı yaygınlaşmıştır (LaRocca, & Boccia Artieri, 2022). Hashtagler için aşağıdaki düzenli ifade kullanılmıştır. Bu ifade yakaladığı diziyi doğrudan döndürmektedir.

- "hashtag": re.compile(r'^#[a-zA-ZıiİüÜçÇöÖşŞğĞ0-9_\uFE0F]{1,139}$')

Mention: Mention, Twitter, Facebook, WeChat gibi sosyal medya platformlarında @ işaretini takiben verilen kullanıcının işaret edilmesini sağlayan bir kullanımdır (Zhang, ve diğerleri, 2017). Mentionlar için aşağıdaki düzenli ifade kullanılmıştır. Bu ifade yakaladığı diziyi doğrudan döndürmektedir.

- "mention": re.compile(r'^@[a-zA-ZıiİüÜçÇöÖşŞğĞ0-9_\uFE0F]{1,15}$')

Tırnak İçi Diziler: Bu düzenli ifade, bir dizinin tanımlanmış tırnak işaretleri içinde yazılmış olup olmadığını denetler. Tırnak işaretleri önceden tanımlanmış bir fonksiyon ile düzenlenmektedir (bkz. Birimlendirici Fonksiyonları).

- "in_quotes": re.compile(r'^[\'"][^\'"]*[\'"]$')

Bu ifade tırnak işaretlerini “Punc” etiketiyle döndürürken iç dizi tekrar işlenerek döndürülür.

Kesme İşareti (’) İçeren Diziler: Türkçede kesme işareti özel adlara gelen ekleri ayırmak amacıyla kullanılır. Örneğin “Ankara’dan” veya “Defne’nin” dizileri birer bütün olarak anlam taşımaları sebebiyle bu ifade birimlendiriciye eklenmiştir.

- "apostrophed":re.compile(r"^([a-zA-ZıiİüÜçÇöÖşŞğĞ]+)'([a-zA-ZıiİüÜçÇöÖşŞğĞ]+)$")

Bu ifade tarafından yakalanan diziler “Apostrophed” etiketiyle döndürülür.

Parantez İçi Diziler: Bu düzenli ifade, bir dizinin tanımlanmış parantez işaretleri içinde yazılmış olup olmadığını denetler. Bu ifade, açılışta bir veya daha fazla parantez karakteri ((, [, {) ile başlayan, içinde başka bir parantez işareti barındırmayan ve kapanışta yine bir veya daha fazla kapama parantezi (),], }) ile biten dizileri hedef alır. Bu tür diziler genellikle açıklayıcı bilgiler, kaynak gösterimleri, numaralandırmalar veya ek bilgiler için kullanılır. Yazı dilinde özellikle parantez içi açıklamalarda ya da dipnotlarda yaygın olarak görülür.

- "in_parenthesis": re.compile(r'^[(\[{]+[^()\[\]{}]*[)\]}]+}$')

Örneğin, (örnek), [madde_1], {bkz} gibi diziler bu ifade tarafından yakalanır ve işlenmek üzere sonraki adıma taşınır. İşlem sonrasında öncül ve ardıl parantezler “Punc” etiketiyle işaretlenirken iç dizi tekrar işlenir.

Roma Rakamları: Bu düzenli ifade, Roma rakamı biçiminde yazılmış dizileri yakalamak amacıyla tasarlanmıştır. Roma rakamları genellikle kitap bölümleri, madde numaraları, kronolojik listeler ya da görsel düzenlemelerde kullanılır.

- "roman_number":re.compile(r'^(M{0,4}(CM|CD|D?C{0,3})(XC|XL|L?X{0,3})(IX|IV|V?I{0,3}))\.?$')

Dizinin sonunda “.” bulunması opsiyonel olarak tanımlanmıştır. Bu ifade ile “I”, “IV”, “XII.”, “MMX”, “XCIX” gibi diziler yakalanır. Bu dizilerin sayısal karşılıklarının ötesinde, yazılı metinlerde kategorik veya sıralayıcı bir işlev görmesi sebebiyle bütüncül olarak birimlendiricinin kapsamına alınması uygun bulunmuştur.

Tescilli Marka (®) Simgesi İçeren Diziler: Bu düzenli ifade, bir dizinin başında veya sonunda tescilli marka işareti olan ® karakterinin bulunup bulunmadığını denetler. “®” simgesi tescilli ticari markaları göstermek amacıyla kullanılır ve internet sitelerinde sıklıkla yer almaktadır.

- "registered":re.compile(r'(^®[a-zA-ZıiİüÜçÇöÖşŞğĞ0-9]+$)|(^[a-zA-ZıiİüÜçÇöÖşŞğĞ0-9]+®$)')

“®” işaretinin hem dizinin sonunda hem dizinin başında olduğu durumlar yakalanır.

Telif Hakkı İşareti İçeren Diziler: Bu düzenli ifade, telif hakkı simgesi olan “©” karakterinin dizinin başında veya sonunda yer aldığı yapıları tanımlamak amacıyla kullanılır. Telif hakkı simgesi özellikle web sayfalarında sıkça gözlenen bir karakterdir.

- "copyright":re.compile(r'(^©[a-zA-ZıiİüÜçÇöÖşŞğĞ0-9]+$)|(^[a-zA-ZıiİüÜçÇöÖşŞğĞ0-9]+©$)')

Bu ifade “©” işaretinin hem dizinin başında hem dizinin sonunda yer aldığı iki varyasyonu yakalar.

Ticari Marka (™) Simgesi İçeren Diziler: Bu düzenli ifade, bir dizinin başında veya sonunda ticari marka sembolü olan “™” karakterinin bulunup bulunmadığını denetler. ™ simgesi, tescil zorunluluğu olmadan kullanılan ticari markaları belirtmek için kullanılır ve özellikle ürün adlandırmaları, logolar veya marka temsilleri içeren metinlerde yer alır.

- "trade_mark":re.compile(r'(^™[a-zA-ZıiİüÜçÇöÖşŞğĞ0-9]+$)|(^[a-zA-ZıiİüÜçÇöÖşŞğĞ0-9]+™$)')

Bu tür yapılar, markasal referanslar açısından özel bir statüye sahiptir. Anlam bakımından bütüncül oldukları için, birimlendirici tarafından doğrudan tek bir birim olarak değerlendirilir.

Madde İmleri (•) ile Başlayan Liste Öğeleri: Bu düzenli ifade, Unicode madde imi karakteri olan • (bullet) ile başlayan ve ardından harf veya rakamlardan oluşan dizilerin geldiği yapıları yakalar.

- "bullet_list": re.compile(r'^•[a-zA-ZıiİüÜçÇöÖşŞğĞ0-9]+$')

Bu diziler bir liste ögesini tanımlamak amacıyla kullanıldıkları için birimlendirici tarafından tek bir anlamlı birim olarak işlenmektedir.

Yüzde Değerleri: Yüzde değerlerini yakalamak için farklı varyasyonlarda üç ayrı düzenli ifade yazılmıştır. İfadeler yüzde işareti ve sayı kombinasyonlarının başta, sonda, veya metin içinde başka karakterlerle birlikte kullanıldığı durumları tespit eder.

- “percentage_numbers_initial": re.compile(r'^%\d{1,3}(?:[.,]\d+)?$')
- "percentage_numbers_final": re.compile(r'^\d{1,3}(?:[.,]\d+)*%$')
- "percentage_numbers_chars": re.compile(r'^%\d{1,3}(?:[.,]\d+)*\D.*$')

İlk düzenli ifade yüzde işaretinin dizinin başında yer aldığı dizileri yakalar. Ardından gelen sayı en az bir ve en fazla üç haneli olabilir; ondalık kısmı ise opsiyoneldir. Hem

nokta (.) hem virgül (,) ayırıcı olarak kabul edilir. %5, %25.4, %99,99 gibi diziler “Percentage_Numbers” etiketiyle döndürülür. İkinci ifade % sembolünün sonda olduğu yapıları yakalar. Bu kullanım Türkçe için doğru gösterim olmadığı için birimlendiricinin güncel sürümünde işlevsel olarak kullanılmamaktadır.

Son ifade ise yüzde işaretiyle başlayan ancak ardından gelen sayı dışında başka karakterleri de içeren diziler için tanımlanmıştır. Örneğin %50indirim, %12artış gibi diziler bu ifade tarafından yakalanır. Mevcut sürümde birimlendirilmeksizin “Percentage_Numbers” etiketiyle döndürülür.

Tek Tireli Yapılar: Bu düzenli ifade, iki sözcüğün tire (-) karakteri ile birleştirilmiş biçimlerini hedef alır. Düzenli ifade, tire işaretinin her iki tarafında da noktalama işareti bulunmaması gerektiği belirtilmiştir.

- "single_hyphen": re.compile(rf'^[^{puncs}]+-[^{puncs}]+$')

  “ara-sıra”, “ileri-geri” gibi yapılar bu ifade tarafından yakalanır.

Çok Tireli Yapılar: Bu düzenli ifade, ikiden fazla sözcüğün tire (-) karakteri ile birleştirilmiş biçimlerini hedef alır. Yine tire işaretinin her iki tarafında da noktalama işareti bulunmaması gerektiği belirtilmiştir.

- "multi_hyphen": re.compile(rf'^[^{puncs}]+(-[^{puncs}]+)+$')

  “ilk-orta-lise”, “üç-beş-sekiz” gibi diziler bu ifade tarafından yakalanır.

Tek Alt Çizgili Yapılar: Bu düzenli ifade, iki sözcüğün alt çizgi (_) karakteri ile birleştirilmiş biçimlerini hedef alır. Alt çizgi işaretinin her iki yanında da noktalama işareti bulunmamalıdır.

- "single_underscore": re.compile(rf'^[^{puncs}]+_[^{puncs}]+$')

Çok Alt Çizgili Yapılar: Bu düzenli ifade, ikiden fazla sözcüğün alt çizgi (_) karakteri ile birleştirilmiş biçimlerini hedef alır. Alt çizgi işaretinin her iki yanında da noktalama işareti bulunmamalıdır.

- "multi_underscore": re.compile(rf'^[^{puncs}]+(_[^{puncs}]+)+$')

Alt çizgili diziler çoğunlukla teknik metinlerde veya etiketleme sistemlerinde anlamlı birimler olarak yer alır ve bu nedenle değerlendirilmeye alınmıştır. Verinin incelenmesi sırasında özellikle dosya adlarının alt çizgi kullanılarak yazıldığı gözlenmiştir. Gelecek sürümlerde dosya uzantılarıyla birleştirilerek yeni bir etiket olarak kullanılması planlanmaktadır.

Birimlendirici Fonksiyonları

Bu bölümde temel birimlendirici fonksiyonları ele alınmıştır.

Metin Tamiri

Birimlendiriciye aktarılan her bir girdi diziye birimlendirmeden önce bir *tamir* işlemi uygulanmaktadır. CharFix sınıfı bu işlemleri gerçekleştirmek amacıyla tasarlanmıştır. Bu sınıf altında tanımlanan 11 metod ile girdi dizideki karakter kodlamasından doğan hatalar temizlenmekte, tırnak işaretleri düzenlenerek standart hale getirilmekte, HTML dilinde kullanılan karakterler düzenlenmekte ve UTF-8 karakter kodlamasında kontrol karakteri işlevi gören karakterler temizlenmektedir.

Karakter kodlaması düzenlenmesi

Bilgisayar için her karakter bir sayısal değerle ifade edilmektir. Bu sayısal değerin hangi karakterin karşılığı olduğu karakter kodlaması seti içinde tanımlanır. Erken dönem metin işleme sistemleri, karakterleri temsil etmek için genellikle 7-bit ASCII[12] kodlamasını temel almıştır. Ancak bu sistem, yalnızca İngilizce karakter kümesini destekleyebilecek sınırlı bir alan sağladığından, farklı dillerin özel karakterlerini temsil etmede yetersiz kalmıştır.

Türkçeye özgü karakterlerin doğru şekilde kodlanabilmesi amacıyla, ISO/IEC tarafından yayımlanan ISO-8859 karakter setlerinin dokuzuncu versiyonu olan ISO-8859-9

[12] American Standard Code for Information Interchange

(Latin-5), Türkçe'de kullanılan 'ç', 'ğ', 'ı', 'ö', 'ş', 'ü' gibi karakterleri desteklemek üzere geliştirilmiştir. Microsoft tabanlı sistemlerde ise bu karakterlerin karşılığı genellikle Windows-1254 karakter seti aracılığıyla sağlanmıştır. Bu iki karakter kümesi, aynı karakterler için farklı bayt değerleri atayabildiğinden, karakterlerin platformlar arasında bozulmadan aktarılması her zaman mümkün olmamıştır. Bu tür sorunlar özellikle web temelli uygulamalarda, veritabanı entegrasyonlarında ve doğal dil işleme projelerinde ciddi uyumsuzluklara yol açabilmektedir. Aynı zamanda dilsel verinin kullanıldığı çalışmalarda da kullanılan yazılımların başarımını olumsuz yönde etkilemektedir (Kubilay, 2024).

Bu bağlamda, Unicode ve UTF-8 gibi çok baytlı kodlama sistemleri bir standartlaşma ihtiyacının sonucu olarak ortaya çıkmıştır. 1980'lerin sonunda bütünleşik bir karakter seti oluşturmak için iki ayrı kuruluş[13] tarafından bağımsız çalışmalar yürütülmekteydi (Kuhn, 1999). Unicode Konsorsiyumu tarafından geliştirilen ve ISO/IEC 10646 standardı ile uyumlu olan UTF-8 kodlaması, hem geriye dönük ASCII uyumluluğunu korumakta hem de tüm dünya dillerine ait karakterleri tek bir kodlama çatısı altında temsil edebilmektedir (Yergeau, 2003). Özellikle çok dilli metin işleme ve dil verisi analizinde, Unicode tutarlı bir temsil sağlar. Unicode Standardı, her karakteri benzersiz bir kod noktası (code point) ile tanımlar ve bu karakterlerin bilgisayar sistemlerinde nasıl saklanacağına ve gösterileceğine dair çeşitli dönüştürme biçimleri sunar. En yaygınları arasında UTF-8, UTF-16 ve UTF-32 yer alır (Moran & Cysouw, 2018). UTF-8, değişken uzunlukta bayt dizileri kullanarak karakterleri temsil eder ve bu özelliği sayesinde sistemler arası taşınabilirliği ve metin bütünlüğünü güvence altına alır.

Tasarlanan birimlendiriciye entegre olarak çalışan char_fix modülü karakter kodlaması çeşitliliğinden kaynaklanan karakter çarpıklıklarını ve bozulmuş metinleri tespit ederek normalize etmeye yönelik olarak tasarlanmıştır. Bu modül temel olarak, Windows-1254 gibi eski kodlamalardan gelen kalıntıları tanıyarak, UTF-8'e dönüşüm sırasında

[13] Uluslararası Standardizasyon Örgütü (ISO) ve çok dilli yazılım üreten bir konsorsiyum tarafından organize edilen Unicode Projesi

oluşmuş hataları düzenler ve Türkçe karakter kümesine uygun biçimde yeniden yapılandırır. Bu hatalarının düzeltilmesi, yalnızca görsel düzeltme sağlamakla kalmaz; aynı zamanda doğru birimlendirme ve takip eden DDİ adımları için de tutarlılık sağlar.

HTML Referansları

Günümüzde büyük metin verilerinin temel kaynağı internettir. Bu çalışmada kullanılan üç veri seti de internet kaynaklarından elde edilen metinler ile oluşturulmuştur (bkz. *Veri Toplama Süreci*). Web temelli metinlerin işlenmesinde sıklıkla karşılaşılan sorunlardan biri, karakterlerin HTML (Hypertext MarkUp Language) *karakter referansları* (ya da “entity”leri) biçiminde kodlanmış olmasıdır. Bu referanslar hem özel karakterlerin çakışmalara neden olmaması için hem de farklı karakter kümelerinin doğru biçimde aktarılabilmesi amacıyla geliştirilmiştir. Bu karakterler, genellikle Unicode karşılıklarına çözülmeden işlenirse metin analizi sürecini olumsuz etkileyebilir.

Örneğin, ö HTML referansı, Unicode sisteminde U+00F6 kod noktasıyla temsil edilen “ö” harfine karşılık gelir. Benzer şekilde ç ifadesi, ç karakterinin Unicode karşılığı olan U+00E7'yi gösterir. Bu tür referansların çözülmemesi durumunda, bir metin üzerinde yapılan birimlendirme işlemi verimliliği ve başarımı bozulabilir. Aşağıdaki örnek kodda hem metin tamiri hem HTML referansların düzenlemesi işinin CharFix sınıfı tarafından nasıl yapıldığı örneklenmiştir.

```
from ts_tokenizer.char_fix import CharFix
line = "Bug&uuml;n hava &ccedil;ok gÃ¼zel."
print(CharFix.fix(line))  # Fixes corrupted characters
```

Bugün sözcüğünde geçen “ü” ve çok sözcüğünde geçen “ç” karakterleri HTML referansı, güzel sözcüğünde geçen “ü” ise UTF-8 olmayan karakterlerdir. Kod çalıştırıldığında ekrana “Bugün hava çok güzel.” şeklinde düzenlenmiş çıktıyı getirecektir.

Büyük/Küçük Harf Dönüşümü

Hazırlanan birimlendirici Python3 programlama dilinde yazılmıştır. Python 3 her ne kadar yerleşik olarak UTF-8 (Unicode) desteği vermekte olsa da “I” “ı” ve “İ” “i” büyük/küçük harf dönüşümünde beklenilen çıktıyı verememektedir. Aşağıdaki kod örneğinde büyük/küçük harf dönüşümü Python 3.12.3 sürümünde yerleşik olarak bulunan lowercase() metodu örneklenmiştir.

```
taner@ts-idea:~/Desktop$ python3
Python 3.12.3 (main, Feb  4 2025, 14:48:35) [GCC 13.3.0] on linux
Type "help", "copyright", "credits" or "license" for more
information.
>>> word = "İstanbul"
>>> print(word.lower())
i̇stanbul
>>> word = "Iğdır"
>>> print(word.lower())
iğdır
```

Python yorumlayıcısının ürettiği çıktıda görüldüğü üzere “İstanbul” sözcüğünün küçük harf dönüşümünde “İ”, “Iğdır” sözcüğünün dönüşümünde “I” beklenilen biçimde değildir. Bu sebeple CharFix sınıfına tr_lowercase metodu eklenmiştir. Aşağıda bu metodun kullanımı örneklenmiştir.

```
from ts_tokenizer.char_fix import CharFix
line = "İstanbul ve Iğdır ''arası'' 1528 km'dir."
print(CharFix.tr_lowercase(line))
$ istanbul ve ığdır ''arası'' 1528 km'dir.
```

Görüldüğü üzere, büyük/küçük harf dönüşümü tr_lowercase metodu tarafından beklenen biçimde bir çıktı üretmektedir.

Tırnak İşaretlerinin Standartlaştırılması

Metin içinde farklı eş görev yapan farklı tırnak işaretleri sıklıkla kullanılmaktadır. Özellikle iki tek tırnak işareti ile çift tırnak işaretinin yazılması çalışma kapsamında işlenen veri setinde sıklıkla karşılaşılan bir problem olarak tespit edilmiştir. Bu kullanım birimlendirici açısından fazladan birim üretilmesine sebep olmaktadır. Örneğin "yeni" ve “yeni” dizileri insanlar tarafından aynı algılansa da ilk dizi [‘ - ‘ - *yeni* - ‘ ‘] ikinci dizi ise [“ - *yeni* - “] biçiminde birimlendirilmektedir. Bu durumu standart hale getirmek için CharFix sınıfına fix_quote metodu eklenmiştir. Aşağıda bu metodun kullanımı örneklenmiştir.

```
from ts_tokenizer.char_fix import CharFix
line = "İstanbul ve Iğdır ''arası'' 1528 km'dir."
print(CharFix.fix_quote(line))
$ İstanbul ve Iğdır "arası" 1528 km'dir.
```

Görüldüğü gibi arası sözcüğünü çevreleyen tırnak işaretleri standart hale getirilmiştir.

CharFix sınıfında yer alan ve karakter düzenleyen tüm metodlar fix metodu altında toplanmıştır. Böylelikle karakter kodlaması, HTML referansları, Unicode kontrol karakterlerinin temizlenmesi ve tırnak işaretlerinin düzenlemesi tek bir işlev haline getirilmiştir.

Tarih Kontrolü

Sosyal bilimler açısından incelenen dil verisinde tarih ifade eden yapılar, zaman odaklı çıkarım, kronolojik sıralama vb. işlevlerin yerine getirilmesi için önem taşımaktadır. Bu sebeple hem “gün/ay/yıl” hem “ay/gün/yıl” olarak iki farklı yazım tarzında tarih belirten dizilerin yakalanması ve yakalanan dizinin geçerliliğinin kontrolü için özel bir sınıf hazırlanmıştır. Bu sınıf altında yer alan “is_valid_date_dd_mm_yyyy” metodu gün/ay/yıl,

“is_valid_date_mm_dd_yyyy” metodu ay/gün/yıl olarak yazılmış dizileri yakalarken “_is_valid_day_month_year” metodu bu dizilerin geçerliliğini kontrol etmektedir. Örneğin 30.02.2025 veya 02.30.2025 Şubat ayında 30. gün olmaması sebebiyle kontrolden geçemeyecektir.

```
taner@ts-idea:~/test$ cat test_text.txt
30.02.2025 veya 02.30.2025
22.02.2016 veya 24.06.1983
taner@ts-idea:~/test$ ts-tokenizer -o tagged test_text.txt
30.02.2025  Number
veya    Valid_Word
02.30.2025  Number
22.02.2016  Date
veya    Valid_Word
24.06.1983  Date
```

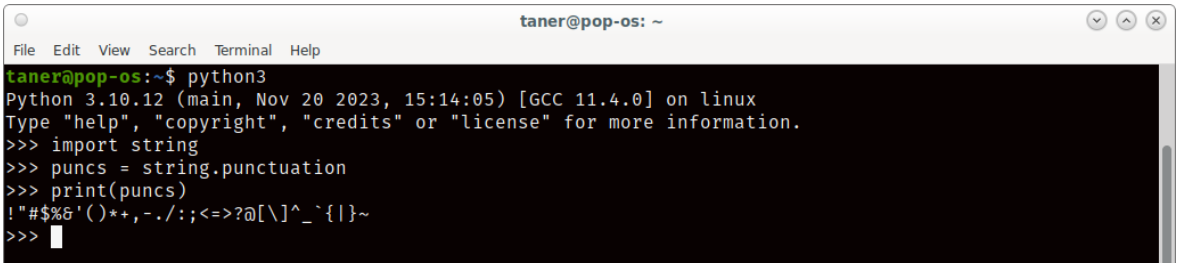
Yukarıdaki örnekte metin dosyası içinde 4 ayrı tarih belirtme olasılığı olan dizi vardır. İlk satırda bulunan iki dizi tarih kontrolünde geçersiz bulunarak sayı dizisi olarak döndürülürken ikinci satırda bulunan diziler “Date” olarak döndürülmüştür.

Noktalama İşaretleri

Birimlendirme konusundaki en önemli sorunlardan biri noktalama işaretlerinin işlenmesidir. Dizi içindeki noktalama işaretlerinin konumu ve sayısının belirlenmesi amacıyla “PuncMatcher” ve “PuncTagCheck” sınıfları hazırlanmıştır. Bu sınıflar Pyhon içinde tanımlı string kütüphanesinden faydalanır. *string* kütüphanesinde tanımlı noktalama işaretleri Şekil 16’a gösterilmiştir.

Şekil 16

Python string kütüphanesinin içerdiği noktalama işaretleri



PuncMatcher sınıfı altında yer alan “punc_count” dizi içindeki noktalama işaretlerinin sayısını, “punc_pos” noktalama işaretlerinin konumu belirlemektedir. Aynı sınıf altında yer alan “find_punctuation” metodu ise diğer metodlardan yararlanarak noktalama işaretlerini etiketleyerek döndüren bir metoddur. Bu metod aşağıda verilen etiketleri diziye iliştirerek birimlendirme yapılmasını sağlamaktadır.

FSP (Final Single Punctuation/Sonda Tek Noktalama): Dizinin tamamında yalnızca bir tane noktalama işareti olduğunu ve bu noktalama işaretinin dizinin sonunda olduğunu belirtir. Örneğin bilgisayar. dizisi FSP etiketiyle birimlendiriciye aktarılmaktadır.

ISP (Initial Single Punctuation/Başta Tek Noktalama): Dizinin tamamında yalnızca bir tane noktalama işareti olduğunu ve bu noktalama işaretinin dizinin ilk karakteri olduğunu belirtir. Örneğin .bilgisayar dizisi ISP olarak etiketlenerek birimlendirme işlemine alınmaktadır.

FMP (Final Multiple Punctuation/Sonda Çoklu Noktalama): Dizinin tamamında birden fazla noktalama işareti olduğunu ve bu noktalama işaretlerinin dizinin sonunda ardışık olarak bulunduğunu belirtir. bilgisayar… dizisi FMP etiketiyle birimlendiriciye aktarılır.

IMP (Initial Multiple Punctuation/Başta Çoklu Noktalama): Dizinin tamamında birden fazla noktalama işareti olduğunu ve bu noktalama işaretlerinin dizinin başında ardışık olarak bulunduğunu belirtir. …bilgisayar dizisi IMP etiketiyle birimlendiriciye aktarılır.

MSP (Multiside Punctuation/Başta ve Sonda Noktalama): Bu etiket dizinin tamamında birden fazla noktalama işareti olduğunu ve bu noktalama işaretlerinin dizinin başında ve sonunda yer aldığını belirtir. ..bilgisayar. dizisi birimlendiriciye MSP etiketiyle aktarılır.

MSSP (Multiside Single Punctuation/Başta ve Sonda Tekli Noktalama): Bu etiket dizinin tamamında yalnızca iki adet noktalama işareti olduğunu ve bunların dizinin başında ve sonunda birer adet olarak bulunduğunu bildirir.

Yukarıdaki 6 yapı *find_punctuation* metodu tarafından yakalanır.

Şekil 17

find punctuation metodu

```python
@classmethod
def find_punctuation(cls, word: str) -> Optional[Union[bool, str]]:
    match = re.match(PuncPattern, word)
    if not match:
        return None

    initial_punc = match.group('initial')
    final_punc = match.group('final')

    if initial_punc and final_punc:
        if len(initial_punc) == 1 and len(final_punc) == 1:
            return "MSSP"  # Multi-Side Single Punctuation
        else:
            return "MSP"  # Multi-Side Punctuation
    elif initial_punc:
        if len(initial_punc) == 1:
            return "ISP"  # Initial_Single_Punc
        else:
            return "IMP"  # Initial_Multi_Punc
    elif final_punc:
        if len(final_punc) == 1 and PuncMatcher.punc_count(word) == 1:
            return "FSP"  # Final_Single_Punc
        else:
            return "FMP"  # Final_Multi_Punc
    return None
```

Bu altı ayrı tanımlama noktalama işareti içeren dizilerin birimlendirici tarafından nasıl ele alınacağını ve hangi kurallara göre birimlendirileceğini tanımlamak amacıyla birer belirteç olarak kullanılmaktadır. Bu etiketler nihai çıktıda verilmemektedir.

Bunların dışında noktalama işaretlerinin dizi içindeki konumu ve sayısını belirten iki ayrı kontrol mekanizması da vardır. Ancak bu yapılar birimlendiricinin tasarımı gereğince “token_handler” sınıfı altında işlenmektedir.

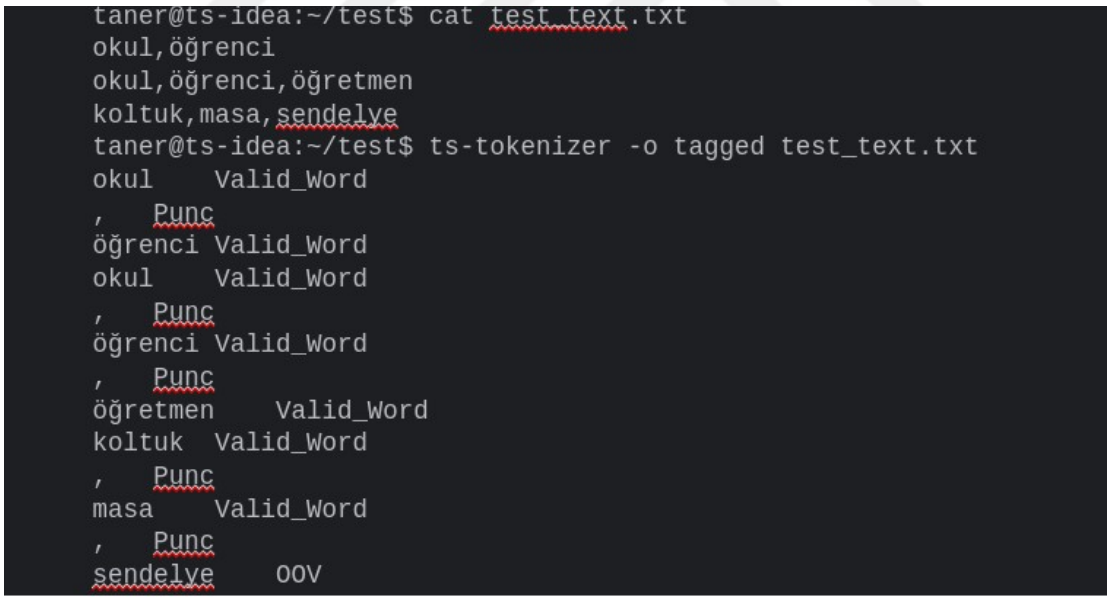
MIDSP (Middle Single Punctuation/Ortada Tekli Noktalama): Dizinin tamamında yalnızca bir adet noktalama işareti olduğunu ve bu noktalama işaretinin dizinin ilk veya son karakteri olmadığını belirtir. okul,öğrenci dizisi MIDSP etiketiyle döndürülür.

MIDMP (Middle Multiple Punctuation/Ortada Çoklu Noktalama): Dizinin birden fazla noktalama işareti içerdiğini, bu noktalama işaretlerinin dizinin ilk ve son karakteri olmadığını belirtir. okul,öğrenci,öğretmen dizisi MIDMP etiketiyle birimlendiriciye aktarılır.

Her iki fonksiyon, yakaladığı çıktıyı tekrarlamalı (recursive) olarak işlemektedir. Böylelikle noktalama işaretlerinin ayrıştırılması sonrasında elde edilen dizilerin de kontrolü sağlanmaktadır. Aşağıdaki örnek verilen dizilerin çıktıya nasıl döndürüldüğünü göstermektedir.

Şekil 18

Dizi içi tekil noktalama işaretlerinin birimlendirilmesi



Örnekte her satırda verilen diziler noktalama işareti temel alınarak ayrıştırılmış, ardından bu diziler tekrar işlenerek ilgili etiketle ilişkilendirilmiştir. Tanımlanmış tüm fonksiyonlar belirli kurallara göre diziyi inceler ve diziye birimlendirme için temel olacak bir etiket atar.

Bu düzenli ifadeler de belirli bir hiyerarşiyi takip etmektedir. Bu hiyerarşi TokenPreProcess sınıfı altında single_punc ve multi_punc olmak üzere iki ayrı listede tanımlanmıştır.

Bölüm 5

Bulgular, Yorumlar ve Tartışma

TS Tokenizer kolay kullanım, karakter düzeltme işlevlerin otomatik uygulanması, büyük hacimli dosyaların kolaylıkla işlenebilmesi amacıyla tasarlanmıştır. Temel hedef yüksek teknoloji okuryazarlığına ihtiyaç duyulmadan kullanılabilir olmasının sağlanmasıdır. Bununla birlikte uçtan uca tasarlanmış yazılım zincirleri içinde diğer doğal dil işleme süreçlerine entegre edilebilmesi de mümkündür.

Kurulum

TS Tokenizer, kullanıcıların kolay erişimi ve farklı kullanım senaryolarına uygulanabilirliği amacıyla hem Python paketi olarak PyPI (Python Package Index) üzerinde hem de kaynak kod düzeyinde GitHub üzerinden yayımlanmıştır. Bu sayede kullanıcılar, ister kararlı sürüm paketiyle hızlı bir şekilde kurulumu gerçekleştirebilir, isterse kaynak kod düzeyinde geliştirici modunda kullanılabilir.

PyPI üzerinden TS Tokenizer'ın en güncel ve kararlı sürümüne https://pypi.org/project/ts-tokenizer/ adresinden erişilebilir.

Kurulum, güncelleme ve kaldırma için sırasıyla aşağıdaki pip komutları kullanılır.

$ pip install ts-tokenizer

$ pip install --upgrade ts-tokenizer

$ pip uninstall ts-tokenizer

TS Tokenizer'ın kaynak kodlarına https://github.com/tanerim/ts_tokenizer adresinden erişelebilir. Kaynak kodlardan kurulum senaryosunda bir sanal ortam (virtual environment) kullanılması önerilir. Sanal ortamın kurulması için Linux platformunda

$ python3 -m venv [sanal_ortam_adı]

komutunun verilmesi ve

$ source venv/bin/activate

komutuyla bu sanal ortamın aktive edilmesi yeterlidir. Windows ortamında sanal ortamı aktive etmek için

venv\Scripts\activate

komutu kullanılmalıdır.

Ardından Github reposu yerel bir klasöre klonlanarak kurulum yapılır.

$ git clone https://github.com/tanerim/ts_tokenizer.git

$ cd ts_tokenizer

$ pip install .

Birimlendiricinin faydalandığı sözlükler, istisnalar listesi, kısaltmalar listesi vb. sözlüklerde veya düzenli ifadelerde yapılan değişiklerin doğrudan yansıtılması için kurulum düzenlenebilir (editable) modda yapılabilir. Bu durumda aşağıdaki komut kullanılmalıdır.

pip install -e .

Kurulum öncesinde gerekli bağımlılıkların yüklenmesi için

$ pip install -r requirements.txt

komutu kullanılmalıdır. Bu sayede TS Tokenizer ile birlikte çalışan yardımcı kütüphaneler (örneğin: regex, pandas, tqdm, setuptools vb.) sisteme otomatik olarak kurulacaktır.

Bu kurulum adımlarının ardından, TS Tokenizer hem komut satırı (CLI) üzerinden, hem de bir Python modülü olarak doğrudan kullanılabilir.

Temel İşlevler

TS Tokenizer hem komut satırı üstünde hem de Python kütüphanesi olarak kullanılabilecek yapıda tasarlanmıştır. Komut satırı üstünde kullanımı çeşitli parametreler alabilmektedir. Bu parametreler Tablo 9’da verilmiştir.

Tablo 9

Birimlendiricinin aldığı parametreler

Parametre	Kısaltma	Tanım	Ön Tanımlı Değer
--output	-o	tokenized, lines, tagged, tagged_lines	tokenized
--num-workers	-n	İşlemleme sırasında kullanılacak çekirdek sayısını tanımlar	Toplam CPU çekirdeği -1
--version	-V	TS Tokenizer sürümünü gösterir	N/A
--help	-h	Bilgi mesajını ve kullanılabilir parametreleri gösterir	N/A

Tablo 9'da verilen parametreler TS Tokenizer'ın alabildiği parametreleri göstermektedir. --output parametresi birimlendiricinin 4 ayrı biçimde çıktı üretmesini sağlar.

tokenized: Bu parametre ön tanımlı olarak atanmış parametredir. Bir parametre tanımlanmadığında TS Tokenizer çıktıyı --o tokenized parametresiyle işler. Bu parametre verilen metni her satırda bir sözcük olacak biçimde döndürür.

lines: Bu parametre verilen metindeki satır sonu işaretini temel alarak çıktıyı aynı düzende koruyarak üretir.

tagged: Bu parametre kullanıldığında birimlendirici algoritmasında kullanılan token_handler ve ts_tokenize sınıflarının girdi diziyle ilişkilendirildiği son etiket çıktıya eklenir.

tagged_lines: Bu parametre satır sonu işareti ile ayrıştırılmış bölümleri koruyarak tagged parametresinin ürettiği çıktıyı döndürür.

Tablo 10 parametrelerin döndürdüğü çıktıyı "Defne, sergilediği Tinker Bell performansÄ±yla izleyicileri bÃ¼yÃ¼ledi; sahnedeki görÃ¼ntÃ¼sÃ¼, kostÃ¼mÃ¼ ve mÃ¼zikle uyumu herkesi etkiledi." örneği üzerinden göstermektedir.

Tablo 10

Birimlendiricinin verilen parametreler ile ürettiği çıktılar

Parametre	Açıklama	Çıktı
--o tokenized	Her bir birimi ayrı satırlarda verir	Defne , sergilediği Tinker Bell performansıyla izleyicileri büyüledi ; sahnedeki görüntüsü , kostümü ve müzikle uyumu herkesi etkiledi .
--o tagged	Her birimi ve etiketini tab karakteriyle ayırarak verir	Defne Valid_Word , Punc sergilediği Valid_Word Tinker English_Word Bell English_Word performansıyla Valid_Word izleyicileri Valid_Word büyüledi Valid_Word ; Punc sahnedeki Valid_Word görüntüsü Valid_Word , Punc kostümü Valid_Word ve Valid_Word müzikle Valid_Word uyumu Valid_Word herkesi Valid_Word etkiledi Valid_Word . Punc
--o lines	Satır başı ve satır sonu arasında kalan tüm dizileri birimlendirerek	Defne , sergilediği Tinker Bell performansıyla

Parametre	Açıklama	Çıktı
	biçimlendirmeyi koruyan bir çıktı üretir.	izleyicileri büyüledi ; sahnedeki görüntüsü , kostümü ve müzikle uyumu herkesi etkiledi .
--o tagged_lines	Satır başı ve satır sonu arasında kalan tüm dizileri birimlendirerek biçimlendirmeyi koruyan ve ilgili etiketleri iliştiren bir çıktı üretir.	[('Defne', 'Valid_Word'), (',', 'Punc'), ('sergilediği', 'Valid_Word'), ('Tinker', 'English_Word'), ('Bell', 'English_Word'), ('performansıyla', 'Valid_Word'), ('izleyicileri', 'Valid_Word'), ('büyüledi', 'Valid_Word'), (';', 'Punc'), ('sahnedeki', 'Valid_Word'), ('görüntüsü', 'Valid_Word'), (',', 'Punc'), ('kostümü', 'Valid_Word'), ('ve', 'Valid_Word'), ('müzikle', 'Valid_Word'), ('uyumu', 'Valid_Word'), ('herkesi', 'Valid_Word'), ('etkiledi', 'Valid_Word'), ('.', 'Punc')]

tagged parametresinin döndürdüğü etiketin “sözcük türü etiketi” olmadığı, bu etiketlerin dizisinin işlevi veya yapısını belirttiğine dikkat edilmelidir.

Etiketler

TS Tokenizer ürettiği çıktılara etiketler iliştirebimektedir. Ancak bu etiketler sözcük türü etiketleri değildir. Etiketli yapı, birimlendiricinin diziyi nasıl ele aldığını göstermek veya e-posta, mention, tarih vb. işlevsel dizileri yakalayabilmek için tanımlanmıştır. Tablo 10’da verilen örnekler incelendiğinde 31.10.1975 dizisinin “Date” etiketiyle döndürüldüğü görülecektir. Birimlendirici verilen diziyi geçerli bir tarih olarak algılamış, bu sebeple bütünlüğünü koruduğu bir birim olarak döndürmüştür.

TS Tokenizer algoritmasında toplam 51 farklı etiket tanımlanmıştır. Bu etiketlerden bazıları token_handler altında tanımlı birimlendirici algoritmasına dizinin tekrar işlenmesi

gerektiğini bildirmektedir. Örneğin is_isp etiketiyle dönen “,gel” dizisini tekrar işlenerek “,” ve “gel” olarak iki ayrı birim olarak döndürülmektedir. Bu noktada “,” için Punc, “gel” için Valid_Word etiketi nihai çıktıda verilmektedir.

Tablo 11

Birimlendiricinin kullandığı ve iliştirdiği etiketler

Fonksiyon	Örnek	Gösterim	Etiket
is_mention	@ts-tokenizer	Evet	Mention
is_hashtag	#ts-tokenizer	Evet	Hashtag
is_in_quotes	“ts-tokenizer”	Hayır	--
is_numbered_title	(1)	Evet	Numbered_Title
is_in_paranthesis	(bilgisayar)	Hayır	--
is_date_range	01.01.2024-01.01.2025	Evet	Date_Range
is_complex_punc	-yeniden,sonradan..	Hayır	--
is_date	22.02.2016	Evet	Date
is_hour	14.05	Evet	Hour
is_hour_suffix	14.05’te	Evet	Hour_Suffix
is_percentage_numbers	%75	Evet	Percentage_Numbers
is_percentage_numbers_chars	%75’lik	Evet	Percentage_Numbers
is_roman_number	XII	Evet	Roman_Number
is_bullet_list	•Giriş	Evet	Bullet_List
is_email	tanersezerr@gmail.com	Evet	Email
is_email_punc	tanersezerr@gmail.com!	Hayır	---
is_full_url	https://tscorpus.com	Evet	Full_URL
is_web_url	www.tscorpus.com	Evet	Web_URL
is_full_url (suffix)	www.tscorpus.com'un	Evet	URL_Suffix
is_copyright	©tscorpus	Evet	Copyright
is_registered	tscorpus®	Evet	Registered
is_trademark	tscorpus™	Evet	Trademark
is_currency	100₺	Evet	Currency
is_num_char_sequence	380A	Hayır	---
is_abbr	TBMM	Evet	Abbr
is_in_lexicon	bilgisayar	Evet	Valid_Word
is_in_exceptions	e-mail	Evet	Exception
is_in_eng_words	computer	Evet	English_Word
is_smiley	:)	Evet	Smiley
is_multiple_smiley	:) :)	Hayır	---
is_emoticon		Evet	Emoticon
is_multiple_emoticon		Hayır	---
is_multiple_smiley_in	hey:):)	Hayır	---
is_number	175.01	Evet	Number
is_apostrophed	Türkiye’nin	Evet	Apostrophed
is_single_punc	!	Evet	Punc
is_multi_punc	!!	Hayır	---

is_non_latin	한국드	Evet	Non_Latin
is_single_hyphenated	sabah-akşam	Evet	Single_Hyphenated
is_multi_hyphenated	sabah-öğle-akşam	Evet	Multi_Hyphenated
is_single_underscored	Gel_Git	Evet	Single_Underscored
is_multi_underscored	Gel_Gör_Yen	Evet	Multi_Underscored
is_one_char_fixable	bilgisa¬yar	Evet	One_Char_Fixed
is_three_or_more	heyyyyy	Hayır	---
is_fsp	bilgisayar.	Hayır	---
is_isp	.bilgisayar	Hayır	---
is_fmp	bilgisayar..	Hayır	---
is_imp	..bilgisayar	Hayır	---
is_msp	--bilgisayar--	Hayır	---
is_mssp	-bilgisayar-	Hayır	---
is_midsp	okul,öğrenci	Hayır	---
is_midmp	okul,öğrenci,öğretmen	Hayır	---

Python İçinde Kullanım Özellikleri

TS Tokenizer Python programlama diline çağrılarak da kullanılabilmektedir. Aşağıda bu kullanım yöntemine ilişkin örnekler verilmiştir.

```
from ts_tokenizer.tokenizer import TSTokenizer
sample = "ParÃ§a ve bÃ¼tÃ¼n iliåÿkisi her zaman iåÿlevsel deðildir."
token_list = TSTokenizer.ts_tokenize(sample, output_format="tokenized")
for token in token_list:
   print(token)
```

Örnekte TSTokenizer metodu çağrılmış, verilen cümle birimlendirilerek ekrana yazdırılmıştır. Birimlendirici karakter kodlaması sorunlarını düzelterek çıktı üretmektedir.

Lisans

TS Tokenizer MIT lisansı kapsamında açık kaynak olarak sunulmaktadır. Bu lisans yazılımın özgür kullanımını garanti altına almaktadır. MIT Lisansı'nın tercih edilmesinin temel gerekçesi, projenin “açık bilim ilkeleri” doğrultusunda “erişilebilir, şeffaf ve yeniden üretilebilir” olmasını sağlamaktır. Bu sayede çalışmanın **etik, açık** ve **sürdürülebilir** olması amaçlamaktadır. Lisansın sağladığı hükümler EK E’de verilmiştir.

Örnek Uygulama

Hazırlanan birimlendirici ile iki ayrı veri seti kullanılarak örnek bir uygulama yapılmıştır. İlk veri seti, söz varlığı çalışmalarına sıklıkla konu edilen ders kitaplarından, ikinci veri seti ise tweetlerden oluşturulmuştur.

Birinci veri seti, Türkiye Cumhuriyeti Millî Eğitim Bakanlığı (MEB) tarafından 2024-2025 eğitim-öğretim yılında ortaokullarda okutulan Türkçe ders kitaplarında yer alan okuma parçalarının bir araya getirilmesiyle oluşturulmuştur. Bu metinler, editör denetiminden geçmiş olmaları sebebiyle *temiz veri* kapsamında değerlendirilmiştir. Üç ayrı kitaptan toplam 39.870 sözcük içeren bir veri kümesi elde edilmiştir.

İkinci veri seti ise, birimlendiricinin gürültülü ve düzenlenmemiş veride nasıl bir başarım gösterdiğini test edebilmek amacıyla sosyal medya verisinden oluşmaktadır. Sosyal medyadan gelen metinler dilbilgisel hatalar içeren ve gürültülü bir veri sunmaktadır (Derczynski ve diğerleri, 2013). Ancak hem sunduğu verinin hacimsel büyüklüğü hem de çeşitliliği yüzünden dil çalışmaların için verimli bir kaynak sunmaktadır (Gimpel ve diğerleri, 2013). Bu amaçla 35 bin 284 tweetten oluşan Türkçe Sosyal Medya Ofansif Dil Derlemi (troff-v 1.0) (Çöltekin, 2020) kullanılmıştır.

Birimlendirilmiş verinin karşılaştırılması için NLTK içinde yer alan WordTokenize (wt), WordPunctTokenize (wpt), WhitespaceTokenizer (ws) ve TreebankWordTokenizer (tb) tarafından üretilen birimlendirilmiş çıktı ile TS Tokenizer tarafından üretilmiş çıktı karşılaştırılmıştır. TS Tokenizer çıktısında OOV (sözlük dışı sözcükler) doğru şekilde birimlendirilmemiş diziler olarak kabul edilmiştir. NLTK birimlendiricilerinin ürettiği çıktıysa noktalama işaretleri ve verilen geçerli sözcük listeleriyle tam eşleşme dışında kalan diziler OOV olarak kabul edilmiştir.

Tablo 12

MEB Derlemi birimlendirici değerlendirmesi

Birimlendirici	Toplam Birim	Sözlük İçi Birim	Sözlük İçi Birim Oranı	Sözlük Dışı Birim	Sözlük Dışı Birim Oranı
TS Tokenizer	49.080	48.519	%98.86	561	%1.14
TreeBank	44.148	32.713	%74.10	11.435	%25.90
WordTokenizer	50.083	40.246	%80.36	9837	%19.64
WordPunct	49.745	40.115	%80.64	9630	%19.36
Whitespace	39.870	25.899	%64.96	13.971	%35.04

Tablo incelendiğinde MEB derleminde yer alan metinlerde TS Tokenizer'ın %98.86 oranında başarımla birimlendirme yaptığı görülmektedir. Bu veri setinin içerdiği temiz veride NLTK birimlendiricilerinin eriştiği en yüksek başarım WordPunct ve WordTokenizer ile yakalanmış ve %80 oranında gerçekleşmiştir.

Tablo 13

Sosyal medya verisi birimlendirici değerlendirmesi

Birimlendirici	Toplam Birim	Sözlük İçi Birim	Sözlük İçi Birim Oranı	Sözlük Dışı Birim	Sözlük Dışı Birim Oranı
TS Tokenizer	618.258	571.092	%92.37	47.166	%7.63
TreeBank	611.609	423.055	%69.17	188.554	%30.83
WordTokenizer	638.602	463.079	%72.51	175.523	%27.49
WordPunct	649.270	471.058	%72.55	178.212	%27.45
Whitespace	546.457	345.312	%63.19	201.145	%36.81

Daha gürültülü veriden oluşan sosyal medya verisinin birimlendirmesinde TS Tokenizer'ın başarım oranı %92.37 olarak gözlenmiştir. NLTK birimlendiricileriyse bu veri setinde en yüksek oran yine %72 ile WordPunct ve WordTokenizer ile erişilmiştir.

Her iki derlemin TS Tokenizer tarafından *-o tagged* parametresiyle elde edilen birimlendirilmiş çıktısında iliştirilen etiketler ve gözlenme sıklıkları Tablo 14'te verilmiştir.

Tablo 14

TS Tokenizer ile etiketlenmiş birimlendirici çıktısında gözlenen etiket sıklıkları

MEB Derlemi		**Sosyal Medya Derlemi (troff-v 1.0)**	
Etiket	**Sıklık**	**Etiket**	**Sıklık**
Valid_Word	38074	Valid_Word	445107
Punc	9399	Punc	70407
Apostrophed	682	OOV	47166
OOV	561	Mention	26782
Number	162	Number	7129
English_Word	91	Hashtag	5300
Abbr	39	Apostrophed	3703
Single_Hyphenated	26	English_Word	3619
Ordinal_Number	15	Abbr	3224
One_Char_Fixed	11	Emoticon	2692
Roman_Number	6	Smiley	1006
Bullet_List	6	Non_Latin	534
Exception	3	Ordinal_Number	280
Date_Range	3	One_Char_Fixed	276
Percentage_Numbers	2	Single_Hyphenated	271
Non_Latin	2	Exception	262

		Percentage_Numbers	142
		Roman_Number	112
		Number_Sequence	75
		Hour	70
		Multi_Hyphenated	52
		Numbered_Title	22
		Hour_Suffix	15
		Date	15
		Currency	15
		Date_Range	10
		Multi_Underscored	9
		Bullet_List	8
		Single_Underscored	7
		Email	6
Toplam Birim:	49.344	**Toplam Birim:**	618.316

Tablo 14, TS Tokenizer'ın sunduğu etiket iliştirilmiş çıktının dağılım sıklığını göstermektedir. Bu çıktıda verilen etiketler, daha önce de anıldığı üzere, sözcük türü etiketi olmamakla birlikte, metin içinde sözcüklerin yüklendiği işlevlerin takibi ve analizi için kullanılabilir bir çıktı sunmaktadır.

Bölüm 6

Sonuç ve Öneriler

Bu çalışmada, hibrit bir yaklaşımla tasarlanmış, eğitim bilimleri, dilbilim, uygulamalı dilbilim, derlem dilbilim, dijital insanî bilimler gibi sosyal bilimler disiplinlerinin ihtiyaçlarına cevap verecek bir birimlendiricinin tasarım ve geliştirme süreci ele alınmıştır. Problem Durumu başlığı altında DDİ'nin günümüzde birimlendiricileri büyük dil modellerinin eğitimi için verinin ön işlemesinin yapıldığı bir araç olarak kullandığı, özellikle 2010 ve sonrasında sözcük altı birimlendirme yöntemlerinin alanı domine ettiği savı tartışılmıştır. Aynı bölümde sözcük altı birimlendirme yöntemleri kullanılarak işlenmiş Clarin ParlaMint-TR 2.1 derleminden örnekler verilerek, bu derlemin ürettiği sözcük listelerinin sosyal bilimler çalışmalarında verimli şekilde kullanılamayacakları örneklerle gösterilmiştir.

Farklı birimlendirme yaklaşımlarının tartışıldığı 2. bölümde DDİ'nin sıklıkla kullandığı BPE ve WordPiece yöntemleriyle hazırlanan birimlendirciler sunulmuş, bu yaklaşımla üretilen çıktılar örneklenmiştir. Bu bölümde, bilgisayar kullanarak dil işleme çalışmaları 1940'lardan itibaren ele alınmıştır. BD ve DDİ arasındaki ayrımın günümüzde hedef ve amaçlar bakımından benzeştikleri ancak BD'nin daha çok "neden" sorusuna yanıt arayarak kuramsal bir tartışmayı da barındırdığı, DDİ'nin ise "nasıl" sorusuna odaklanarak daha çok baştan sona / uçtan uca yazılım zincirleri üretmeye odaklanan bir mühendislik alanına dönüştüğü tartışılmıştır.

Yöntem bölümünde çalışmanın kullandığı veri setinin hacimsel büyüklüğü nedeniyle, hedeflenen çıktıları üretmek amacıyla "bilişimsel dilbilim" ve "veri madenciliği" yöntemlerinin neden seçildiği ve kullanıldığı tartışılmıştır. Çalışmaya konu olan ham verinin içerdiği gürültünün anlamlı kırılım noktaları çerçevesinde kümelenmesi amacıyla K-means ve Elbow metodolojilerinin neden kullanıldığı açıklanmış, sunulan grafikler ile gürültü verinin anlamlı kurallar üretmek için nasıl kullanıldığı örneklenmiştir.

Çalışmanın 4. bölümü olan “Birimlendirici Tasarımı” başlığı altında hazırlanan birimlendiricinin hem düzenli ifadelerden ve hem de sözlüklerden nasıl yararlandığı, kullanılan düzenli ifadelerin işleme alınma sırasının nasıl belirlendiği ortaya konulmuştur. Çalışmanın temel araştırma problemi olan “*Türkçe bir metnin Türkçenin yapısına uygun bir şekilde birimlendirilmesi için kullanılabilecek verimli bir işlemler dizisi nasıl tasarlanabilir?*” sorusuna bu bölümde yanıt verilmiştir. Farklı sözlükler ve düzenli ifadelerden yararlanılarak, hibrit bir yaklaşımla tasarlanacak bir birimlendiricinin amaçlanan hedefler için daha verimli olduğu sonucu elde edilmiştir.

Girdi metnin işlenmesinde iki boşluk karakteri arasında kalan her dizi bir birim adayı olarak ele alınmış,

- birim adayının geçerli sözcük listelerinde bulunan bir sözcükle tam eşleşmesi durumunda birim olarak kabul edilmesi,
- tam eşleşme olmayan durumlarda bir düzenli ifade ile tam eşlemesinin kontrol edilmesi ve eğer eşleşme sağlanmışsa bir birim olarak kabul edilmesi,
- yukarıdaki iki adımda eşleşme sağlanamayan dizilerin içerdikleri karakterlere göre tanımlanarak birimlendirici içinde işlemesi

şeklinde sarmal ve yinelemeli bir algoritmanın kullanılması benimsemiştir. Böylelikle birimlendirme işlemi sırasında, özellikle kirli veride sıklıkla gözlenen boşluk kullanılmadan yazılan diziler, noktalama işaretlerinin aynı dizi içinde birden fazla kez geçmesi gibi sebeplerle ortaya çıkan düzensizlik işlemlenebilmiştir.

Yine çalışmanın 4. bölümünde “Düzenli İfadeler ile Tanımlama” başlığı altında, çalışmanın alt problemlerinden biri olan “*Bir metnin birimlendirilmesinde işlem basamakları arasında nasıl bir hiyerarşi vardır?*” sorusuna cevap aranmıştır. Düzenli ifadelerin “Özelden Genele” ve “Karmaşıklık ve Örtüşme” prensiplerine göre sıralanmasının en verimli çıktıyı ürettiği belirlenmiştir. Bu iki prensip, diğer bir soru olan “*Farklı kaynaklardan gelen veri açısından (haber metinleri, ders kitapları, sosyal medya)*

birimlendirme sürecinde farklı işlem dizileri gerekli midir?” sorusuna da cevap üretmiştir. Haber metinleri ve ders kitapları gibi metinler üreticilerinin kurallara bağlı kalarak ürettikleri, dil bilgisi, imla vb. kuralların dikkatle ele alındığı ve editörel süreçten geçmiş metinler olması sebebiyle bu metinler için daha genel kuralların üretilmesi yeterli olmuştur. Bu metinlerde yazım standartlarının korunmuş olması sebebiyle, örneğin noktalama işaretinin kendinden önce gelen diziye, arada boşluk bırakmaksızın yazılması ve ardından bir boşluk karakterinin bırakılması, işlemlemeyi kolaylaştırmış, daha az kural ile verimli çıktı elde edilmesini sağlamıştır. Öte yandan, sosyal medya mecralarında,

- bir editöryal süreç olmaması,
- üreticilerin farklı yazım tercihleri,
- imla ve dil bilgisi kurallarına uyulması mecburiyetinin olmaması,
- kendine özgü ve genelleşmiş bir takım yazım kurallarının olması,
- başka kullanıcılara, konu başlıklarına, farklı sitelere gönderimlerin yoğunluğu

gibi etkenler verideki gürültüyü belirgin bir şekilde arttırmaktadır. Bu sebeple daha fazla ve özgün kurallara ihtiyaç duyulduğu belirlenmiştir.

Çalışmanın dördüncü bölümünde ele alınan bir başka konu da hazırlanan birimlendiricinin sunduğu fonksiyonlardır. Bu bölümde,

- karakter kodlaması sebebiyle metinlerde gözlenen bozulmaların birimlendirme sırasında düzeltilmesi,
- Python dilinin büyük/küçük harf dönüştürme konusunda Türkçe “I” ve “İ” karakterlerini doğru şekilde ele alamaması sorununun çözümü,
- metin içindeki HTML referanslarının otomatik temizlenmesi,

amacıyla hazırlanan CharFix sınıfı ve altında yer alan metotlar açıklanmış ve örneklendirilmiştir.

5. Bölümde hazırlanan ve TS Tokenizer adıyla anılan birimlendiricinin kurulum ve kullanımı, temel işlevleri, birimlendiricinin referans olarak kullandığı ve çıktıya da yansıtabildiği etiketler örneklerle verilmiştir. Bu bölümde aynı zamanda TS Tokenizer'ın açık bilim ilkeleri doğrultusunda "erişilebilir, şeffaf ve yeniden üretilebilir" şekilde sunulduğu ve MIT lisansı ile dağıtıldığı vurgulanmıştır.

Bütünsel olarak değerlendirildiğinde TS Tokenizer, sözcüksel ve işlevsel bütünlüğün korunarak sosyal bilim disiplinlerini hedef alacak şekilde, hibrit bir yaklaşımla geliştirilmiş bir birimlendiricidir. Hem kolay ve basit bir kullanımla, yüksek teknoloji okuryazarlığı gerekmeden kullanılabilir olması hem de farklı iş akışlarına entegre edilebilmesi amacıyla hem bir Python kütüphanesi[14] olarak hem de açık kaynaklı olarak kod reposu halinde[15] sunulmuştur.

TS Tokenizer'ın 4 farklı türde çıktı üretmesi sağlanarak yalnızca bir birimlendirici olarak değil, farklı amaçlara hizmet edebilecek bir yapıda, kendinden sonra gelen işlem süreçleri için de bir çıktı üretmesi sağlanmıştır. Bu sebeple karakter temizleme gibi fonksiyonların bağımsız olarak da kullanılmasını sağlayacak bir tasarım ortaya konulmuştur.

Hazırlanan birimlendirici mevcut halinde 0.1.20 sürüm numarasına erişmiştir. Gelecek sürümlerde,

- ihtiyaç duyulabilecek daha fazla özel kuralın hazırlanarak birimlendirici algoritmasına eklenmesi,
- sözcük türü (POS) etiketleme entegrasyonu,
- çok birimli yapıların birimlendirme sürecine entegrasyonu

gibi özelliklerle birimlendirici geliştirilebilir.

---

[14] https://pypi.org/project/ts-tokenizer/

[15] https://github.com/tanerim/ts_tokenizer.git

Dil doğası gereği değişen ve gelişen bir olgudur. Her gün yeni bir sözcüğün, farklı bir kullanımın ortaya çıkması, sosyal medya gibi kendi kurallarını geliştiren mecraların açılması olasıdır. Bu sebeple doğal dili işleyen araçlar için nihaî mükemmelliğe ulaşmaktan söz etmek zordur. Dolayısıyla dil işleyen araçlar için, üretildikleri çağın ihtiyaçlarına cevap verecek şekilde tasarlanmalarının daha akılcı bir yaklaşım olduğu söylenebilir. *TS Tokenizer*, bu genel çerçeve içinde, günümüz koşullarında, eğitim bilimleri, derlem dilbilim, dijital insanî bilimler gibi sosyal bilimler disiplinlerinin ihtiyaçlarına cevap vermek amacıyla tasarlanmış ve açık bilim ilkesiyle sunulmuş bir birimlendiricidir.

Kaynaklar

Akbari, A. (2014). An overall perspective of machine translation with its shortcomings. *International Journal of Education and Literacy Studies, 2*(1), 1–10.

Aksoy, A. (2022). *Augmenting a Turkish dataset for spam filtering using natural language processing techniques* (Yüksek lisans tezi). Orta Doğu Teknik Üniversitesi, Türkiye.

Almaaytah, S. A. (2024). Arabic word tokenization system using the maximum matching model. *Edelweiss Applied Science and Technology, 8*(6), 3210–3217.

Automatic Language Processing Advisory Committee. (1966). *Language and machines: Computers in translation and linguistics* (Report No. 1416). National Academy of Sciences.

Bar-Hillel, Y. (1960). The present status of automatic translation of languages. *Advances in Computers, 1*, 91–163.

Barcala, F. M., Vilares, J., Alonso, M. A., Grana, J., & Vilares, M. (2002, September). Tokenization and proper noun recognition for information retrieval. In *Proceedings of the 13th International Workshop on Database and Expert Systems Applications* (pp. 246–250). IEEE.

Baş, B. (2011). Söz varlığı ile ilgili çalışmalarda kullanılacak ölçütler. *Türklük Bilimi Araştırmaları*, (29), 27–61.

Baykara, B., & Güngör, T. (2022). Abstractive text summarization and new large-scale datasets for agglutinative languages Turkish and Hungarian. *Language Resources and Evaluation, 56*(3), 973–1007.

Budur, E., Özçelik, R., Güngör, T., & Potts, C. (2020). *Data and representation for Turkish natural language inference* (arXiv:2004.14963). arXiv. https://arxiv.org/abs/2004.14963

Camburn, R. (2013). *A short history of computational linguistics*. https://www.academia.edu/6204763/A_Short_History_of_Computational_Linguistics (Erişim tarihi: 2 Nisan 2025).

Carter, R. (2012). *Vocabulary: Applied linguistic perspectives*. Routledge.

CLARIN.SI. (2025a). *ParlaMint-TR: En sık geçen 2-gramlar* [Ekran görüntüsü]. CLARIN.SI NoSketch Engine. https://www.clarin.si/noske/run.cgi/wordlist?corpname=parlamint21_tr&refs=&wlmaxitems=100&wlsort=f&subcnorm=freq&corpname=parlamint21_tr&reload=&wlattr=word&usengrams=0&ngrams_n=2&ngrams_max_n=2&nest_ngrams=0&wlpat=&wlminfreq=5&wlmaxfreq=0&wlfile=&wlblacklist=&wlnums=frq&wltype=simple

CLARIN.SI. (2025b). *ParlaMint-TR: "li" sözcüğü için sorgu ekranı* [Ekran görüntüsü]. CLARIN.SI NoSketch Engine. https://www.clarin.si/noske/run.cgi/view?corpname=parlamint21_tr;usesubcorp=;q=q%5Bword%3D%3D%22li%22%5D

Çetiner, M., Yıldırım, A., Onay, B., & Öksüz, C. (2021, June). Word sense disambiguation using KeNet. In *2021 29th Signal Processing and Communications Applications Conference (SIU)* (pp. 1–4). IEEE.

Çöltekin, C. (2010). A freely available morphological analyzer for Turkish. In *Proceedings of the 7th International Conference on Language Resources and Evaluation (LREC)* (Vol. 2, pp. 19–28).

Çöltekin, Ç. (2020, May). A corpus of Turkish offensive language on social media. In *Proceedings of the Twelfth language resources and evaluation conference* (pp. 6174-6184).

Delioğlu, G., & Şen, Ü. (2022). Almanlara Türkçe öğretimi için hazırlanmış Güle Güle (A1-A2) ders kitabında sözcük öğretimi. *Uluslararası Türkçe Öğretimi Araştırmaları Dergisi (UTÖAD), 2*(1), 60–80. http://dx.doi.org/10.47834/utoad.39

Derczynski, L., Ritter, A., Clark, S., & Bontcheva, K. (2013, September). Twitter part-of-speech tagging for all: Overcoming sparse and noisy data. In *Proceedings of the international conference recent advances in natural language processing ranlp 2013* (pp. 198-206).

Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019, June). BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational*

Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers) (pp. 4171–4186).

Dolgun, M. Ö., Özdemir, T. G., & Oğuz, D. (2009). Veri madenciliğinde yapısal olmayan verinin analizi: Metin ve web madenciliği. *İstatistikçiler Dergisi: İstatistik ve Aktüerya, 2*(2), 48–58.

Erat, D. (2022). Kant–Frege ve Wittgenstein'da anlam problemi üzerine. *Erciyes Akademi, 36*(2), 593–608.

Evert, S. (2005). *The statistics of word cooccurrences* (Doktora tezi). Stuttgart Üniversitesi, Almanya.

Fayyad, U., Piatetsky-Shapiro, G., & Smyth, P. (1996). From data mining to knowledge discovery in databases. *AI Magazine, 17*(3), 37–37.

Ferrari, G. (2004). State of the art in computational linguistics. In *Linguistics today: Facing a greater challenge* (pp. 163–186).

Firth, J. R. (1957). A synopsis of linguistic theory 1930–55. In *Studies in Linguistic Analysis* (pp. 1–32). The Philological Society.

Forrester, R. (2020). *History of printing: From Gutenberg to the laser printer* (SSRN Working Paper No. 3512249). SSRN. https://doi.org/10.2139/ssrn.3512249

Friedl, J. (2006). *Mastering regular expressions* (3rd ed.). O'Reilly Media, Inc.

Gage, P. (1994). A new algorithm for data compression. *The C Users Journal, 12*(2), 23–38.

Gallé, M. (2019, November). Investigating the effectiveness of BPE: The power of shorter sequences. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)* (pp. 1375–1381).

Garje, G. V., & Kharate, G. K. (2013). Survey of machine translation systems in India. *International Journal on Natural Language Computing, 2*(4), 47–65.

Gimpel, K., Schneider, N., O'connor, B., Das, D., Mills, D. P., Eisenstein, J., ... & Smith, N. A. (2011, June). Part-of-speech tagging for twitter: Annotation, features, and experiments. In

Proceedings of the 49th annual meeting of the Association for Computational Linguistics: Human Language Technologies (pp. 42-47).

Goldman, O., Caciularu, A., Eyal, M., Cao, K., Szpektor, I., & Tsarfaty, R. (2024). *Unpacking tokenization: Evaluating text compression and its correlation with model performance* (arXiv:2403.06265). arXiv. https://arxiv.org/abs/2403.06265

Goyvaerts, J. (2006). *Regular expressions*. Regular-Expressions.info. https://www.regular-expressions.info/

Görgün, M. K., Demirok, G. B., & Kutlu, M. (2023). Türkçe sosyal medya mesajlarından kullanıcıların yaş ve cinsiyetini tahmin etme. *Niğde Ömer Halisdemir Üniversitesi Mühendislik Bilimleri Dergisi, 12*(2), 325–333.

Grefenstette, G., & Tapanainen, P. (1994). What is a word, what is a sentence: Problems of tokenization. In *Proceedings of COMPLEX* (pp. 79–87), Budapeşte.

Grishman, R. (1986). *Computational linguistics: An introduction*. Cambridge University Press.

Guo, J. (1997). Critical tokenization and its properties. *Computational Linguistics, 23*(4), 569–596.

Han, J., Kamber, M., & Pei, J. (2012). *Data mining: Concepts and techniques* (3rd ed.). Elsevier.

Josselson, H. H. (1971). Automatic translation of languages since 1960: A linguist's view. *Advances in Computers, 11*, 1–58.

Haspelmath, M. (2023). Defining the word. *WORD, 69*(3), 283–297. https://doi.org/10.1080/00437956.2023.2237272

Hausser, R. (2013). *Foundations of computational linguistics* (3rd ed.). Springer-Verlag Telos.

Hays, D. G. (1971). Applied computational linguistics. In G. E. Perren & J. L. M. Trim (Eds.), *Applications of linguistics* (pp. 65–84). Cambridge University Press.

Hockett, C. F. (1987). *Refurbishing our foundations*. Benjamins.

Hopton, Z. W., Scherrer, Y., & Samardzic, T. (2025, April). Functional lexicon in subword tokenization. In *Proceedings of the 2025 Conference of the Nations of the Americas*

Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers) (pp. 7839–7853).

Hutchins, J. (2003). ALPAC: The (in)famous report. In *Readings in machine translation* (Vol. 14, pp. 131–135).

Hutchins, J. (2004). Two precursors of machine translation: Artsrouni and Trojanskij. *International Journal of Translation, 16*(1), 11–31.

Hutchins, J., & Lovtskii, E. (2000). Petr Petrovich Troyanskii (1894–1950): A forgotten pioneer of mechanical translation. *Machine Translation, 15*(3), 187–221.

Hutchins, W. J. (1986). *Machine translation: Past, present, future* (p. 66). Ellis Horwood.

Johri, P., Khatri, S. K., Al-Taani, A. T., Sabharwal, M., Suvanov, S., & Kumar, A. (2021). Natural language processing: History, evolution, application, and future work. In A. Abraham, O. Castillo, & D. Virmani (Eds.), *Proceedings of 3rd International Conference on Computing Informatics and Networks* (Lecture Notes in Networks and Systems, Vol. 167). Springer. https://doi.org/10.1007/978-981-15-9712-1_31

Jones, K. S. (1994). Natural language processing: A historical review. In *Current issues in computational linguistics: In honour of Don Walker* (pp. 3–16).

Jurafsky, D., & Martin, J. H. (2025). *Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition with language models* (3rd ed., Draft of January 12, 2025). https://web.stanford.edu/~jurafsky/slp3/

Kaplan, R. M. (2005). A method for tokenizing text. In *Inquiries into words, constraints and contexts* (pp. 55–70).

Kay, M. (2003). In R. Mitkov (Ed.), *The Oxford handbook of computational linguistics*. Oxford University Press.

Ketchen, D. J., & Shook, C. L. (1996). The application of cluster analysis in strategic management research: An analysis and critique. *Strategic Management Journal, 17*(6), 441–458.

King, M. (1984). When is the next ALPAC report due? In *Proceedings of the 10th International Conference on Computational Linguistics and 22nd Annual Meeting of the Association for Computational Linguistics* (pp. 352–353).

Korkmaz, İ. (2019). Makine çevirisinin kısa tarihçesi. *International Journal of Social and Humanities Sciences Research (JSHSR), 6*(32), 155–166.

Kubilay, Y. (2024). *Dede Korkut Hikâyeleri'ndeki eş dizimli yapıların bilgisayarlı dil bilim yöntemlerine göre incelenmesi* (Doktora tezi). Uşak Üniversitesi, Uşak.

Kuhn, M. (1999, June 4). *UTF-8 and Unicode FAQ for Unix/Linux*. https://www.cl.cam.ac.uk/~mgk25/unicode.html

Kurudayıoğlu, M., & Karadağ, Ö. (2005). Kelime hazinesi çalışmaları açısından kelime kavramı üzerine bir değerlendirme. *Gazi Eğitim Fakültesi Dergisi, 25*(2), 293–307.

La Rocca, G., & Boccia Artieri, G. (2022). Research using hashtags: A meta-synthesis. *Frontiers in Sociology, 7*, 1–12. https://doi.org/10.3389/fsoc.2022.1081603

Luz, S. (2022). Computational linguistics and natural language processing. In F. Zanettin & C. Rundle (Eds.), *The Routledge handbook of translation and methodology* (pp. xx–xx). Routledge.

Manaris, B. (1998). Natural language processing: A human-computer interaction perspective. *Advances in Computers*, *47*, 1–66.

Manning, C., & Schütze, H. (1999). *Foundations of statistical natural language processing*. MIT Press.

Melby, A. K. (2019). Future of machine translation: Musings on Weaver's memo. In C. O'Brien & M. Simard (Eds.), *The Routledge handbook of translation and technology* (pp. 419–436). Routledge.

Melchuk, I. A. (1967). Linguistics and automatic translation. *International Social Science Journal*, *19*(1), 41–57.

Mermer, C. (2010). Unsupervised search for the optimal segmentation for statistical machine translation. In *Proceedings of the ACL 2010 Student Research Workshop* (pp. 31–36).

Mielke, S. J., Alyafeai, Z., Salesky, E., Raffel, C., Dey, M., Gallé, M., ... & Tan, S. (2021). Between words and characters: A brief history of open-vocabulary modeling and tokenization in NLP. *arXiv Preprint*, arXiv:2112.10508. https://arxiv.org/abs/2112.10508

Mikheev, A. (2005). Text segmentation. In R. Mitkov (Ed.), *The Oxford handbook of computational linguistics* (pp. 210–218). Oxford University Press.

Mills, J. (1998, August). Lexicon based critical tokenisation: An algorithm. In *Actes EURALEX'98 Proceedings: Papers Submitted to the Eighth EURALEX International Congress on Lexicography in Liège, Belgium* (pp. 213–220).

Moran, J. C. (1971). The development of the printing press. *Journal of the Royal Society of Arts*, *119*(5177), 281–293.

Muslim, E. M. (2007). An introduction to computational linguistics: Advantages & disadvantages. *Journal of the College of Basic Education*, (51), 29–40.

Nwagwu, W. (2022). *The rise and rise of natural language processing research, 1958–2021*.

Özdemir, A., & Yeniterzi, R. (2020, December). Su-NLP at SemEval-2020 Task 12: Offensive language identification in Turkish tweets. In *Proceedings of the Fourteenth Workshop on Semantic Evaluation* (pp. 2171–2176).

Pagel, M. (2017). Q&A: What is human language, when did it evolve and why should we care? *BMC Biology, 15*, 1–6.

Pérusse, D. (1983). Machine translation. *ATA Chronicle, 12*(8), 6–8.

Piatek, S. J. (2021). #sandiegofire—ilk başarılı hashtag. Erişim 22 Mart 2025, https://www.sjpiatek.com/research/sandiegofire-the-first-successful-hashtag/

Pierce, J. R., & Carroll, J. B. (1966). *Language and machines: Computers in translation and linguistics*. Harper & Row.

Pilten-Ufuk, Ş. (2021). Derlem dilbilim ve edebiyat çalışmalarının kesişim noktası: Derlem biçem bilimi. In Ö. Solak & S. Doykun (Eds.), *Disiplinlerarası edebiyat çalışmaları* (ss. 145–171). Paradigma Akademi.

Plath, W. (1967). Multiple path analysis and automatic translation. In *Studies in applied linguistics* (pp. 267–315). North-Holland, Amsterdam.

Ploeger, E., Poelman, W., Høeg-Petersen, A. H., Schlichtkrull, A., de Lhoneux, M., & Bjerva, J. (2024). A principled framework for evaluating on typologically diverse languages. *arXiv preprint*, arXiv:2407.05022. https://arxiv.org/abs/2407.05022

Rust, P., Pfeiffer, J., Vulić, I., Ruder, S., & Gurevych, I. (2020). How good is your tokenizer? On the monolingual performance of multilingual language models. *arXiv preprint*, arXiv:2012.15613. https://arxiv.org/abs/2012.15613

Rychlý, P., & Spalek, S. (2022, December). Utok: The fast rule-based tokenizer. In *Proceedings of RASLAN* (pp. 149–154).

Salton, G. (1989). *A syntactic approach to automatic book indexing*. Cornell University.

Sankur, B. (2004). *İngilizce-Türkçe ansiklopedik bilişim sözlüğü*. Pusula Yayıncılık.

Schmidt, C. W., Reddy, V., Zhang, H., Alameddine, A., Uzan, O., Pinter, Y., & Tanner, C. (2024). Tokenization is more than compression. *arXiv preprint*, arXiv:2402.18376. https://arxiv.org/abs/2402.18376

Schubert, L. K. (2020). Computational linguistics. In E. N. Zalta (Ed.), *The Stanford Encyclopedia of Philosophy* (Summer 2020 ed.). Stanford University. Erişim 23 Mart 2024, https://plato.stanford.edu/archives/sum2020/entries/computational-linguistics/

Schuster, M., & Nakajima, K. (2012, March). Japanese and Korean voice search. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (pp. 5149–5152). IEEE.

Sennrich, R., Haddow, B., & Birch, A. (2015). Neural machine translation of rare words with subword units. arXiv preprint, arXiv:1508.07909. https://arxiv.org/abs/1508.07909

Shannon, C. E. (1948). A mathematical theory of communication. *The Bell System Technical Journal, 27*(3), 379–423.

Steedman, M. (2008). Last words. *Computational Linguistics, 34*(1), 137–144. https://doi.org/10.1162/coli.2008.34.1.137

Stein, D. (2018). Machine translation: Past, present and future. *Language Technologies for a Multilingual Europe, 4*(5).

Syakur, M. A., Khotimah, B. K., Rochman, E. M. S., & Satoto, B. D. (2018, April). Integration k-means clustering method and elbow method for identification of the best customer profile cluster. In *IOP Conference Series: Materials Science and Engineering* (Vol. 336, p. 012017). IOP Publishing.

Tokgöz, M., Turhan, F., Bolucu, N., & Can, B. (2021). Tuning language representation models for classification of Turkish news. In *2021 International Symposium on Electrical, Electronics and Information Engineering* (pp. 402–407).

Toraman, C., Yılmaz, E. H., Şahinuç, F., & Özçelik, O. (2022). Impact of tokenization on language models: An analysis for Turkish. *arXiv preprint*, arXiv:2204.08832. https://arxiv.org/abs/2204.08832

Tsujii, J. (2021). Natural language processing and computational linguistics. *Computational Linguistics, 47*(4), 707–727. https://doi.org/10.1162/coli_a_00420

Tunalı, V., & Bilgin, T. T. (2012, Aralık). Türkçe metinlerin kümelenmesinde farklı kök bulma yöntemlerinin etkisinin araştırılması. ELECO 2012 Elektrik - Elektronik ve Bilgisayar Mühendisliği Sempozyumu'nda sunulan bildiri, Bursa. Erişim: https://www.emo.org.tr/ekler/8995418dd0b5d24_ek.pdf

Tunalı, V. (2011). *Metin madenciliği için iyileştirilmiş bir kümeleme yapısının tasarımı ve uygulaması* (Doktora tezi). Marmara Üniversitesi, İstanbul.

Uzun, N. E. (2004). *Dünya dillerinden örnekleriyle dilbilgisinin temel kavramları*. Türk Dilleri Araştırmaları Dizisi, 39.

Van Rijsbergen, C. (1979, September). Information retrieval: Theory and practice. In *Proceedings of the Joint IBM/University of Newcastle upon Tyne Seminar on Data Base Systems* (Vol. 79, pp. 1–14).

Vijayarani, S., & Janani, R. (2016). Text mining: Open source tokenization tools—An analysis. *Advanced Computational Intelligence: An International Journal (ACII), 3*(1), 37–47.

Weaver, W. (1949). Translation. In G. Locke & S. Booth (Eds.), *Machine translation of languages* (pp. 15–23). MIT Press.

Webster, J. J., & Kit, C. (1992). Tokenization as the initial phase in NLP. In *Proceedings of COLING 1992, Volume 4: The 14th International Conference on Computational Linguistics*.

Wegmann, A., Nguyen, D., & Jurgens, D. (2025). Tokenization is sensitive to language variation. *arXiv preprint*, arXiv:2502.15343. https://arxiv.org/abs/2502.15343

Weiss, S. M., Indurkhya, N., Zhang, T., & Damerau, F. (2010). *Text mining: Predictive methods for analyzing unstructured information*. Springer Science & Business Media.

Witten, I. H., Frank, E., Hall, M. A., & Pal, C. J. (2017). *Data mining: Practical machine learning tools and techniques* (4th ed.). Morgan Kaufmann.

Wong, P. K., & Chan, C. (1996). Chinese word segmentation based on maximum matching and word binding force. In *Proceedings of COLING 1996, Volume 1: The 16th International Conference on Computational Linguistics*.

Wu, J. (2012). *Advances in K-means clustering: A data mining thinking*. Springer Science & Business Media.

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., ... & Dean, J. (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint*, arXiv:1609.08144. https://arxiv.org/abs/1609.08144

Yaşar, P., Şahin, İ., & Adalı, E. (2019, September). Knowledge-based question and answering system for Turkish. In *2019 4th International Conference on Computer Science and Engineering (UBMK)* (pp. 307–312). IEEE.

Yazar, U. T., Kutlu, M., & Bayırlı, İ. K. (2024). *Turkronicles: Diachronic resources for the fast evolving Turkish language*.

Yergeau, F. (2003). UTF-8, a transformation format of ISO 10646. *RFC 3629*. https://www.rfc-editor.org/info/rfc3629

Zeman, D. (2018). *The world of tokens, tags and trees* (Studies in Computational and Theoretical Linguistics, No. 19). Institute of Formal and Applied Linguistics.

Zhang, Y., Wang, H., Yin, G., Wang, T., & Yu, Y. (2017). Social media in GitHub: The role of @-mention in assisting software development. *Science China Information Sciences, 60*, 1–18.

Zouhar, V., Meister, C., Gastaldi, J. L., Du, L., Sachan, M., & Cotterell, R. (2023). Tokenization and the noiseless channel. *arXiv preprint*, arXiv:2306.16842. https://arxiv.org/abs/2306.16842

EK-A: MIT Lisansı

TS Tokenizer MIT Lisansı ile lisanslanmıştır. İlgili lisans https://github.com/tanerim/ts_tokenizer/blob/main/LICENSE adresinde, yazılımın kodları ile binlikte sunulmuştur. Lisans metninin Türkçe çevirisi aşağıda verilmiştir.

MIT Lisansı

Telif Hakkı (c) 2024 Taner Sezer - TS Corpus

İşbu belgeyle, bu yazılımın ve ilişkili dokümantasyon dosyalarının (“Yazılım”) bir kopyasını edinen herkese, Yazılım üzerinde herhangi bir kısıtlama olmaksızın işlem yapma izni ücretsiz olarak verilmiştir. Bu izin, yazılımı kullanma, kopyalama, değiştirme, birleştirme, yayımlama, dağıtma, alt lisans verme ve/veya yazılımın kopyalarını satma haklarını da içermektedir. Ayrıca, yazılımın sağlandığı kişilerin de bu işlemleri yapmasına izin verilir.

Aşağıdaki koşula bağlı olarak:

Yukarıdaki telif hakkı bildirimi ve bu izin bildirimi, Yazılımın tüm kopyalarına veya önemli bölümlerine dahil edilmelidir.

YAZILIM, AÇIK VEYA ZIMNİ HİÇBİR GARANTİ OLMAKSIZIN, “**OLDUĞU GİBİ**” SAĞLANMAKTADIR. BU GARANTİLER TİCARETE ELVERİŞLİLİK, BELİRLİ BİR AMACA UYGUNLUK VE İHLAL ETMEME GARANTİLERİNİ DE KAPSAR AMA BUNLARLA SINIRLI DEĞİLDİR. HİÇBİR DURUMDA, YAZILARIN YAZARLARI VEYA TELİF HAKKI SAHİPLERİ, BİR SÖZLEŞME, HAKSIZ FİİL VEYA DİĞER HERHANGİ BİR SORUMLULUK TEORİSİ KAPSAMINDA; YAZILIMIN KULLANIMI VEYA DİĞER İŞLEMLERİNDEN KAYNAKLANAN ZARARLAR, TALEPLER VEYA DİĞER YÜKÜMLÜLÜKLERDEN SORUMLU TUTULAMAZ.

EK-B: Araştırma Etik Komisyon İzin Muafiyeti Formu[16]

Hacettepe Üniversitesi
Eğitim Bilimleri Enstitüsü
Tez Çalışması/Araştırma Etik Komisyon İzin Muafiyeti Formu

F46

21 / 12 / 2022

Hacettepe Üniversitesi
Eğitim Bilimleri Enstitüsü
Türkçe ve Sosyal Bilimler Eğitimi Ana Bilim Dalı Başkanlığına

Tez/Araştırma Başlığı	KARAKTER DİZİLERİNDEN ANLAMLI BİRİMLERE: DOĞAL DİL İŞLEME YÖNTEMLERİYLE SÖZ VARLIĞI ÇALIŞMALARININ STANDARTLAŞTIRILMASI

Yukarıda başlığı/konusu verilen tez/araştırma çalışmam,

1. İnsan ve hayvan üzerinde deney niteliği taşımamaktadır.
2. Biyolojik materyal (kan, idrar vb. biyolojik sıvılar ve numuneler) kullanılmasını gerektirmemektedir.
3. Beden bütünlüğüne veya ruh sağlığına müdahale içermemektedir.
4. Anket, ölçek (test), mülakat, odak grup çalışması, gözlem, deney, görüşme gibi teknikler kullanılarak katılımcılardan veri toplanmasını gerektiren nitel ya da nicel yaklaşımlarla yürütülen araştırmalar niteliğinde değildir.
5. Diğer kişi ve kurumlardan temin edilen veri kullanımını (kitap, belge vs.) gerektirmektedir. Ancak bu kullanım, diğer kişi ve kurumların izin verdiği ölçüde Kişisel Bilgilerin Korunması Kanuna riayet edilerek gerçekleştirilecektir.

Çalışmada kullanacağım veriler:
(X) Kamusal erişime açık (buraya yazınız): 1- TS Corpus Türkçe Derlem
2- Oscar Açık Erişimli Web Derlemi
3- MaCoCu Açık Erişimli Web Derlemi
( ) Özel izin ve onaya tabi (buraya yazınız): .....
( ) Üretilmiş veri (buraya yazınız): .....
( ) Diğer (buraya yazınız): .....

Yükseköğretim Kurumları Etik Kurullar ve Komisyonlarının Yönergelerini inceledim ve bunlara göre çalışmamın yürütülebilmesi için herhangi bir Etik Komisyondan/Kuruldan izin alınmasına gerek olmadığını; aksi durumda doğabilecek her türlü hukuki sorumluluğu kabul ettiğimi ve yukarıda vermiş olduğum bilgilerin doğru olduğunu beyan ederim.

Gereğini saygılarımla arz ederim.

Taner Sezer

Araştırmacı Bilgileri

Adı Soyadı	Taner Sezer
Öğrenci ise No	N20142390
Ana Bilim Dalı	Türkçe ve Sosyal Bilimler Eğitimi Ana Bilim Dalı
Programı	Türkçe Eğitimi Programı
Statüsü	☐ Yüksek Lisans ☒ Doktora ☐ Bütünleşik Dr. ☐ Diğer

Danışman Görüşü ve Onayı* Etik Kurul izinine tabi olmayan bir çalışmadır.

Prof. Dr. Özay Karadağ

*Tez ve tezden üretilen yayınlarda gerekli

Hacettepe Üniversitesi Eğitim Bilimleri Enstitüsü. Beytepe Yerleşkesi, 06800, Çankaya / ANKARA
Telefon: 0(312) 297 85 72 Belgegeçer: 0(312) 297 85 66 e-Ağ: http://ebe.hacettepe.edu.tr/ e-Posta: ebe@hacettepe.edu.tr

[16] Tez başlığı savunma sınavı sırasında değiştirilmiştir.

EK-C: Etik Beyanı

Hacettepe Üniversitesi Eğitim Bilimleri Enstitüsü, tez yazım kurallarına uygun olarak hazırladığım bu tez çalışmasında,

* tez içindeki bütün bilgi ve belgeleri akademik kurallar çerçevesinde elde ettiğimi,
* görsel, işitsel ve yazılı bütün bilgi ve sonuçları bilimsel ahlak kurallarına uygun olarak sunduğumu,
* başkalarının eserlerinden yararlanılması durumunda ilgili eserlere bilimsel normlara uygun olarak atıfta bulunduğumu,
* atıfta bulunduğum eserlerin bütününü kaynak olarak gösterdiğimi,
* kullanılan verilerde herhangi bir tahrifat yapmadığımı,
* bu tezin herhangi bir bölümünü bu üniversitede veya başka bir üniversitede başka bir tez çalışması olarak sunmadığımı

beyan ederim.

27/06/2025

Taner SEZER

EK-Ç: Doktora Tez Çalışması Orijinallik Raporu

27/06/2025

HACETTEPE ÜNİVERSİTESİ
Eğitim Bilimleri Enstitüsü
Türkçe ve Sosyal Bilimler Eğitimi Ana Bilim Dalı Başkanlığına,

Tez Başlığı : Dizilerden Birimlere: Bilişimsel Dilbilim Çerçevesinde Bir Birimlendirici Tasarımı

Yukarıda başlığı verilen tez çalışmamın tamamı (kapak sayfası, özetler, ana bölümler, kaynakça) aşağıdaki filtreler kullanılarak **Turnitin** adlı intihal programı aracılığı ile kontrol edilmiştir. Kontrol sonucunda aşağıdaki veriler elde edilmiştir:

Rapor Tarihi	Sayfa Sayısı	Karakter Sayısı	Savunma Tarihi	Benzerlik Oranı	Gönderim Numarası
17/07/2025	82	106224	27/06/2025	%3	2716774370

Uygulanan filtreler:

1. Kaynaklar hariç
2. Alıntılar dâhil
3. 5 kelimeden daha az örtüşme içeren metin kısımları hariç

Hacettepe Üniversitesi Eğitim Bilimleri Enstitüsü Tez Çalışması Orijinallik Raporu Alınması ve Kullanılması Uygulama Esaslarını inceledim ve çalışmamın herhangi bir intihal içermediğini; aksinin tespit edileceği muhtemel durumda doğabilecek her türlü hukuki sorumluluğu kabul ettiğimi ve yukarıda vermiş olduğum bilgilerin doğru olduğunu beyan eder, gereğini saygılarımla arz ederim.

Ad Soyadı: Taner Sezer

Öğrenci No.: N20142390

Ana Bilim Dalı: Türkçe ve Sosyal Bilimler Eğitimi Ana Bilim Dalı

Programı: Türkçe Eğitimi

Statüsü: ☐ Y.Lisans ☒ Doktora ☐ Bütünleşik Dr.

İmza

DANIŞMAN ONAYI

UYGUNDUR.

Prof. Dr. Özay KARADAĞ

EK-D: Dissertation Originality Report

27/06/2025

HACETTEPE UNIVERSITY

Graduate School of Educational Sciences

To The Department of Turkish and Social Sciences Education

Thesis Title: Strings to Tokens: Designing a Tokenizer Within a Computational Linguistics Framework

The whole thesis that includes the *title page, introduction, main chapters, conclusions and bibliography section* is checked by using **Turnitin** plagiarism detection software take into the consideration requested filtering options. According to the originality report obtained data are as below.

Time Submitted	Page Count	Character Count	Date of Thesis Defense	Similarity Index	Submission ID
17/07/2025	82	106224	27/06/2025	3%	2716774370

Filtering options applied:

1. Bibliography excluded
2. Quotes included
3. Match size up to 5 words excluded

I declare that I have carefully read Hacettepe University Graduate School of Educational Sciences Guidelines for Obtaining and Using Thesis Originality Reports; that according to the maximum similarity index values specified in the Guidelines, my thesis does not include any form of plagiarism; that in any future detection of possible infringement of the regulations I accept all legal responsibility; and that all the information I have provided is correct to the best of my knowledge.

I respectfully submit this for approval.

Name Lastname: Taner Sezer

Student No.: N20142390 Signature

Department: Turkish and Social Sciences Education

Program: Turkish Teaching

Status: ☐ Masters ☒ Ph.D. ☐ Integrated Ph.D.

ADVISOR APPROVAL

APPROVED

Prof. Dr. Özay KARADAĞ

EK-E: Yayımlama ve Fikrî Mülkiyet Hakları Beyanı

Enstitü tarafından onaylanan lisansüstü tezimin/raporumun tamamını veya herhangi bir kısmını, basılı (kâğıt) ve elektronik formatta arşivleme ve aşağıda verilen koşullarla kullanıma açma iznini Hacettepe Üniversitesine verdiğimi bildiririm. Bu izinle Üniversiteye verilen kullanım hakları dışındaki tüm fikri mülkiyet haklarım bende kalacak, tezimin tamamının ya da bir bölümünün gelecekteki çalışmalarda (makale, kitap, lisans ve patent vb.) kullanım hakları bana ait olacaktır.

Tezin kendi orijinal çalışmam olduğunu, başkalarının haklarını ihlal etmediğimi ve tezimin tek yetkili sahibi olduğumu beyan ve taahhüt ederim. Tezimde yer alan telif hakkı bulunan ve sahiplerinden yazılı izin alınarak kullanılması zorunlu metinlerin yazılı izin alınarak kullandığımı ve istenildiğinde suretlerini Üniversiteye teslim etmeyi taahhüt ederim.

Yükseköğretim Kurulu tarafından yayınlanan "**Lisansüstü Tezlerin Elektronik Ortamda Toplanması, Düzenlenmesi ve Erişime Açılmasına ilişkin Yönerge**" kapsamında tezim aşağıda belirtilen koşullar haricince YÖK Ulusal Tez Merkezi / H.Ü. Kütüphaneleri Açık Erişim Sisteminde erişime açılır.

Enstitü/ Fakülte yönetim kurulu kararı ile tezimin erişime açılması mezuniyet tarihinden itibaren 2 yıl ertelenmiştir. (1)

- Enstitü/Fakülte yönetim kurulunun gerekçeli kararı ile tezimin erişime açılması mezuniyet tarihimden itibaren … ay ertelenmiştir. (2)
- Tezimle ilgili gizlilik kararı verilmiştir. (3)

27 / 06 / 2025

(imza)

Taner SEZER

"Lisansüstü Tezlerin Elektronik Ortamda Toplanması, Düzenlenmesi ve Erişime Açılmasına İlişkin Yönerge"

(1) Madde 6. 1. Lisansüstü tezle ilgili patent başvurusu yapılması veya patent alma sürecinin devam etmesi durumunda, tez danışmanının önerisi ve enstitü anabilim dalının uygun görüşü Üzerine enstitü veya fakülte yönetim kurulu iki yıl süre ile tezin erişime açılmasının ertelenmesine karar verebilir.

(2) Madde 6. 2. Yeni teknik, materyal ve metotların kullanıldığı, henüz makaleye dönüşmemiş veya patent gibi yöntemlerle korunmamış ve internetten paylaşılması durumunda 3. şahıslara veya kurumlara haksız kazanç; imkânı oluşturabilecek bilgi ve bulguları içeren tezler hakkında tez danışmanın önerisi ve enstitü anabilim dalının uygun görüşü üzerine enstitü veya fakülte yönetim kurulunun gerekçeli kararı ile altı ayı aşmamak üzere tezin erişime açılması engellenebilir.

(3) Madde 7. 1. Ulusal çıkarları veya güvenliği ilgilendiren, emniyet, istihbarat, savunma ve güvenlik, sağlık vb. konulara ilişkin lisansüstü tezlerle ilgili gizlilik kararı, tezin yapıldığı kurum tarafından verilir*. Kurum ve kuruluşlarla yapılan işbirliği protokolü çerçevesinde hazırlanan lisansüstü tezlere ilişkin gizlilik kararı ise, ilgili kurum ve kuruluşun önerisi ile enstitü veya fakültenin uygun görüşü Üzerine üniversite yönetim kurulu tarafından verilir. Gizlilik kararı verilen tezler Yükseköğretim Kuruluna bildirilir.

Madde 7.2. Gizlilik kararı verilen tezler gizlilik süresince enstitü veya fakülte tarafından gizlilik kuralları çerçevesinde muhafaza edilir, gizlilik kararının kaldırılması halinde Tez Otomasyon Sistemine yüklenir

*Tez danışmanının önerisi ve enstitü anabilim dalının uygun görüşü üzerine enstitü veya fakülte yönetim kurulu tarafından karar verilir.