TÜRKİYE CUMHURİYETİ
ANKARA ÜNİVERSİTESİ
SAĞLIK BİLİMLERİ ENSTİTÜSÜ

KÖŞE YAZILARINDA YAZARLIK ANALİZİNİN ADLİ BİLİŞİME KATKISI

Cemre KOÇYİĞİT

DİSİPLİNLERARASI ADLİ BİLİMLER ANABİLİM DALI
ADLİ BİLİŞİM
YÜKSEK LİSANS TEZİ

DANIŞMAN

Dr.Öğr.Üyesi Bülent TUĞRUL

ANKARA
2022

TÜRKİYE CUMHURİYETİ
ANKARA ÜNİVERSİTESİ
SAĞLIK BİLİMLERİ ENSTİTÜSÜ

KÖŞE YAZILARINDA YAZARLIK ANALİZİNİN ADLİ BİLİŞİME KATKISI

Cemre KOÇYİĞİT

DİSİPLİNLERARASI ADLİ BİLİMLER ANABİLİM DALI
ADLİ BİLİŞİM
YÜKSEK LİSANS TEZİ

DANIŞMAN
Dr.Öğr.Üyesi Bülent TUĞRUL

ANKARA
2022

ETİK BEYAN

Ankara Üniversitesi

Sağlık Bilimleri Enstitüsü Müdürlüğü'ne,

Yüksek Lisans tezi olarak hazırlayıp sunduğum "Köşe Yazılarında Yazarlık Analizinin Adli Bilişime Katkısı" başlıklı tez; bilimsel ahlak ve değerlere uygun olarak tarafımdan yazılmıştır. Tezimin fikir/hipotezi tümüyle tez danışmanım ve bana aittir. Tezde yer alan deneysel çalışma/araştırma tarafımdan yapılmış olup, tüm cümleler, yorumlar bana aittir.

Yukarıda belirtilen hususların doğruluğunu beyan ederim.

Öğrencinin Adı Soyadı: Cemre KOÇYİĞİT

Tarih: 06.01.2022

İmza:

KABUL VE ONAY

Ankara Üniversitesi Sağlık Bilimleri Enstitüsü
Disiplinlerarası Adli Bilimler Anabilim Dalında
Cemre KOÇYİĞİT tarafından hazırlanan
"Köşe Yazılarında Yazarlık Analizinin Adli Bilişime Katkısı" adlı tez çalışması
aşağıdaki jüri tarafından YÜKSEK LİSANS TEZİ olarak OY BİRLİĞİ/OY ÇOKLUĞU ile kabul edilmiştir.

Tez Savunma Tarihi:
24/01/2022

Prof.Dr. Süleyman TOSUN
Hacettepe Üniversitesi
Jüri Başkanı

Dr.Öğr.Üyesi Yılmaz AR
Ankara Üniversitesi
Üye

Dr.Öğr.Üyesi Bülent TUĞRUL
Ankara Üniversitesi
Mühendislik Fakültesi
(Tez Danışmanı)

Tez hakkında alınan jüri kararı, Ankara Üniversitesi Sağlık Bilimleri Enstitüsü Yönetim Kurulu tarafından onaylanmıştır.

Prof. Dr. Fügen AKTAN
Sağlık Bilimleri Enstitüsü Müdürü

İÇİNDEKİLER

ÖNSÖZ

Teknolojinin gelişimi ile birlikte bilişim suçlarının arttığı göz ardı edilemez bir gerçektir. Özelleştirildiğinde ise intihal bu suçlar arasında en yaygın bilişim suçlarından birisidir. Bu çalışma vasıtasıyla bilişim sistemleri kullanılarak en basitinden intihali tespit etme ve engelleme amaçlanacaktır. Bu hedefe ulaşmada Doğal Dil İşleme ve makine öğrenme algoritmaları sonucuna göre araştırmanın başarılı olup olmadığı hususunda gözlemler yapılmıştır.

Cumhuriyet tarihi boyunca öncelikle bir kadın olarak ilimle bu günlere gelmemizi; dolayısıyla bu çalışmayı yapabilmemi sağlayan ilk olarak başöğretmen Mustafa Kemal ATATÜRK'e minnetlerimi sunar, bu tez çalışmasını yürütmede yardımlarını ve katkılarını hiçbir zaman esirgemeyen danışman hocam, Dr.Öğr.Üyesi Bülent TUĞRUL'a ve her koşulda beni destekleyen, yanımda olan sevgili aile fertlerime ve arkadaşlarıma teşekkür ederim.

SİMGELER VE KISALTMALAR

AI	Artificial İntelligence (Yapay Zekâ)
API	Application Programming Interface (Uygulama Programlama Ara yüzü)
CPU	Central Processing Unit (Merkezi İşlem Birimi)
SRI	Scientific Research Institute (Bilimsel Araştırma Kurumu)
DDİ	Doğal Dil İşleme
EDM	Euclidean Distance Matrix (Öklid Uzaklık Matrisi)
GPU	Graphics Processing Unit (Grafik İşlem Birimi)
KNN	K-Nearest Neighbours (En yakın K Komşu)
LSTM	Long Short Term Memory (Uzun Kısa Süreli Bellek)
MDS	Multidimensional Scaling (Çok boyutlu Ölçekleme)
ML	Machine Learning (Makine Öğrenme)
MLP	Multi-Layer Perceptron (Çok Katmanlı Algılayıcı)
NB	Naive Bayes
NER	Named Entity Recognition (Adlandırılmış Varlık Tanıma)
NLP	Natural Language Processing
NLTK	Natural Language Toolkit (Doğal Dil Araç Takımı)
RNN	Recurrent Neural Networks (Tekrarlayan Sinir Ağları)
SVC	Support Vector Classifier (Destek Vektör Sınıflandırıcısı)
SVM	Suppor Vector Machines (Destek Vektör Makineleri)
TF-IDF	Term Frequency- Inverse Document Frequency (Terim Frekansı- Ters Metin Frekansı)
VSC	Visual Studio Code

ŞEKİLLER

ÇİZELGELER

1. GİRİŞ

1.1. Problem ve Önlem

Bilişim sistemleri teknolojinin gelişmesiyle hayatımızın her alanında var olmuştur ve olmaktadır. Yeni suç türleri kendini göstermekte, suçlular teknolojiye ulaşarak yasa dışı faaliyetlerine devam etmektedirler. İncelediğinde özellikle bilişim sistemlerini kullanarak suç işlemede büyük bir artış bulunmuş; Türkiye'de de dünyada olduğu gibi, bilişim sistemlerine karşı ve bilişim sistemleri kullanarak suç işleme, kolluk ve yargı mercilerinin hemen her gün yüz yüze geldiği suç türleri içinde yer almaktadır. Bilişim suçları üç başlıkla gruplandırılır;

1. Hedef Olarak Bilişim Sistemleri: Yetki olmadan erişim sağlama, servis dışı bırakma, verilere zarar verme, vb.,
2. Bilişim Bağlantılı Suçlar: Kredi kartı ve internet bankacılığı dolandırıcılığı, telif hakları ihlali vb.,
3. Bilişim Vasıtalı Normal Suçlar: Uyuşturucu trafiği, kara para aklama, hakaret, terör propagandası, vb.dir (Ekizer, 2014).

Günümüz bilgi teknolojisi çağında, herhangi bir bilgiye erişim o kadar zor olmamaktadır. Kullanıcı, bilgiye dünyanın her yerinden 7/24 ulaşabilir. Ancak bilginin kullanımıyla birlikte kötüye kullanımı da arttığından, bazı yazarlar diğer yazarların eserlerini küçük değişikliklerle kendi adlarıyla yayınlamaktadır (Soni, 2018). Böylece dijital teknoloji ile metin üzerinde intihal (plagiarism), artan bir sorun haline gelmektedir (Björnson, 2019).

Aslen intihal, birçok kişi tarafından diğer insanların araştırmalarını kopyalamak ya da özgün fikirlerini ödünç almak olarak düşünülür. Ancak “kopyalama” ve “ödünç alma” gibi ifadelerle bu suçun ciddiyeti kamufle edilmiş olabilir: Merriam-Webster çevrimiçi sözlüğünde, “intihal” şu şekilde tanımlanmıştır; (başkasının fikirlerini veya sözlerini) çalmak ve kendisine aitmiş gibi sunmak, (başka bir

kimsenin eserini) atıfta bulunmadan kullanmak, edebi hırsızlık yapmak, var olan bir kaynaktan türetilen bir fikri ya da ürünü yeni ve özgünmüş gibi yansıtmak, vb. anlamlarına gelir. Diğer bir söylemle, intihal dolandırıcılık içeren bir fiildir. Sadece başka bir kimsenin ürününü çalmayı değil, aynı zamanda bu konuda sonradan yalan söylemeyi kapsar (Tupa, 2017).

"Kansu (1994) intihali, bilimsel yanıltma (scientific misconduct) adı altında üç madde ile gruplamaktadır. Bilimsel yanıltmanın üç unsuru aşağıda verilmiştir;

1. Bilimsel korsanlık (piracy): Araştırmacının izni olmadan diğer çalışmalardaki verilerin alınması.
2. İntihal (plagiarism): Başkalarının fikirlerini, yazılarını ve eserlerini çalmak ve bunları uygun bir atıf olmaksızın kendisininmiş gibi sunmak.
3. Saptırma (fabrication, desk-research, dry-lab): Verilerin çarpıtılması ya da var olmayan bilgilerin/verilerin oluşturulmasıdır (s.72).

Bu sınıflandırmadan da anlaşıldığı üzere birçok bilimsel yanıltma ya da etik olmayan eylem örneği bulunmaktadır. İntihali diğerlerinden farklı kılan temel özelliği atfın bilimsel intihalin merkezinde konumlandırılmasıdır. Atfın asıl işlevi, atıf yapılan belge ile atıf yapan belge arasında bir bağlantı kurmak olarak tanımlanır (Smith, 1981, s.84, aktaran Al ve Coştur, 2007, s.144). Ayrıca kaynağa göre kaynak göstermeden yapılan veya kaynak göstererek yapılan intihaller de vardır.

İntihal, internette bulunan erişilebilir bilgilerin artmasıyla farklı bir boyutta görülmektedir. İntihali artıran bir diğer önemli faktör ise internetteki bilgilerin "kamu malı" olduğu yargısıdır (Uçak & Birinci, 2008). İntihali saptamak için birçok sistem geliştirilmiştir ancak bu sistemlerin tamamı gelişen ve değişen teknoloji ile güvenilir nitelikte olmamaktadır (Björnson, 2019).

İntihali belirlemede kullanılan programlarda benzerlik yüzdesinin tamamı mutlaka intihal olmayabilir. Çıkan raporun doğru bir şekilde incelenmesi gerekir çünkü benzer metodolojiyi veya yaygın objektif terimleri açıklayan metinlerin birbirine benzemesi muhtemeldir. Ayrıca, değerlendirmeyi yapan kişinin alıntıları ve kaynak dizinini hariç tutmak için manuel müdahaleye ihtiyacı vardır. Bu nedenle, bir yazı için "kabul edilebilir benzerlik yüzdesi" üzerinde hakemlik yapmak, eleştirel analiz gerektirir (Kadam, 2018). Dolayısıyla metinler üzerinde yazarlık analizi yöntemi, diğer yöntemlerin yanı sıra metin üzerindeki intihali belirlemede yardımcı olabilir.

1.2. Araştırmanın Amacı ve Önemi

Bu çalışmada köşe yazıları üzerinden yazarlık analizi yapılacağından intihali belirlemede önemli bir yer tutabilir. Köşe yazılarını online ortamdan elde edip, Doğal Dil İşleme adımlarından geçirdikten sonra Makine Öğrenme Algoritmalarına başvurulmuştur. Elde edilen sonuçlara göre köşe yazılarının yazdığı yazara ait olup olunmadığı tartışılmıştır. Dolayısıyla bu çalışmanın başta intihal olmak üzere bilişim suçlarını önlemede bir etken olacağı düşünülmüş ve Adli Bilişim bilimine katkı sağlanması hedeflenmiştir.

Adli yazarlık analizleri Adli Bilimler içerisinde önemli bir yer tutmaktadır. Yazarlık analizi içinde olan yazar tanıma alanına ise metin ve olası yazar sayısı birden fazla olduğunda başvurulur. Metinler vasiyetnameler, tehdit mektupları, intihar notları vb. içerebilir, ancak son on yılda e-postaları, metin mesajlarını ve en son tweet'leri, Facebook ve WhatsApp mesajlarını içeren analizlerde büyük bir artış yaşanmaktadır. Bu analizler sadece plagiarismi önlemede değil aynı zamanda siber suçları önlemede de önemli bir konuma sahiptir (Coulthard ve ark., 2009).

1.3. Sınırlılıklar

Tez çalışması 6 yazarın toplamda 900'ü aşkın köşe yazısı üzerinden yapılmıştır. Bu konu itibariyle çalışma sınırlı tutulmuştur. Ayrıca bu 6 köşe yazarının yazdığı konular da sayıdan dolayı sınırlı tutulmuştur. Ancak yine de çalışma sonunda elde edilen verilerle genel geçer bir açıklama yapılmaya çalışılmıştır.

1.4. Literatür Özeti

1.4.1. Bilişim

Edinburgh Üniversitesi'nin (2016) yaptığı açıklamaya göre, bilişim, doğal ve tasarlanmış sayısal sistemlerin yapısı, davranışı ve etkileşimlerinin incelenmesidir. Sayısal, bilişsel ve sosyal yönleri içinde barındırır. Ana fikir, ister hesaplama ister iletişim, ister organizmalar ister yapay nesneler tarafından olsun, bilginin dönüştürülmesidir. Bilişimin birçok yönü vardır ve yapay zekâ, bilişsel bilim ve bilgisayar bilimi gibi bir dizi mevcut akademik disiplini kapsar. Her biri kendi doğal alanı olarak bilişimin bir parçasını alır: geniş anlamda, bilişsel bilim, doğal sistemlerin incelenmesiyle ilgilidir; bilgisayar bilimi, hesaplama analizi ve bilgisayar sistemlerinin tasarımı ile ilgilidir; yapay zekâ, doğada bulunanları taklit eden sistemler tasarlayarak bağlayıcı bir rol oynar. Bilişim ayrıca matematik, elektronik, biyoloji, dilbilim ve psikoloji gibi diğer disiplinleri etkiler ve onlar tarafından da etkilenir.

1.4.2. Adli Bilişim

Dr. H. B. Wolfe'ye göre, adli bilişim, bir mahkemede tutarlı ve anlamlı bir formatta sunulabilecek, programlama ekipmanından ve çeşitli depolama cihazlarından ve dijital medyadan kanıt toplamak için sistemli bir dizi teknik ve prosedürü içinde barındırır. Adli bilişimi daha ayrıntılı olarak tanımlarsak, dijital

kanıtların toplanması, analizi ve mahkemeye sunulması prosedürüdür. Adli bilişimin kapsamı yalnızca bir suçu soruşturmakla sınırlı değildir; ayrıca veri kurtarma, sistem günlüğü (log) izleme, (kullanımdan kaldırılmış veya hasar görmüş cihazlardan) veri toplama ve uyumluluk ihtiyaçlarını karşılama durumlarıyla da iç içedir (Shakeel, 2003).

Genel olarak adli bilişim, herhangi bir suçun veya yasaklı aktivitenin olup olmadığına karar vermek için bilişim sistemleri ve depolama birimleri üzerinde yapılan bütün çalışmaları içermektedir (Kılıç, 2019).

1.4.2.1. Adli Bilişimin Tarihi

"İnsanlar bilgisayar merkezine girdiklerinde ahlaki değerlerini kapıda bıraktılar."

(Donn B. Parker, 1968)

Bilgisayarlar ilk olarak 1940'ların ortalarında ortaya çıktı ve bu teknolojinin hızlı gelişimini kısa sürede çeşitli bilişim suçları izledi. 1960'ların ortalarında, SRI International'dan Donn Parker, bilişim suçları ve etik olmayan bilgisayarla işlenmiş faaliyetler hakkında araştırma yapmaya başladı ve 'insanların bilgisayar merkezine girdiklerinde ahlaki değerlerini kapıda bıraktıklarını' fark etti (Bynum, 2001). Parker'ın çalışmaları önümüzdeki yirmi yıl boyunca devam etti ve bilgisayar etiği tarihinde bir dönüm noktası olarak kabul edildi. Bilişim suçuyla ilgili ilk kovuşturma vakası 1966'da ABD'nin Teksas eyaletinde kaydedildi (Dierks, 1993) ve beş yıl hapis cezasıyla sonuçlandı. 1970'lerde ve 1980'lerde kişisel bilgisayarlar hem evde hem de işyerinde yaygınlaştı; daha sonra emniyet teşkilatı yeni bir suç sınıfının ortaya çıktığını fark etti: bu da bilişim suçuydu (Overill, 1998). Tüm suçlar gibi, bu yeni sınıf da başarılı kovuşturmalar için güvenilir kanıtlar gerektiriyordu. Böylece bilişim suçlarını çözmeyi, belgelemeyi ve kovuşturmayı mümkün kılmayı amaçlayan adli bilişim disiplini ortaya çıktı (Bern ve ark., 2008).

Wikipedia'ya göre adli bilişim, 1980'lerin ortalarından beri ceza hukukunda kanıt olarak kullanılmıştır, bazı dikkate değer örnekler şunlardır:

1. BTK Katili: Dennis Rader, on altı yıl boyunca meydana gelen bir dizi seri cinayetten hüküm giymişti. Bu sürenin sonuna doğru Rader bir disket üzerinde polise mektuplar gönderdi. Belgelerdeki meta veriler, "Christ Lutheran Kilisesi"nde "Dennis" adlı bir yazarla ilgiliydi; bu kanıt Rader'in tutuklanmasına yardımcı oldu.
2. Joseph Edward Duncan: Duncan'ın bilgisayarından alınan bir elektronik çizelge, onun suçlarını planladığını gösteren kanıtlar içeriyordu. Savcılar bunu kasıtlı olarak göstermek ve ölüm cezasını güvence altına almak için kullandılar.
3. Sharon Lopatka: Lopatka'nın bilgisayarındaki yüzlerce e-posta, araştırmacıları katili Robert Glass'a yönlendirmişti.
4. Corcoran Grup: Bu dava, dava açıldığında veya makul bir şekilde beklendiğinde tarafların dijital kanıtları koruma vazifeleri olduğu durumunda kesinleşmişti. Sabit diskler, sanıklara ait olması gereken ilgili e-postaları bulamayan bir adli bilişim uzmanı tarafından analiz edildi. Uzman, sabit disklerde silme kanıtı bulamamış olsa da sanıkların e-postaları kasıtlı olarak imha ettikleri, maddi gerçekleri davacılara ve mahkemeye ifşa etmede başarısız oldukları ve yanlış bilgi verdikleri hususlarında kanıtlar ortaya çıktı.
5. Conrad Murray: Merhum Michael Jackson'ın doktoru Dr. Conrad Murray, bilgisayarındaki dijital kanıtlarla kısmen mahkûm edilmişti. Bu kanıt, öldürücü miktarlarda propofol gösteren tıbbi belgeleri kapsamaktaydı.

1990'lara gelindiğinde, teknolojik olarak gelişmiş her ülkedeki kolluk kuvvetleri bilişim suçlarının farkındaydı, soruşturma ve kovuşturması için bir sisteme sahipti. Birçok bilimsel araştırma merkezi de kuruldu ve yazılım endüstrisi, bilişim suçlarının araştırılmasına yardımcı olmak için çeşitli özel araçlar sunmaya başladı (Noblett ve ark., 2000) (Bem ve ark., 2008).

1.4.3. Adli Dilbilim

Olsson'a (2008) göre adli dilbilim ise, dilbilimin yasal konulara uygulanmasıdır. En geniş anlamıyla adli dilbilim dil, suç ve hukuk arasındaki ara yüzdür. Genel olarak yaşam için dil kullanımının ve özellikle de hukukun merkezi olduğu düşünüldüğünde, parmak izi tanımlama ve ayakkabı izi analizi gibi diğer disiplinlere göre adli dilbilimin arenaya göreceli olarak yeni olması yargı süreçlerinde köklü bir varlığa sahip olduğundan belki biraz şaşırtıcıdır. Dilbilimsel yöntemlerin hukuki sorunlara uygulanması, adli dilbilimin bir bilimin uygulaması olduğu anlamlardan yalnızca bir tanesidir, çünkü bir araştırmadaki dil örneklerinin analizine çeşitli dilbilim teorileri uygulanabilir. Bu nedenle adli dilbilimci, dil ve hafıza çalışmaları, konuşma analizi, söylem analizi, dilbilgisi teorisi, bilişsel dilbilim, konuşma yasası teorisi vb. gibi çok çeşitli alanlarda yürütülen araştırmalardan gözlemlerini alıntılayabilir. Dilbilimsel alanların geniş yelpazesine olan bu güvenin nedeni şöyle açıklanabilir: Dilbilimcinin analiz için aldığı veriler, ortalama bir kişinin dili nasıl hatırladığı, konuşmaların nasıl kurulduğu, konuşmacıların/yazarların konuşma/metin boyunca nasıl davrandıkları ya da bir mahkemeye ifade veya cümle yapılarının hangi yönlerinin yansıtılması gerektiği hakkında bilgi sunabilir.

Adli dilbilim araştırmaları diğer birçok alanla bağlantılı olabilir. Örneğin, adli dilbilim alanı bilişim alanında çeşitli çalışmalara öncü olmuştur. Daha özelleştirmek gerekirse, siber suç araştırmalarında adli dilbilim büyük bir önem arz etmektedir. Çünkü günümüzde neredeyse her suç dijital iz bırakabileceğinden, internette ceza soruşturmaları giderek önem kazanmıştır (Tropina ve ark., 2017).

Bu çalışma da birçok yazarlık analizi çalışmaları gibi disiplinler arası adli bilişim ve adli dilbilim alanlarını birleştirir niteliktedir. Yazarlık analizi yapılırken bu iki bilime başvurulmuştur.

1.4.4. Yazarlık Analizi

Adli dilbilimin doğuşuyla birlikte, herkesin dili benzersiz bir şekilde kullandığı ve dil kullanımındaki farklılıklar parmak izi kadar kolay gözlenebileceği anlamına gelen "adli parmak izi" kavramını ortaya çıkardı (Coulthard, 2004; Olsson, 2008). Yazarlık analizinde potansiyel olarak geçerli ayırt edici belirteçler sunan, bilinçaltı dil alışkanlıklarıdır. Bu da yazarların dil kullanımlarında farklılıklar göstermiştir.

İnsanlar sözlü veya yazılı dil kullandıklarında, belirli sözlük-dilbilgisi (lexico-grammatical) yapılarını diğerlerinden daha sık seçme ve belirli bir şekilde birleştirme alışkanlığı vardır. Bu tür alışkanlıkların dil kullanımının yazara özgü özellikleri olduğu söylenir ve genellikle sorgulanan bir metnin yazarını tanımlamak için adli dil analizinde uygulanır (Coulthard, 2004; Coulthard & Johnson, 2010).

1.4.4.1. "Idiosyncratic Alışkanlıklar" nedir?

Son yıllarda bir metnin yazarlığını doğrulamak veya tartışmak için kullanılan yöntemlerden bazıları stilometri, korpus analizi (Coulthard, 2004) ve dilbilimsel analizi (Coulthard & Johnson 2010) içerir. Stylometrics, öncelikle, dilin kelime veya cümle uzunluğu, sözlüksel zenginlik, tempo, hapax legomena (metinde yalnızca bir kez kullanılan kelime öğeleri) ve benzeri gibi teknik yönlerinin ölçülmesi ile ilgilidir (McMenamin, 2002; Coulthard & Johnson, 2010).

Öte yandan, korpus analizi, bu tür dizelerin diğer yazarlar tarafından kullanılma olasılığını belirlemek için, belirli bir kelime öğesinin veya kelime dizgisinin büyük bir toplulukta tartışmalı metinden sıklığını aramayı içerir. Son olarak, dilbilimsel analiz sözdizimi, dilbilgisi, hatalar vb. gibi kendine özgü dil kullanımının (idiosyncratic alışkanlıklar) çeşitli yönlerinin analiz edilmesini içerebilir (Coulthard & Johnson, 2010) (Tomić, 2019).

Yazarlık analizi yapılırken, her biri farklı bir amaca hizmet eden üç farklı alt araştırma dalı bulunmaktadır. Bu üç alt dal, yazarlığı tanıma, yazarlığı sınıflandırma ve benzerliği tespit etmekten oluşmaktadır.

1. Yazarlık tanıma (bkz. yazarlık özniteliği): Belirli bir kişi tarafından o kişiden gelen diğer yazıları inceleyerek bir yazı parçasının üretilme olasılığını belirlemek için kullanılır.
2. Yazarlık sınıflandırma: Bir yazarın cinsiyet, yaş, eğitim düzeyi veya kültür geçmişi gibi kişisel niteliklerini, o yazara ait mevcut yazıları kullanarak belirlemek için kullanılan bir tekniktir.
3. Benzerlik tespiti: Farklı yazı parçalarını karşılaştırır ve aynı kişi tarafından üretilip üretilmediğini belirler. Bu tekniğin en popüler kullanımı intihal tespitidir.

1.4.4.2. Stilistik özellikler (Stylometric features)

Yazarlık tanıma araştırmasını kolaylaştırabilecek yazı stili özellikleri dört farklı kategoriye ayrılmaktadır; bunlar sözcüksel (lexical) özellikler, yapısal (structural) özellikler, içeriğe özgü (content specific) özellikler ve sözdizimsel (syntactic) özelliklerdir.

1. Sözcüksel özellikler, sözcük ve karakter tabanlı analizlerle desteklenir. Kelime bazlı analiz ile bir kişinin özelliği, yazılarındaki kelime zenginliği, cümle başına ortalama kelime sayısı veya bir metindeki toplam kelime sayısı gibi diğer ölçümlerle birleştirilerek anlatılabilir. Karakter bazlı analiz ile toplam karakter sayısı, cümle başına düşen ortalama karakter sayısı, kelime başına düşen karakter veya tek tek harflerin kullanım sıklığı gibi bazı özellikler belirli bir kişinin yazma alışkanlıklarını ortaya çıkarmaya yardımcı olur.

2. Yapısal özellikler, metnin düzenini ve planını analiz etmeye odaklanır. Yazının uzunluğu diğer analiz özellikleri için yeterli olmadığında, yazılar arasındaki farklılıkları ayırt etmek için yapısal özellikler önemli bir rol oynar.
3. Sözdizimsel özellikler, cümleleri oluşturmak için kullanılan yazma modellerini araştırma ve analiz etmeyle ilgilenir. Sözdizimsel özellikler, cümleleri düzenlemede insanların farklı alışkanlıklarından türetildikleri için sözdizimsel özelliklerin ayırt edici gücünden yola çıkarak farklı kişileri ayırt etmek için Türkçe'deki noktalama ve işlev (function) kelimeleri gibi yaygın kelimelerin kullanımını analiz etmeye odaklanır.
4. İçeriğe özgü özellikler ise, belirli bir konu alanıyla ilgili olan sözcükleri ifade eder (Dinh, 2014).

Bu çalışmada yazarlık tanıma ve benzerlik tespiti çalışma alanlarından yararlanılmıştır. Çalışma, köşe yazılarında yazarlık analizi içermekte olup ilk adım olan stilometrik özellikler üzerinden yazarlık tanıma yapılmıştır. Daha sonra benzerlik tespit çalışmasına göre yazılarının belirtilen yazarlara ait olup olmadığı elde edilen verilere göre tartışılacaktır. Yazarlık tanıması yapılırken veri seti Doğal Dil İşleme (Natural Language Processing) uygulama alanından geçirilmiş, benzerlik tespiti yapılırken de makine öğrenme (machine learning) metotları ve algoritmalarına başvurulmuştur.

1.4.5. Doğal Dil İşleme

Doğal dil, insanlar tarafından birbirleriyle iletişim kurmak için kullanılan dili ifade eder. Bu iletişim sözlü veya yazılı olabilir. Örneğin, yüz yüze görüşmeler, tweetler, bloglar, e-postalar, web siteleri, SMS mesajları, hepsi doğal dil içerir.

Ancak, insanlardan farklı olarak bilgisayarlar doğal dili kolayca anlayamazlar. Doğal dili bilgisayarların anlayabileceği bir biçime çevirmek için gelişmiş teknikler ve yöntemler gereklidir. Doğal Dil İşleme (NLP), bu hedefe ulaşılmasına yardımcı olan uygulama alanıdır (Robinson, 2017).

Wikipedia'ya göre NLP, dilbilim, bilgisayar bilimi ve yapay zekanın bir alt alanıdır. NLP, bilgisayarlar ve insan dili arasındaki etkileşimlerin yanı sıra, özellikle bilgisayarların büyük miktarda doğal dil verisini işlemesi ve analiz etmesi için nasıl programlanacağı ile ilgilenir. Belgelerdeki dilin bağlamsal nüansları da dahil, sonuç, belgelerin içeriklerini "anlayabilen" bir bilgisayardır. Teknoloji, belgeleri kendi başına sınıflandırmanın ve düzenlemenin yanı sıra, belgelerde yer alan bilgileri ve iç görüleri daha sonra doğru bir şekilde çıkarabilir.

NLP, doğal dillerin kurallı yapısını analiz ederek anlamayı ya da yeniden üretmeyi amaçlar. Bu analizin insanlara sağlayacağı kolaylıklar, soru-cevap makineleri, yazılı belgelerin otomatik tercümesi, bilgi sağlama, otomatik konuşma ve komut anlama, konuşma üretme, oluşturma ve sentezi, otomatik metin özetleme gibi birçok konuyla toparlanabilir. Mesela, birer imla düzeltme aracı tüm kelime işlem yazılımlarında bulunur. Bu araçlar aslında yazılı metni analiz eden ve dil kurallarını kontrol eden doğal dil işleme yazılımlarıdır. NLP, bilgisayarların insan dilini algılaması gereken her yerde kullanılabilir, olası uygulama alanları şu şekildedir:

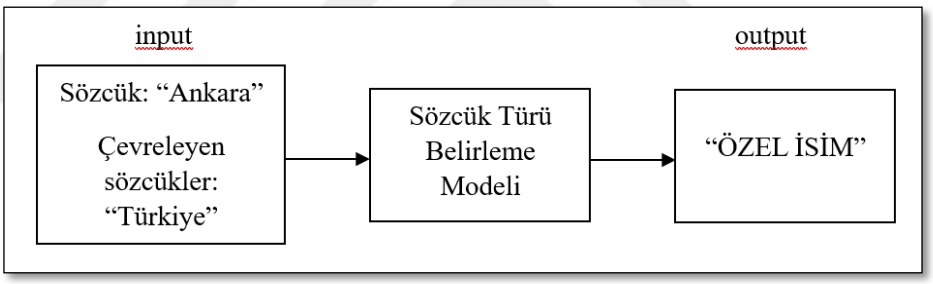
1. İnternet ortamında git gide artan dokümanların değerlendirilmesinde,
2. Uluslararası çalışan şirketlerin müşteri profilini belirlemede,
3. Elektronik ticarette,
4. Savunma ve istihbarat alanlarında (Güvenlik ve suçlu teşhisinde),
5. Yabancı dil öğretiminde,
6. Makine çevirisinde,
7. Elektronik sözlüklerde,
8. İmla hatalarının otomatik düzeltilmesinde,
9. Film ve sinema alanında,
10. Mobil telefonların konuşma algılama sistemlerinde,
11. Otomatik özet çıkarmada,
12. Bilgi aramada,
13. Görme engellilerin bilgisayar kullanmalarındadır (Tarcan & Çakar, 2008).

NLP, bilgisayarların insan dillerini anlamasını ve işlemesini sağlamaya odaklanan yapay zekanın (AI) alt alanı olduğunu gösteren aşamalardan oluşmaktadır. Python kullanarak ham metinden bilgi çıkarabilecek programların nasıl yazılacağının adımları sistematik bir şekildedir. Adım adım NLP süreçleri şu şekilde incelenmektedir:

1. Cümlelere Bölme: Verilen bir paragraf öncelikle cümlelere bölünür.
2. Kelime ve Simgelere Ayırma: Bölünen cümleler, kelimeler ve simgelere ayrılır.

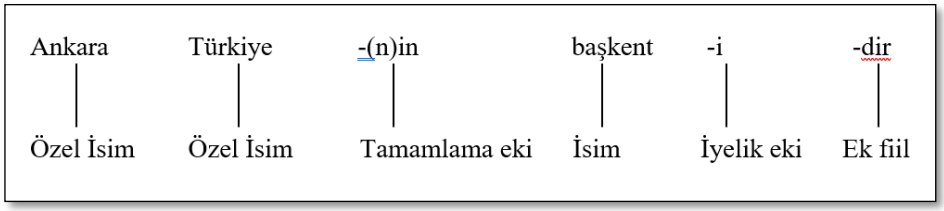
   Ankara, Türkiye'nin başkentidir.

   → "Ankara", ", ", "Türkiye'nin", "başkentidir", ". "
3. Her Simge/Sözcük İçin Tür Belirleme: Söz konusu sözcük isim, fiil, sıfat vb. olabilir. Böylece cümledeki her kelimenin rolü öğrenilir. (Şekil 1.1.)



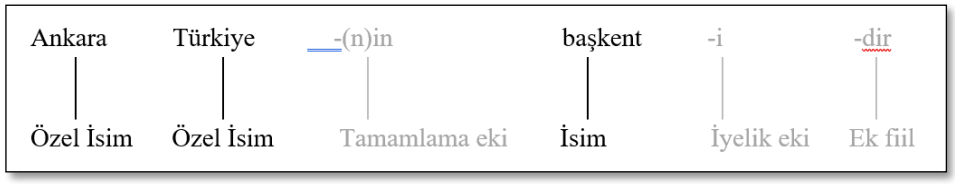
Şekil 1.1. Türkçe dilinde her simge/sözcük için tür belirleme örneği.

4. Metin Kök Çözümlemesi: Her sözcüğün kökü (temel formu) çıkarılır (Geitgey, 2018). (Şekil 1.2.)



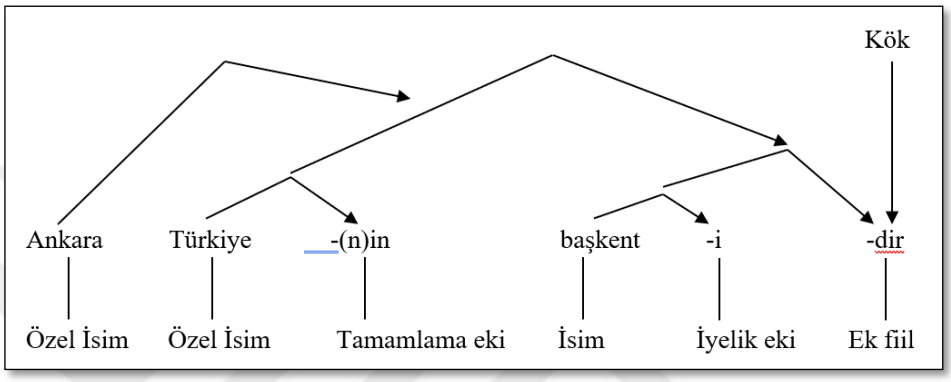
Şekil 1.2. Türkçe dilinde metin kök çözümleme örneği (Kuzucular, 2014).

5. Durduran İfadeleri Belirleme: Metin üzerinde istatistik yaparken, bu ifadeler çok fazla engel çıkarır, çünkü diğer kelimelere göre çok daha sık görünürler. (Şekil 1.3.)



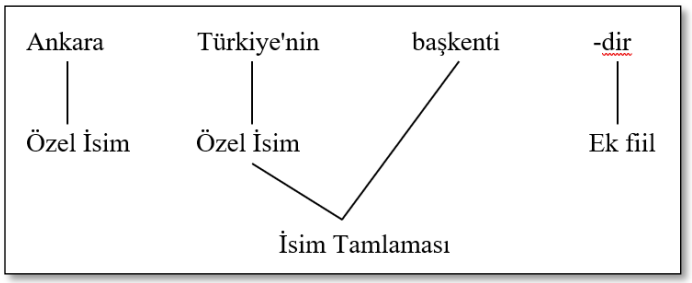
Şekil 1.3. Türkçe dilinde durduran ifadeleri belirleme örneği.

6. Bağımlı Ayrıştırma: Amaç, cümledeki her kelimeye tek bir ana kelime atayan bir ağaç oluşturmaktır. Ağacın kökü cümlenin ana fiili olmalıdır. (Şekil 1.4.)



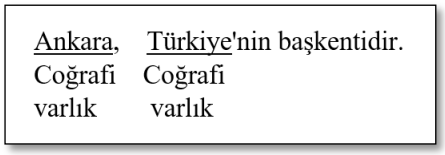
Şekil 1.4. Türkçe dilinde bağımlı ayrıştırma örneği.

6b. İsim Öbeklerini Bulma: Aynı varlıktan bahseden kelimeleri otomatik olarak gruplandırmak için bağımlı ayrıştırma ağacındaki bilgileri kullanabiliriz. (Şekil 1.5.)



Şekil 1.5. Türkçe dilinde isim öbeklerini bulma örneği.

7. Adlandırılmış Varlık Tanıma (NER): NER'in amacı, bu isimleri temsil ettikleri gerçek dünya kavramlarıyla tespit etmek ve etiketlemektir. (Şekil 1.6.)



Şekil 1.6. Türkçe dilinde adlandırılmış varlık teşhisi örneği.

Tipik bir NER sisteminin etiketleyebileceği bazı nesne türleri şunlardır:

- İnsan isimleri,
- Şirket isimleri,
- Coğrafi konumlar (hem fiziksel hem de politik),
- Ürün isimleri,
- Tarihler ve saatler,
- Para miktarları,
- Etkinlik isimleridir.

8. Eşgönderge Çözümlemesi: Bu adımın amacı cümlelerdeki zamirleri veya o ismin yerine kullanılan öbekleri takip ederek aynı eşlemeyi bulmaktır (Geitgey, 2018).
   ➔ Ankara, Türkiye'nin başkentidir. Ankara'nın başkent ilan edilmesinin ardından (13 Ekim 1923) şehir hızla gelişmiş ve Türkiye'nin ikinci en kalabalık ili olmuştur (Vikipedi).

1.4.5.1. Metin Madenciliğinde Python İşlevselliği

Yukarıda da görüldüğü üzere Python'da Kodlama yaparken NLP Boru Hattı genel haliyle bu şekildedir (Geitgey, 2018). Python metin madenciliği paketi, birçok kullanışlı işlevler içerir. Temelde istatistiksel metin madenciliğine odaklanır ve bir doküman koleksiyonundan bir özet doküman oluşturmaya yardımcı olur. Bu matris daha sonra başka analizler için istatistiksel pakette incelenebilir. Paket, koleksiyon bulmada, kelime arasındaki düzen mesafesini hesaplamada ve uzun bir dokümanı daha küçük parçalara dönüştürmede faydalı araçlar sağlamaktadır. Model, Google,

Twitter, Wikipedia API, DOM ayrıştırıcı ve NLP veri madenciliği için araçlar sağlayan Python programlama dili için web madenciliği modülüdür (Zende ve ark., 2016). Bu çalışmada da Python'da metin madenciliği kodla çağrılan Python kütüphaneleri aracılığıyla NLP adımlarıyla yapılmıştır.

1.4.5.2. Tez Çalışmasında Kullanılan Python Kütüphaneleri

Bu tez çalışmasında metin madenciliği yapılırken ve veri makine öğrenme (ML) algoritmalara sokulurken birçok Python kütüphanesinden faydalanılmıştır. Bu kütüphaneler kısaca şunlardır:

1. Urllib ve BeautifulSoup Kütüphaneleri: Urllib standart bir Python kütüphanesidir ve web üzerinden veri istemi oluşturmak, çerezleri işlemek ve hatta başlıklar ve kullanıcı araç gibi meta verileri değiştirmek için işlevler içerir. Örneğin, 'urlopen' işlevi, bir ağ üzerinden uzaktaki bir nesneyi açmak ve onu okumak için kullanılır. Oldukça genel bir kütüphane olduğu için HTML dosyalarını, görüntü dosyalarını veya diğer dosya akışlarını kolaylıkla okuyabilir. BeautifulSoup ise, anlamsız olanı anlamlandırmaya çalışır; düzensiz HTML'yi düzelterek ve XML yapılarını temsil eden, kolay geçişli Python nesneleri sunarak dağınık web'i biçimlendirmeye ve düzenlemeye yardımcı olur. (O'Reilly Media, 2015). Dolayısıyla Python modülü 'urllib.request', benzer kaynak konumlayıcıları (URL'leri) çıkarmaya yardım ederken, Python modülü Beautifulsoup, Python'daki HTML ve XML dosyalarından veri çıkarmaya yardımcı olur (Wingate, 2020).
2. NLTK (Natural Language Toolkit) Kütüphanesi: Doğal Dil Araç Takımı (NLTK) paketi, sözcük türü seçme, bölümleme ve sınıflandırma gibi çok sayıda NLP yöntemlerini içerir. NLTK, python programlama dili için istatistiksel doğal dil işlemeye yönelik bir kütüphane ve program paketidir. Sınıflandırma, bölme ve ayrıştırma için metin işleme kütüphaneleri için ara yüz kullanımını kolaylaştırır. NLTK, Windows, MAC OS X ve Linux yazılımlarında bulunmaktadır (Zende ve ark., 2016).

3. Stanza Kütüphanesi: Stanza web sitesine göre, Stanza, birçok dilin dilbilimsel analizi için doğru ve verimli araçlardan oluşan bir koleksiyondur. Ham metinden sözdizimsel analiz ve varlık tanımaya kadar Stanza, en gelişmiş NLP modellerini seçilen dillere getirir. Araç takımı, Evrensel Bağımlılıklar (Universal Dependencies) biçimini kullanarak 70'ten fazla dil arasında paralel olacak şekilde tasarlanmıştır. Stanza, simgeleştirme (tokenization), çok sözcüklü simge (multi-word token) genişletme, kök çözümleme (lemmatization), sözcük türü (part-of-speech) ve biçembilimsel (morphological) özellikler etiketleme, bağımlılık ayrıştırma ve NER dahil olmak üzere tam sinir ağı boru hattı sağlamaktadır.
4. TensorFlow Kütüphanesi: TensorFlow, sayısal hesaplamada veri akışı grafiklerini kullanan açık kaynaklı bir yazılım kütüphanesidir (Unruh, 2017). Makine öğrenimi için uygulamalar oluşturur ve derin sinir ağlarının (deep neural networks) eğitimi ve çıkarımına odaklanır. Örneğin, Google TensorFlow'u arama motorunda arama yaparken bir kelime yazıldığında başka öneriler sunarak var olan aramayı genişletir. TensorFlow, verilerin bir grafikte nasıl hareket ettiğini tanımlamada veri akışı grafikleri ve yapıların oluşturulmasına yardım eder, akış şeması oluşturur. TensorFlow yapısı 3 bölümde çalışır; veri önişlemi, modeli oluşturma ve modeli eğitip, tahmin etmedir (Johnson, 2021).
5. Keras Kütüphanesi: Keras ise, derin öğrenme modelleri oluşturmak için TensorFlow, Theano vb. gibi popüler derin öğrenme kütüphanelerinin üzerine inşa edilmiş en güçlü ve kullanımı kolay Python kütüphanelerinden biridir. Keras, üst düzey sinir ağı uygulama programlama ara yüzünü (API) daha kolay ve daha performanslı hale getirmek için çeşitli optimizasyon tekniklerinden yararlanır. Keras, birtakım özellikleri destekler. Bunlar:
    - Tutarlı, basit ve genişletilebilir API,
    - Minimal yapı (herhangi bir ayrıntı olmadan sonuca ulaşmak kolaydır),
    - Çoklu platform ve arka uç,
    - Hem merkezi işlem birimi (CPU) hem de grafik işlem birimi (GPU) üzerinde çalışan kullanıcı dostu bir sistem,

- Hesaplamanın yüksek düzeyde ölçeklenebilirliğidir (Tutorials Point (I) Pvt. Ltd. - Keras, 2019, s.:1).

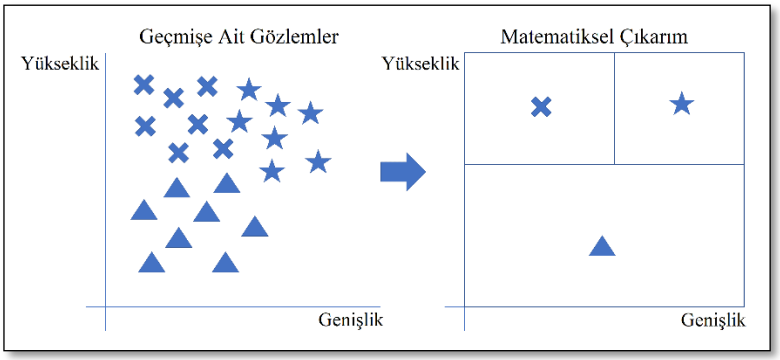
6. NumPy ve Pandas Kütüphaneleri: 'NumPy', açılım olarak "Numerical Python (Sayısal Python)" anlamına gelir. Dizinler ve matrisler üzerinde hızlı matematiksel hesaplama sağlayan açık kaynaklı bir Python modülüdür. Dizinler ve matrisler, Makine Öğrenimi ekosisteminin önemli bir parçası olduğu için NumPy; Scikit-learn, Pandas, Matplotlib, TensorFlow vb. gibi Makine Öğrenimi modülleriyle birlikte Python Makine Öğrenimi Ekosistemini tamamlar. NumPy, tamsayılar, dize veya (homojen) karakterler içeren aynı tip elementlere sahip bir çizelgedir. NumPy'de boyutlara eksen (axes) adı verilir. Eksen sayısına da dizi (rank) denir. NumPy'ye benzer şekilde Pandas, veri biliminde en yaygın kullanılan python kütüphanelerinden biridir. Yüksek performans, kullanımı kolay yapılar ve veri analiz araçları sağlar. Çok boyutlu dizinler için nesneler sağlayan NumPy kütüphanesinden farklı olarak Pandas, veri çerçevesi adlı bellek içi iki boyutlu (2D) çizelge nesnesi sağlar. Sütun adları ve satır etiketleri içeren bir hesap çizelgesi gibidir. Pandas'da yaygın olarak kullanılan bazı veri yapıları şunlardır:
    - Seri nesneler: Bir boyutlu (1D) dizin – hesap çizelgesindeki bir sütuna benzer,
    - Veri çerçevesi nesneleri: İki boyutlu (2D) tablo – hesap çizelgesine benzer,
    - Panel nesneleri: Veri çerçevesi Sözlüğü – MS Excel'deki sayfaya benzer (Pratik, 2017).
7. Scikit-Learn (Sklearn) Kütüphanesi: Python'da makine öğrenimi için en kullanışlı ve güçlü kütüphanedir. Python'da bir tutarlık ara yüzü aracılığıyla sınıflandırma, regresyon, kümeleme ve boyutluluk azaltma içeren makine öğrenimi ve istatistiksel modelleme için verimli araçları ayırır (Tutorials Point (I) Pvt. Ltd. - Scikit-Learn, 2019, s.:1).
8. Matplotlib Kütüphanesi: Matplotlib 2 ve 3 boyutlu grafikler çizen Python kütüphanesidir. Matplotlib kullanılarak, çizgi grafiği, histogram, çubuk grafiği, pasta grafiği, dağılım grafikleri, alan çizimleri, hata çizelgeleri, güç izgesi vb. çizilebilir. Veri analizi model geliştirmenin bir parçası olduğundan

sadece sayısal veriye bakılarak doğru sonuçlar elde edilmeyebilir. O yüzden veriyi kullanarak grafiklerin çizilmesi bir sonraki işlemler için karar vermede en iyi yoldur. Veri görselleştirme, veri temizleme işleminden sonra yapılan bir işlemdir. Matplotlib, NumPy gibi bazı üçüncü taraf Python kütüphanelerine bağlıdır (Indian AI Production, 2019).

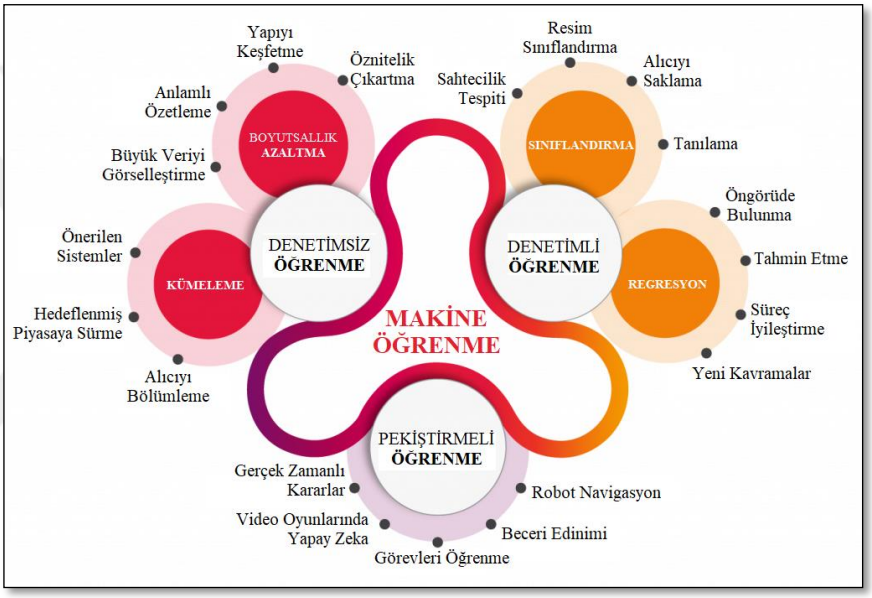
1.4.6. Makine Öğrenmesi (ML)

Wikipedia'ya göre makine öğrenimi (ML), deneyim ve veri kullanımı yoluyla otomatik olarak iyileştirilebilen bilgisayar algoritmalarının incelenmesidir. ML, yapay zekanın (AI) bir bölümü olarak görülmektedir. Makine öğrenimi algoritmaları, açık bir şekilde programlanma yapmadan tahminlerde bulunmak ya da kararlar almak için "eğitim verileri" olarak bilinen örnek verilere uygun bir model yaratır. Makine öğrenimi algoritmaları, gerekli olan görevleri yerine getirmek için geleneksel algoritmalar geliştirmenin zor veya imkânsız olduğu tıp bilimi, e-posta filtreleme, konuşma tanıma ve bilgisayar görüntüsü gibi çok çeşitli uygulamalarda kullanılmaktadır. Makine öğreniminin bir alt kümesi, bilgisayarları kullanarak tahminler yapmaya odaklanan hesaplama istatistikleriyle yakından ilişkilidir; ancak bütün makine öğrenimi istatistiksel öğrenme olarak sayılmamaktadır.

Makine öğreniminin genel fikri, herhangi bir konudaki geçmiş verilerden eğilimleri öğrenmek için bir model elde etmek ve bu eğilimleri gelecekte karşılaştırılabilir veriler üzerinde yeniden üretebilmektir (Şekil 1.7.). Aşağıdaki grafik, geçmiş verilere uyan bir makine öğrenimi modelinin görsel bir temsilidir. Solda üç değişkenli (yükseklik, genişlik ve şekil) orijinal gözlemler vardır. Şekiller, yıldızlar, çarpılar ve üçgenlerden oluşmaktadır. Şekiller grafiğin farklı alanlarına yerleştirilmiştir. Sağ tarafta, bu orijinal gözlemlerin bir karar kuralına nasıl çevrildiği gösterilmektedir. Yeni bir gözlemde, hangi kareye düştüğünü belirlemek için genişliği ve yüksekliği bilinmelidir. Düştüğü kare, hangi şekle sahip olma olasılığının en yüksek olduğunu tanımlar. Temel makine öğrenimi süreci bu şekilde özetlenmektedir (Korstanje, 2021).



Şekil 1.7. Makine öğrenimi modelinin görsel bir örneği.



Şekil 1.8. Makine Öğrenme sınıflandırması (Sakal, 2020).

Sakal'a (2020) göre "Makine öğrenme algoritmaları makine öğrenme bulmacasının sadece bir parçasıdır. Algoritma seçimine (manuel veya otomatik) ek olarak, optimize ediciler, veri temizleme, özellik seçimi, özellik normalizasyonu ve (isteğe bağlı olarak) hiperparametre ayarlarıyla ilgilenmek gerekir" (Şekil 1.8). Dört genel makine öğrenimi yöntemi vardır: Bunlar; (1) denetimli, (2) denetimsiz, (3) yarı denetimli ve (4) pekiştirmeli öğrenmelerdir.

1. Denetimli Öğrenme (Supervised Learning): Bu öğrenmede hedef, *etiketlenen* eğitim verilerinden bir fonksiyon veya eşleştirme çıkarmaktır. Bu çıkarım regresyon ve sınıflandırma ile gerçekleşir.

2. Denetimsiz Öğrenme (Unsupervised Learning): Bu öğrenmede sahip olunan tek şey *etiketlenmemiş* verilerdir. Amaç, bu verilerde gizli bir yapı bulmaktır çünkü eğitim verileri burada yer almazlar. Sayısız veri toplama cihazıyla veri, benzeri görülmemiş bir oranda toplanır ve sınıflandırılmaz.
3. Yarı Denetimli Öğrenme: Bu tür öğrenmede, veriler sınıflandırılmış ve sınıflandırılmamış verilerin bir karışımıdır. Etiketli ve etiketsiz verilerin bu kombinasyonu, verilerin sınıflandırılmasında uygun bir model geliştirmek için kullanılır. Çoğu durumda, etiketlenmiş veriler az, etiketlenmemiş veriler bol miktarda bulunur. Yarı denetimli sınıflandırmanın hedefi, yalnızca etiketlenmiş veriler kullanılarak geliştirilen modeldense, gelecekteki test verilerinin sınıflarını daha iyi tahmin edecek bir model öğrenmektir (Mohammed ve ark., 2016, s.:7-10).
4. Pekiştirmeli Öğrenme: Wikipedia'ya göre bu öğrenmede, bir bilgisayar programı, belirli bir amacı gerçekleştirmesi gereken (araç sürmek veya bir rakibe karşı oyun oynamak gibi) dinamik bir ortamla etkileşime girer. Sorun alanında gezinirken, programa, en üst düzeye çıkarmaya çalıştığı, ödüllere benzer bir geri bildirim sağlanır.

Bu öğrenmeler sadece görüntü ve ses verilerinde değil aynı zamanda metin verilerinde de kullanır. Burada veri, *metin madenciliği* adı altında toplanır. Metin verilerini elde etmek için birçok mecraya başvurulabilir. Sosyal medya, metin verilerinin üretimini benzeri görülmemiş bir düzeyde görebileceğimiz yerdir. Metin madenciliği, birçok uygulamada yardımcı olur. Bunlar;

- Ticari istihbarat,
- Ulusal güvenlik,
- Fen bilimleri,
- Duygu sınıflandırmasıyla ilgili olanlar,
- Otomatik reklam yerleştirme,
- Haber makalelerinin otomatik sınıflandırılması,
- Sosyal medya takibi,
- İstenmeyen posta filtresidir (Mohammed ve ark., 2016, s.:23-24).

Bu çalışmada metin madenciliği yapılırken denetimli öğrenme algoritmalarından yararlanılmıştır. Bu algoritmalar yine Python kodlamaları ile gerçekleştirilmiştir.

1.4.6.1. Derin Öğrenme

Derin öğrenme (Deep learning), bir hiyerarşi içinde düzenlenmiş birçok seviyeden oluşan yapay sinir ağı kullanarak makine öğrenimi sürecini gerçekleştirir. Ağ, hiyerarşide ilk seviyede basit bir şey öğrenir ve ardından bu bilgiyi bir sonraki seviyeye gönderir. Bir sonraki seviye bu basit bilgiyi alır, onu biraz daha karmaşık bir şeyle birleştirir ve üçüncü seviyeye iletir. Bu süreç, hiyerarşideki her bir seviye önceki seviyeden aldığı girdiden daha karmaşık bir aşama inşa edene dek devam eder.

Derin öğrenme ağları, bilgi keşfi, bilgi uygulaması ve bilgiye dayalı tahmin için büyük verilere başarıyla uygulanabilir. Başka bir deyişle, derin öğrenme, işlemeye uygun sonuçlar üretmek için güçlü bir motor olabilir (Murnane, 2016).

1.4.6.2. Tez Çalışmasında ML için Kullanılan Metotlar

Makine Öğrenimi algoritmaları metinleri değil sayıları anlar. Bu nedenle, algoritma için anlaşılır hale getirmek için tüm "metin" sütunları "sayısal" sütunlara dönüştürülmelidir. Etiketlerin veya kategorik/metin değerlerinin sayılara veya sayısal değerlere dönüştürülmesi ile gerçekleştirilir (Gurav, 2020). Bu tez çalışmasında veri setini ML algoritmalarına hazırlayan bu tür bazı metotlar kullanılmıştır. Bu metotlar aşağıda sıralanmaktadır:

1. CountVectorizer: Makineler, karakter ve kelimeleri anlayamadığından metin verilerinin üstesinden gelebilmek için sayısal değerlere dönüştürülmesi gerekmektedir. CountVectorizer metni sayısal veriye dönüştürmede bir metot olarak kullanılmaktadır. Örnek vermek gerekişe Şekil 1.9.'da metin CountVectorizer ile seyrek bir matrise (sparse matrix) dönüştürülür; metinde

6 benzersiz sözcük bulunmaktadır, o yüzden matriste her bir benzersiz sözcüğü temsil eden 6 farklı sütun vardır. Satırda ise sözcük sayısı temsil edilir. 'Benim' kelimesi iki kez tekrarlandığı için bu kelime 2, diğer kelimeler ise 1 kez tekrarlandığı için 1 değeri almıştır. CountVectorizer, metin verileri için doğrudan makine öğrenimi ve metin sınıflandırma gibi derin öğrenme modellerinde kullanılmasını kolaylaştırır.

metin = ["Merhaba benim adım Ali, bu benim bilgisayarım"]

	merhaba	benim	adım	ali	bu	bilgisayarım
0	1	2	1	1	1	1

Şekil 1.9. Bir girdiyi sayısal değerlere dönüştürme örneği.

Eğer girdi 1'den fazla ise (Şekil 1.10.); her bir girdi önişlemden geçirilir, simgeleştirilir (tokenization) ve seyrek bir matris olarak sunulur. Varsayılan olarak, CountVectorizer metni küçük harfe dönüştürür ve sözcük düzeyinde simgeleştirmeyi kullanır. Kod yazarken matrisi görselleştirmek için Pandas kütüphanesi, vektörleştirmeyi gerçekleştirmek için bir Sklearn kütüphanesi olan 'sklearn.feature_extraction.text' kullanılır (Jain, 2021). Bu iki cümleye bakıldığında çıktı bölmesinde her bir cümle için var olan kelimeye 1, olmayan kelimeye ise 0 değeri atanır ve böylece cümlelerin ve kelimelerin vektörleri elde edilmiş olur.

metin = ["Merhaba benim adım Ali", "bu benim bilgisayarım"]

	adım	ali	benim	bilgisayarım	bu	merhaba
0	1	1	1	0	0	1
1	0	0	1	1	1	0

Şekil 1.10. İki girdiyi sayısal değerlere dönüştürme örneği.

TfidfVectorizer: CountVectorizer, özellik oluşturmak için kelime sayısını dikkate alır, bu nedenle cümle yapısını ve sırasını dikkate almaz ve semantik anlam olarak eksiktir. Ayrıca, büyük bir seyrek matris ile sonuçlanır. Bu durumda TF-IDF kullanılmalıdır (Kaul, 2021). TF-IDF (Term Frequency - Inverse Document Frequency), "Terim Frekansı - Ters Metin Frekansı" anlamına gelir. TF-IDF, bir dokümandaki kelimenin önemini ölçen sayısal

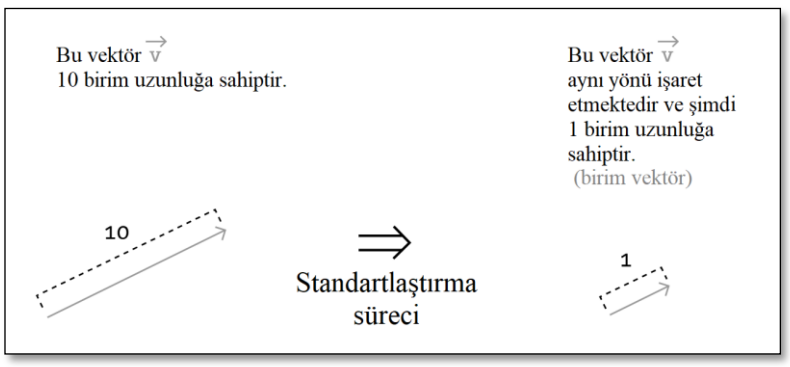
bir istatistiktir. Terim Frekansı, bir kelimenin metinde görüntülenme sayısı iken, Ters Metin Frekansı, metinde nadir veya yaygın olan kelimeyi ölçer (Kanani, 2019). TF-IDF, bağlamı okumak yerine anlamaya çalışır ve Sklearn kütüphanesini kullanır. TF için metinde belli bir terim fazla kullanılmışsa o terim önemlidir. IDF ise ilgili terimlere odaklanır, metinde durduran ifadeler fazla kullanılmışsa bile görmezden gelerek çalışır. Şekil 1.11.'deki denklemleri kullanırlar (Kaul, 2021). TfidfVectorizer, işlenmemiş dokümanların koleksiyonunu bir TF-IDF özellik matrisine dönüştürür. Metin analizi, makine öğrenimi algoritmalarında önemli bir uygulama alanıdır. Metin öncelikle sayısal özellik vektörlerine dönüştürülmelidir çünkü bilgisayar sayısal verileri işler (Tracy, 2021).

$$TF(m) = \frac{\text{metindeki terim m'nin görünme sayısı}}{\text{metindeki toplam terimler}}$$

$$IDF(m) = \log \frac{\text{metinlerin toplam sayısı}}{\text{terim m bulunan metinlerin sayısı}}$$

Şekil 1.11. TF-IDF Vectorizer kullanılan denklemler (Kaul, 2021).

2. Vektörleri Standartlaştırma (Normalization): Bir vektörün büyüklüğünü hesaplamak sadece başlangıçtır. Vektörlere bakıldığında, standart bir vektörün uzunluğunun bir olduğunu varsayarsak, bir vektörü standart hale getirmek, herhangi bir uzunlukta bir vektörü almak ve onu aynı yöne bakacak şekilde, uzunluğunu bir birime değiştirerek birim vektör (unit vector) olarak adlandırılan olguya dönüştürmektir (Shiffman, 2012) (Şekil 1.12.).



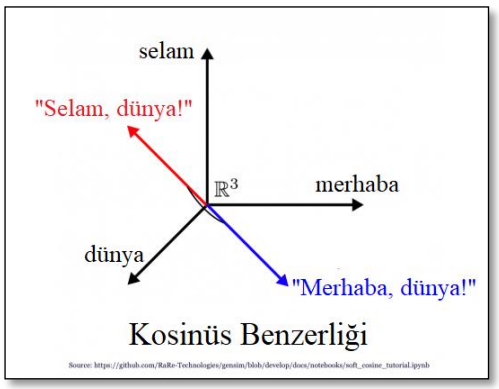
Şekil 1.12. Vektörleri standartlaştırma örneği.

3. Öklid Uzaklık Matrisi (Euclidean Distance Matrix (EDM)): Bu matris, satırların "kaynak", sütunların "hedef" varlıklar olduğu ve üzerinde (Öklid tarzında) bir mesafe hesaplanabilen tabloyu temsil eder. Örneğin, A noktasından B noktasına iki boyuttaki mesafeyi hesaplarken, A ve B vektörünün iki boyut ekseni üzerindeki izdüşümleri hesaplanmaktadır. Formül şu şekildedir:

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Ancak, bir tarafında binlerce satır ve yüzlerce "özellik" (n-boyutlu uzay) içeren veri tablolarını düşünüldüğünde, bu basit formül bile kolayca çok daha karmaşık hale gelebilir (Grianti, 2020).

4. Kosinüs Benzerliği (Cosine Similarity): EDM'de metnin boyutundan dolayı iki benzer metin birbirine uzak görülebilir. Böylece Kosinüs Benzerliği metriği ile boyutuna bakmaksızın dokümanların ne kadar benzediğine karar verilebilir. Matematiksel olarak, çok boyutlu bir uzayda yansıtılan iki vektör arasındaki açının kosinüsünü ölçer. Bu iki vektör arasındaki açısal mesafe (angular distance) ne kadar düşükse benzerlik o kadar yüksektir. Ayrıca semantik anlam dikkate alınır (Şekil 1.13.) ve buna göre anlam bakımından benzer kelimeler de benzer olarak ele alınmalıdır (Prabhakaran, 2018).



Şekil 1.13. Kosinüs Benzerliği Örneği.

5. MDS (Multidimensional Scaling), "çok boyutlu ölçekleme" olarak adlandırılır. Scikit Learn'e göre MDS, mesafelerin orijinal yüksek boyutlu uzaydaki uzaklıklara iyi bir şekilde uyan verilerin düşük boyutlu bir gösterimini araştırır. MDS, benzerlik ve farklılık verilerini analiz etmede

kullanılan bir tekniktir. Benzerlik veya farklılık verilerini geometrik uzaylarda mesafeler olarak modellemeye çalışır. MDS metodunun metrik ve metrik olmayan şeklinde iki türü vardır. Metrik MDS'de, girdi olan benzerlik matrisi bir ölçümden doğar (ve böylece üçgen eşitsizliğine uyar). İki nokta çıktısı arasındaki mesafeler daha sonra benzerlik veya farklılık verilerine mümkün olduğunca yakın olacak şekilde ayarlanır. Metrik olmayan versiyonda ise, algoritmalar mesafelerin sırasını korumaya çalışacak ve bu nedenle gömülü uzaydaki mesafeler ile benzerlikler/farklılıklar arasında tekdüze bir ilişki arayacaktır.

1.4.6.3. Tez Çalışmasında Kullanılan ML Algoritmaları

Bu tez çalışmasında köşe yazıları üzerinden yazar tanıma yapılabilmesi için ML algoritmalarına başvurulmuştur. Çıkan sonuçlar ve analizi 3. Bölüm'de anlatılmaktadır. Faydalanılan ML algoritmaları aşağıda verilmiştir:

1. LSTM, sinir ağları tabanlı Uzun Kısa Süreli Bellek (Long Short Term Memory), NLP alanında önemli bir role sahiptir. Ayrıca, ardışık (sequence) modelleme için yaygın olarak kullanılmıştır. LSTM'lerin kullanımı yaygındır çünkü model, örnekler ilerlerken kendisine geri döner ve böylece herhangi yeni bir örnek için tahmin yapılırken önceki tahminler tarafından oluşturulan bağlamdan yararlanır. Tensorflow ve Keras kütüphanelerinden faydalanır. (Versloot, 2021).

   LSTM, uzun süreli belleği tekrarlayan sinir ağlarına (Recurrent Neural Networks - RNN) tanıtır. Bir ünite içinde üç tip geçit vardır; girdi geçidi, girdiyi hücreye ölçekler (yazma); çıktı geçidi, çıktıyı hücreye ölçekler (okuma); unutma geçidi ise, eski hücre değerini ölçekler (resetleme). Her geçit, okuma/yazmayı kontrol eden ve böylece uzun süreli hafıza fonksiyonunu modele dahil eden bir anahtar gibidir. LSTM'in el yazısı tanıma, zaman serisi anomali tespiti, konuşma tanıma, dilbilgisi öğrenme, müzik besteleme, vb. gibi kullanılabileceği birçok yol vardır (Gall, 2018).

LSTM veri seti üzerinden uygulanırken, bilinmesi gereken birçok noktası bulunmaktadır. Bu çalışmada LSTM algoritmasından yararlanılırken aşağıdaki kavramlar kullanılmıştır.

Simgeleştirme (Tokenize): Bu, LSTM ağı için bir katman değil, kelimelerimizi simgelere (tamsayılara) dönüştürmek için zorunlu bir adımdır (Agrawal, 2019). Simgeleştirme sınıfı için bazı değişkenler olmalıdır. Bu değişkenler aşağıda gösterilmiştir:

- Kelime sayıları (num_words): Kelime sıklığına bağlı olarak tutulacak maksimum kelime sayısıdır ve yalnızca en yaygın kelimeler tutulmaktadır.
- Kelime kümesi dışındaki simgeler (Oov_tokens): Sözcük dağarcığı dışındaki kelimeleri etiketlemek için kullanılır.
- Sözcük dizini (word_index): Sıralarına/dizinlerine göre kelimeleri eşleyen adlandırılmış listedir. Yalnızca tokenizer'da "fit_text_tokenizer()" çağrıldıktan sonra ayarlanır (Kalinowski ve ark.).
- Veri işlenirken "sequence.pad_sequences" kullanımı şekil içeren girdi verilerinin listesini (veriyi) iki boyutlu NumPy şekil dizisine (veri, zaman aralıklarına) dönüştürür. Temel olarak, verilere zaman aralıkları kavramını ekler. Uzunluğun zaman aralıklarını (maxlen) üretir (Tutorials Point (I) Pvt. Ltd. - Keras, 2019, s.:85).
- Keras web sitesine göre Blok dizileri (pad_sequences) aynı uzunlukta olmalıdır. Doldurma (padding) veya kesmenin (truncation) gerçekleştiği konum, sırasıyla doldurma ve kesme bağımsız değişkenleri tarafından belirlenir. Dizinin başlangıcındaki değerleri önden (pre) doldurma veya kesme varsayılan seçenektir.
- Doldurma / Kesme (Padding / Truncating) İşlemi: Hem kısa hem de uzun metinlerle başa çıkmak için belirli bir uzunluğa göre metni doldurur (padding); ya da baştan/sondan metni keser (truncating). Bu uzunluk dizi uzunluğu (sequence length) olarak tanımlanır.

Veriler iyi bir şekle sokulduğunda, eğitim ve test setlerine ayrılabilir (Agrawal, 2019). Keras'a göre metin verisi önişleme bağımsız değişkenleri şunlardır:

- Küme boyutu (batch_size): Veri gruplarının boyutudur. Varsayılan değer 32'dir.
- Maksimum uzunluk (max_length): Bir metin dizesinin maksimum boyutu. Bundan daha uzun metinler maksimum uzunluğa göre kısaltılacaktır.
- Karıştırma (shuffle): Verilerin karıştırılıp karıştırılmayacağıdır. Varsayılan değeri "Doğru" dur.
- Kaynak (seed): Karıştırma ve dönüşümler için isteğe bağlı rastgele kaynak sunar.

Ardışık (Sequential) model, her bir katmanın tam olarak bir girdi ve çıktı tensörüne sahip olduğu düz bir katman yığını için uygundur. Ardışık model ekle "add()" metoduyla oluşturulabilir. Model inşa edilir edilmez, özet "summary()" metoduyla içerik görüntülenebilir (Chollet, 2020). Daha sonra LSTM katmanları (layers) eklenebilir. Tez çalışmasında kullanılan katmanlar aşağıda yer almaktadır:

a. LSTM Katmanı: gizli konum boyutları ve katman sayısıyla tanımlanır.
b. Gömme (Embedding) Katmanı: kelime simgelerini (tamsayıları) belirli boyutta bir yerleşime dönüştürür.
c. Aktivasyon Katmanı: Tüm çıktı değerlerini 0 ile 1 arasında bir değere çevirir.
d. Yoğun (Dense) Katman: Tam bağlantı oluşturur ve LSTM katmanının çıktısını istenen çıktı boyutuna eşler (Agrawal, 2019). Keras'a göre Yoğun katmanın bir değişkeni olarak softmax kullanılabilir. Softmax, değerler vektörünü bir olasılık dağılımına dönüştürür. Sonuç bir olasılık dağılımı olarak yorumlanabileceğinden, Softmax genellikle bir sınıflandırma ağının son katmanı için aktivasyon olarak kullanılır.

e. Bırakma (Dropout) Katmanı: Eğitim süresi boyunca her adımda bir oran frekansı ile girdi birimlerini rastgele 0'a ayarlar, bu da aşırı öğrenmeyi önlemeye yardımcı olur. Oranı 0 ile 1 arasında reel sayı olmalıdır.
f. Düzleştirme (Flatten) Katmanı: Girdiyi düzleştirir. Küme büyüklüğünü (batch size) etkilemez.
g. Çift Yönlü (Bidirectional) Katman: RNN'ler için çift yönlü modül oluşturur. Girdi hem sağdan sola hem de soldan sağa okunur.

En iyileştiriciler (Optimizers): Keras modelini derlemek için gereken iki bağımsız değişkenden biridir: "compile()" (derleme) veya "fit()" (uydurma) gibi. Optimizer için aşağıdaki gibi varsayılan parametreler kullanılabilir.

→ "model.compile(loss='categorical_crossentropy', optimizer='adam')"

Adam, optimizer için bir yöntemdir ve sadece az bellek ihtiyacı ile birinci derece eğimler gerektiren verimli bir tahmin sağlar. Yöntem, eğimlerin birinci ve ikinci kuvvetlerinin tahminlerinden farklı parametreler için bireysel uyarlanabilir öğrenme oranlarını hesaplar. İsmen "Adam", "adaptive moment estimation (uyarlanabilir kuvvet tahmini)" dan türetilmiştir (Kingma & Ba, 2015). Keras'a göre Adam öğrenme oranı çok yüksek olmamalıdır, yoksa girdi aşırı öğrenmiş olur, o yüzden varsayılan değer 0.001 olarak geçmektedir. Bozunma oranı (decay rate) 1. kuvvet tahminleri varsayılan olarak 0.9, 2. kuvvet tahminlerinde ise varsayılan olarak 0.999 olarak kullanılmaktadır. Sayısal tutarlılık için sabit bir katsayı, sıfıra çok yakın olan değer (epsilon) varsayımı da '1e-7' olarak ifade edilmektedir.
Optimizer'da model derlenirken kayıp (loss) fonksiyonu da kullanılabilir. Kayıp fonksiyonlarının amacı, bir modelin eğitim sırasında en aza indirmeye çalışması gereken miktarı hesaplamaktır. Loss fonksiyonu seyrek kategorik çapraz etkinlik ölçümü (sparse categorical crossentropy) olarak seçilebilir. Bu değişken etiketler ve tahminler arasındaki çapraz etkinlik ölçüm kaybını hesaplar. İki veya daha fazla etiket sınıflarında kullanılır. Etiketlerin tamsayı olarak getirilmesi beklenir.

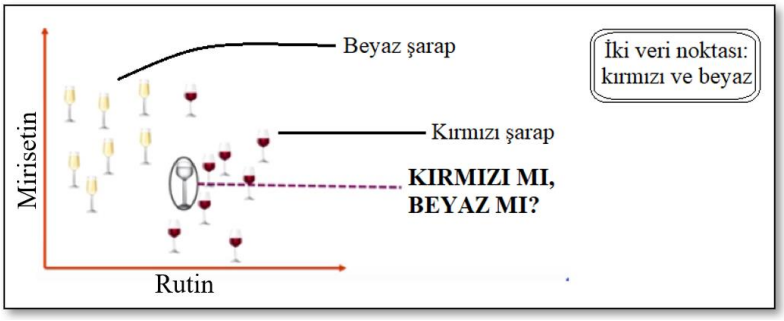
Yine Optimizer'da metrik fonksiyonu da seçilebilir. Metrik, modelin performansını değerlendirmek için kullanılan bir işlevdir. Metrik işlevleri, bir metriğin değerlendirilmesinden elde edilen sonuçların model eğitilirken kullanılmaması dışında kayıp işlevlerine benzer. Herhangi bir kayıp işlevi metrik olarak da kullanılabilir. Doğruluk (accuracy) metriği seçildiğinde, tahminlerin etiketlere ne sıklıkta eşit olduğunu hesaplanır. Bu sıklığı hesaplarken toplam (total) ve sayı (count) olmak üzere iki yerel değişken oluşturur.

En son, eğitim sırasında toplanan tüm bilgileri içeren bir geçmiş (history) nesnesi üzerinde model uydurma yapılırken "model.fit()" birçok değişken belirlenmelidir:

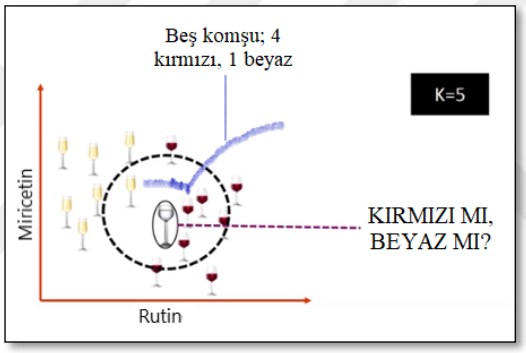
- x: Vektör, matris veya eğitim verisi dizisidir.
- y: Vektör, matris veya etiket veri dizisidir.
- Devir (Epoch): modelin eğitilmesindeki devir sayısıdır.
- Ayrıntı (Verbose): Ayrıntı modudur (0 = sessiz, 1 = ilerleme çubuğu, 2 = devir başına bir satır).
- Validation data (Doğrulama verileri): her devrin sonunda kaybın ve herhangi bir model ölçümünün değerlendirileceği verilerdir. Model bu verilerde eğitilmeyecektir.

KNN (K-Nearest Neighbours), "en yakın k komşu" anlamına gelmektedir ve mevcut tüm durumları depolayan ve yeni verileri veya durumu bir benzerlik ölçüsüne göre sınıflandıran basit bir algoritmadır. Çoğunlukla komşularının nasıl sınıflandırıldığına dayandırılmış bir veri noktasını sınıflandırmak için kullanılır. KNN'deki 'k', veri noktasının nerede sınıflandırılacağına karar vermek amaçlı en yakın komşuların sayısını ifade eden bir parametredir. Şarap örneğine bakıldığında, kırmızı ve beyaz şaraptaki Mirisetin ve Rutin kimyasal bileşen seviyelerine bakılmış, seviye ne kadar ise ona göre kırmızı ve beyaz şarap kadehlerini ayıran bir sınıflandırma yapılmıştır (Şekil 1.15.). Buna göre yeni gelen bir kadeh şarabın hangi sınıfta olduğuna karar verilmelidir. KNN algoritmasında 'k' değeri 5 ise, en yakın 5 kadehe bakılması gerekir. En yakın 5 kadehe bakıldığında; 4'ü kırmızı, 1'i beyaz çıkmıştır, böylece yeni gelen kadehte kırmızı şarap vardır tahmini yapılmıştır

(Şekil 1.16.). KNN algoritmasında ‘k’, öznitelik benzerliğine dayalıdır ve doğru ‘k’ değerini seçmek, daha iyi bir doğruluk için önemlidir. ‘k’ değeri deneme yanılma yöntemiyle doğru bir seçime ulaşır (Subramanian, 2019).



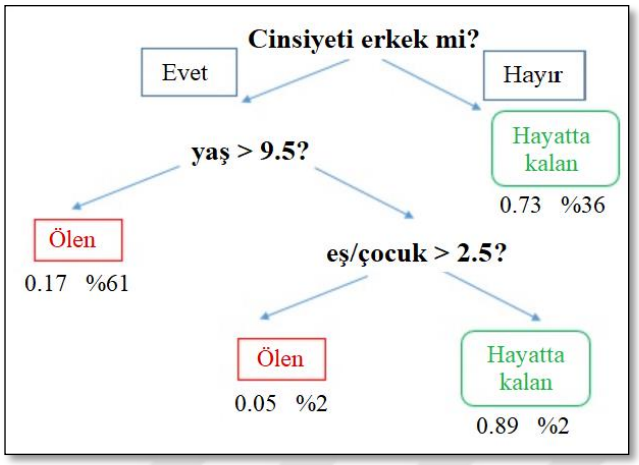
Şekil 1.14. Şarap örneğinde kırmızı ve beyaz kadehlerin mirisetin ve rutin seviyelerine göre ayrımı grafiği.



Şekil 1.15. Yeni bir kadeh şarabın KNN algoritmasına göre sınıflandırılması.

2. Decision Tree Classifier, “Karar Ağacı Sınıflandırıcısı” olarak geçmektedir ve Wikipedia’ya göre daha çok sınıflandırma problemlerinde kullanılır. Sadece ayrık değil aynı zamanda sürekli bağımlı değişkenler için de çalıştırılabilir. Bu algoritma veriyi iki ya da daha fazla homojen setlere ayırır. Mümkün olduğunca farklı gruplar oluşturmak için en önemli özniteliklere bakarak bu ayırmayı gerçekleştirir. Karar ağaçları hem sınıflandırma hem de regresyonu kapsayan makine öğreniminde yaygın olarak kullanılmaktadır. Karar analizinde, kararları ve karar vermeyi görsel ve açık bir şekilde temsil etmek için bir karar ağacı kullanılır. Bu karar ağacı, aslında ağaç benzeri bir karar modelidir. Kökü üstte, dalları altta olacak şekilde bir karar ağacı çizilir. Şekil 1.16’da Titanik veri seti örneğine bakıldığında yolcuların ölüp ölmediğini tahmin edilmeye çalışılmaktadır. Model, cinsiyet, yaş ve eş/çocuk

olmak üzere 3 özniteliği/sütunu kullanmaktadır. Buna göre ölen ve hayatta kalan yolcuların oranları kırmızı ve yeşil renklerde gösterilmiştir.



Şekil 1.16. Karar Ağacı Örneği.

3. Suppor Vector Machines (SVM) "Destek Vektör Makineleri" olarak adlandırılır ve Tutorialspoint'e göre veriyi farklı kategorilere ayıran iyi bilinen denetimli sınıflandırma algoritmasıdır. Vektörler hattı en iyi hale getirerek sınıflandırılır böylece grupların her birindeki en yakın nokta birbirinden en uzak nokta olacaktır. Sklearn'e göre SVM için sınıflandırıcı olan SVC (Support Vector Classifier) kullanılır. SVC birden fazla sınıflar için kullanılabilir. Parametreleri arasında çekirdek (kernel) parametresi varsayılan olarak "rbf"dir (radyal temelli fonksiyon), ancak "linear" (doğrusal), "poly" (çoklu), "sigmoid" veya "precomputed" (önceden hesaplanmış) de seçilebilir. Yine de yaygın kullanılanların seçilmesi önerilmektedir.
4. Naive Bayes, Tutorialspoint'e göre, tahmin değişkenlerinin bağımsız olduğu varsayımıyla Bayes teoremine dayalı bir sınıflandırma tekniğidir. Özetle, bir Naive Bayes (NB) sınıflandırıcısı, bir sınıftaki belli bir özelliğin varlığının başka herhangi bir özelliğin varlığı ile ilgili olmadığını varsayar. Naive Bayesian modelini uygulamak kolaydır ve çok büyük veri kümelerinde oldukça kullanışlıdır. Basit olmaktan ziyade, son derece gelişmiş sınıflandırma yöntemlerinden bile daha iyi bir performans gösterir. Metin sınıflandırmalarında kullanılması yaygındır. Sklearn'e göre Naive Bayes 5 adet sınıflandırıcısı bulunmaktadır:
    a. Katlıterim (Multinomial) NB, katlı çok terimli modeller için bir sınıflandırıcıdır ve ayrık özelliklere sahip sınıflandırma için uygundur

(örneğin, metin sınıflandırması için kelime sayıları gibi). Katlıterim dağılımı normalde tamsayı özellik sayımlarını gerektirir ancak uygulamada TF-IDF gibi kesirli sayımlar da işe yarayabilir.

b. Bernoulli NB, çok değişkenli modeller için bir sınıflandırıcıdır ve ayrık veriler için uygundur. İkili (binary) özellikler için tasarlanmıştır.
c. Kategorik (Categorical) NB, kategorik özellikler için kullanılan bir sınıflandırıcıdır. Kategorik olarak dağıtılmış ayrık özelliklere sahip sınıflandırmaya uygundur.
d. Tümleyici (Complement) NB, standart Katlıterim (Multinomial) NB tarafından yapılan "keskin varsayımları" düzeltmek için tasarlanmıştır ve özellikle dengeli olmayan veri setleri için uygundur.
e. Gaussian NB'de, özelliklerin olasılıkları Gaussian olarak tahmin edilir.

5. Multi-layer Perceptron (MLP) Classifier "Çok Katmanlı Algılayan Sınıflandırıcı" anlamına gelmektedir ve sinir ağlarına dayalı bir sınıflandırıcıdır. Wikipedia'ya göre MLP üç katman devresinden oluşur; girdi katmanı, gizi katman ve çıktı katmanıdır. MLP'ler, problemleri olasılıksal olarak çözme yetenekleri nedeniyle araştırmalarda faydalıdır. MLP'ler, konuşma tanıma, görüntü tanıma ve makine çevirisi yazılımı gibi çeşitli alanlarda uygulanmıştır.

1.4.7. Yazarlık Analizi Yapılmış Çalışma Örnekleri

Bu tez çalışması için yazarlık analizi yapılmış birçok çalışma incelenmiştir. Bu tez çalışmasına en yakın bazı çalışmaların, bu çalışmayla olan benzerlikleri ve farklılıkları aşağıda tartışılmıştır.

1. Adli Yazarlık Analizinde Ayırt Edici Göstergeler Olan Kanıtsallık Stratejileri adlı çalışma kanıtsallık stratejilerinin sıklığının yazara özgü olup olmadığı ve İngilizce dilinde sözlük-dilbilgisel ifadelerin yazarlık analizinde ayırt edici göstergeleri açığa çıkarıp çıkarmadığını incelemektedir. Bu araştırma 5 yazardan gelen 19 örnek ile oluşturulmuştur. Araştırmanın nitel yönü, verilen

ifadelerin kullanımında yazara özgü belirli eğilimlerin tanınmasının yanı sıra, kanıtsallığı belirtmek için kullanılan sözlük-dilbilgisel ifadelerin saptaması ve sınıflandırılmasını göstermektedir. Çalışma sonucunda kanıtsallık ve sözlüksel-dilbilgisel kanıtsal ifade türlerinin seçiminde yazara özgü belirli alışkanlıklar gözlemlenmiştir (Tomić, 2019). Bu araştırma yazarlık analizi içerdiğinden bahsi geçen tez çalışmasına benzemektedir ancak tez çalışması kodlar aracılığıyla gerçekleştirildiği ve bilişim alanını da bünyesine kattığı için bu çalışmadan ayrı tutulabilir.

2. Öznitelik Madenciliğiyle Türkçe Metinlerin Yazar Atıfları adlı çalışma Yıldız Teknik Üniversitesi tarafından yapılan Yazarlık Analizi çalışmalarından biridir. Yıldız Teknik Üniversitesi Yazarlık Analizi içeren birçok çalışma yürütmüştür ve yürütmektedir.
   "http://www.kemik.yildiz.edu.tr/yayinlarimiz.html" sitesi incelendiğinde yapılan çalışmalar görülmektedir. Öznitelik Madenciliğiyle Türkçe Metinlerin Yazar Atıfları çalışması ise WEKA uygulaması üzerinden metin madenciliği yapmaktadır. Yazarların belirlenmesi amaçlanan dokümanlardan yazarlık özniteliklerinden, n-gramlarından ve bu öznitelik vektörlerinin çeşitli kombinasyonlarından on farklı öznitelik vektörü elde edilir. Naive Bayes, SVM, k-NN, RF ve MLP sınıflandırma yöntemleri uygulanarak her öznitelik vektörünün karşılaştırmalı performansı analiz edilir. Çalışmanın sonucunda göre en başarılı sınıflandırıcılar MLP ve SVM'dir (Türkoğlu ve ark., 2007). Bu çalışma tez çalışmasına Yazarlık Analizi ve uygulanan algoritmalar açısından benzer, ancak tez çalışması Yazarlık Analizini WEKA yerine Python üzerinden uygulamaktadır.
3. Twitter'da İntihar İçerikli İletişimin Makine Sınıflandırması ve Analizi adlı çalışma İngilizce dilinde Twitter'da intihar içerikli metinleri sınıflandırmak için oluşturulmuş bir dizi makine sınıflandırıcılarını rapor etmek ve ciddi olmayanları belirlemektir. Twitter gönderilerinden çıkarılan sözcüksel, yapısal, duygusal ve psikolojik özellikleri kullanarak bir takım temel sınıflandırıcı oluşturulmuştur. Bu analizden yola çıkılarak online intihar içerikli tartışma forumlarından ve diğer mikroblog web sitelerinden alınan sözcük listelerinin ve düzenli ifadelerin hem tek sözcükler, n-gramlar (kelime

listeleri) hem de daha karmaşık kalıplar açısından ilgili dil 'ipuçlarını' yakalayabildiğini gözlemlenmiştir (Burnap ve ark., 2015). Bu tez çalışmasında da bahsi geçen çalışmadaki gibi makine sınıflandırıcıları kullanılmıştır ve veri seti online içerikten toplanmıştır.

4. Online Metinlerde Yazar Tanıma adlı yüksek lisans tezi İngilizce yazar tanıma araştırma alanında kullanılan yöntem ve teknikleri araştırmış; ardından bunları uygulanabilir bir prototip haline getirmeyi amaçlamıştır. Veri setleri e-postalar ve yazılardan oluşan hazır 2 corpustan çekilmiştir. Daha sonra uygulanan prototip, nitel (verilen metinlerin orijinal yazarlarını belirlemedeki doğruluk düzeyi) ve nicel (prototipin uygulama süresi) nitelikleri açısından incelenmiş ve değerlendirilmiştir. Çalışma sonucunda denetimli öğrenme tekniklerinin yanı sıra denetimsiz öğrenme tekniklerinin de mevcut yöntem ve tekniklerin performansını geliştirmeye yardımcı olabileceğini göstermektedir (Dinh, 2014). Mevcut tez çalışması, bu tez çalışması ile karşılaştırıldığında denetimli öğrenme tekniklerinin kullanılması açısından birbirine benzemektedirler, ancak veri seti köşe yazılarından oluşmaktadır.
5. Elektronik Postaların Adli Analizinde Yazar Analizi Tekniklerinin Kullanılması adlı yüksek lisans tez çalışması Türkçe dilinde gerçekleştirilmiştir. Bu çalışmada, mesaj birimine dayalı elektronik postaların güvenliğini artırabilecek ve 5 yazardan gelen 250 mesaj için elektronik postaların bilinen sahibi yerine gerçek sahibini bulabilecek bir uygulama, elektronik postaların yazarlarını adli bilimler ve veri seti olarak elektronik posta ile aynı özelliği taşıyan forum mesajları açısından belirlemek amacıyla gerçekleştirilmiştir. Yöntemlerin başarı oranları arasındaki farkın temel nedeni seçilen veri setine, çıkarılan metin özelliklerine, ön işleme adımlarına ve algoritma parametrelerine bağlıdır. Sonuç olarak yazarlık tespiti, gerçek suçluları tespit etmede oldukça başarılı bir yöntemdir ve adli bilimler alanında çalışan bilim insanları için faydalıdır (Ekinci, 2013). Bu tez çalışması, mevcut tez çalışması ile adli bilimlere sunmuş olduğu katkı benzerdir, ancak kullanılan algoritmalar farklı konumdadırlar. Benzer olarak

karar ağacı, Naive Bayes, Bagging sınıflandırma algoritmaları kullanılmış, sonuçlar da benzer yöntemlerle elde edilmiştir.

6. Online Sosyal Platformlarda Yazar Tanıma yüksek lisans tez çalışması Türkçe dilinde "bir sosyal platformda engellenen kullanıcıların farklı kimlikle geri dönmesi durumunda kimliğinin tespit edilmesi ya da sahte hesapların ardındaki kişilerin ortaya çıkarılmasını" amaçlamaktadır. Veriler Ekşisözlük ve COPA isimli online bir oyun platformunda ikiden fazla kişinin online grup sohbetlerinden toplanmıştır. Bu çalışma sonucunda örneğe dayalı yazar atıflama yöntemi, iyi yapılandırılmış resmi metinsel verilerde daha iyi performans gösterirken profil tabanlı yaklaşım, resmi olmayan veri setlerinde yazar atıfları için daha iyi olduğunu göstermektedir. Sorgu metni az olsa bile yazar tanıma yine de mümkün olmuştur (Kuzu, 2010). Mevcut tez çalışması bu tez çalışmasına denetimli algoritmalar kullanması açısından benzerlik göstermektedir. Ancak bu tez çalışması ayrıca İngilizce ve Portekizce dillerinde veri setleri de bulundurmuştur.
7. Metin Madenciliği Yöntemleri ile Yazar Tanıma: Divan Edebiyatı Örneği adlı tez çalışması ise yine Türkçe dilinde gerçekleştirilmiştir. Bu çalışmada Divan Edebiyatına ait 25 şiir eserinin yazarlarını belirlemek için bir yapı geliştirilmiştir. Bu sistemde metin madenciliğinin metin sınıflandırma algoritmaları kullanılıp kelimeler çözümlenmiştir. 20 farklı model her parametrenin olası değerleri için kurulmuştur. Bu araştırmanın, bilinmeyen eserlerin yazarlarının belirlenmesine dair tahminleri uzun vadede destekleyebileceği varsayılmaktadır. Sonuç olarak, divan edebiyatı eserleri için kelime bazlı yaklaşımla metin sınıflandırma işlemi başarılı olarak kaydedilmiştir (Bilgin, 2018). Bu tez çalışması mevcut tez çalışmasına yazar tanıma açısından benzemektedir, ancak mevcut tez çalışmasının veri setleri köşe yazılarından seçilmişken bu tez çalışması şiirleri içermektedir.

Bütün bu açıklanan kavramlardan hareketle yazarlık analizinin ana fikri aslında bir yazarın kendine özgü özelliklerin çıkarılması olduğu savunulabilir. Bu stilistik özellikler NLP adımları ile elde edilecek olup, makine öğrenme algoritmalarına hazır haline getirildikten sonra bahsi geçen algoritmalardan geçirilecektir. Algoritma sonuçlarından elde edilecek verilerle bu çalışmanın amacı sadece söz konusu köşe yazılarının bağlı oldukları yazarlara ait olup olmadığını göstermekle kalmayıp, aynı zamanda bilişim suçlarının en yaygını olan intihali önleme bir öncü olması hedeflenmiştir. Ayrıca sosyal medya gibi bilişim ortamlarında yazarı belli olmadığı düşünülen herhangi bir paylaşımı tespit etmede de yardımcı olacağı düşünülmektedir.

2. GEREÇ VE YÖNTEM

2.1. Verilerin Toplanması

Bu çalışma; yazarları ve köşe yazılarını seçme, köşe yazılarını web sitelerinden çekme, bu yazıları önişlemden geçirme ve önişlemden geçirilmiş köşe yazılarını algoritmalara sokma olarak 4 aşamadan oluşmaktadır. Yazarlar ve köşe yazıları Ağustos 2021 tarihinde seçilmiş ve yazılan kodlar aracılığıyla bağlı oldukları web sitelerinden çekilmişlerdir. Eylül 2021 tarihinde ise bu yazılar önişlem adımlarından geçirilmiştir. Ekim 2021 tarihinde ise önişlemden geçirilen yazılar algoritmalara sokulup sonuçları not edilmiştir.

2.1.1. Birinci Aşama

Köşe yazılarında yazarlık analizi yapmadan önce 6 köşe yazarı seçilmiştir. Bu köşe yazarları "https://www.hurriyet.com.tr/yazarlar/tum-yazarlar/" web sitesinden seçilmiştir. Yazarlar farklı alanlardan olmak üzere rastgele seçilmiştir. Tarih ve edebiyat konularında Doğan HIZLAN ve İlber ORTAYLI; mutfak konularında Müge AKGÜN ve Vedat MİLOR, sosyal konularda Melike KARAKARTAL ve Yaşar SÖKMENSÜER seçilmiştir. Her yazardan 160'şar yazı seçilmiştir. Toplamda 960 metin üzerinde çalışılmıştır ve her yazardan daha fazla metnin algoritmalara sokulabilmesi için yazılar eşit tutulmuştur. 960 metin, yazarlık analizi yapmak üzere sınırlı olsa da diğer çalışmalara nazaran elde edilen verilere göre yeterli görülebilir.

2.1.2. İkinci Aşama

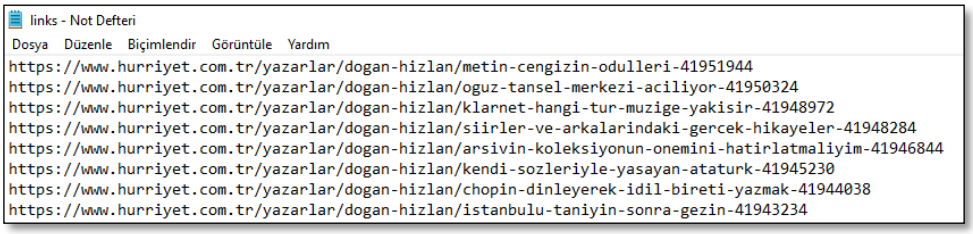
Çalışmanın bu aşamasında kodlar, Visual Studio Code (VSC) uygulaması aracılığıyla yazılmıştır. Visual Studio Code, Vikipedi'ye göre "Microsoft tarafından Windows, Linux ve MacOS için geliştirilen bir kaynak kodu düzenleyicisidir. Gömülü Git kontrolü, hata ayıklama, sözdizimi vurgulama, akıllı kod tamamlama,

snippetler ve kod yeniden yapılandırma desteği içerir. Ayrıca VSC özelleştirilebilir, böylece kullanıcılar klavye kısa yollarını, tercihlerini ve editörün temasını değiştirebilir. Resmi indirme işlemi tescilli bir lisans altında olsa da ücretsiz ve açık kaynaktır.”

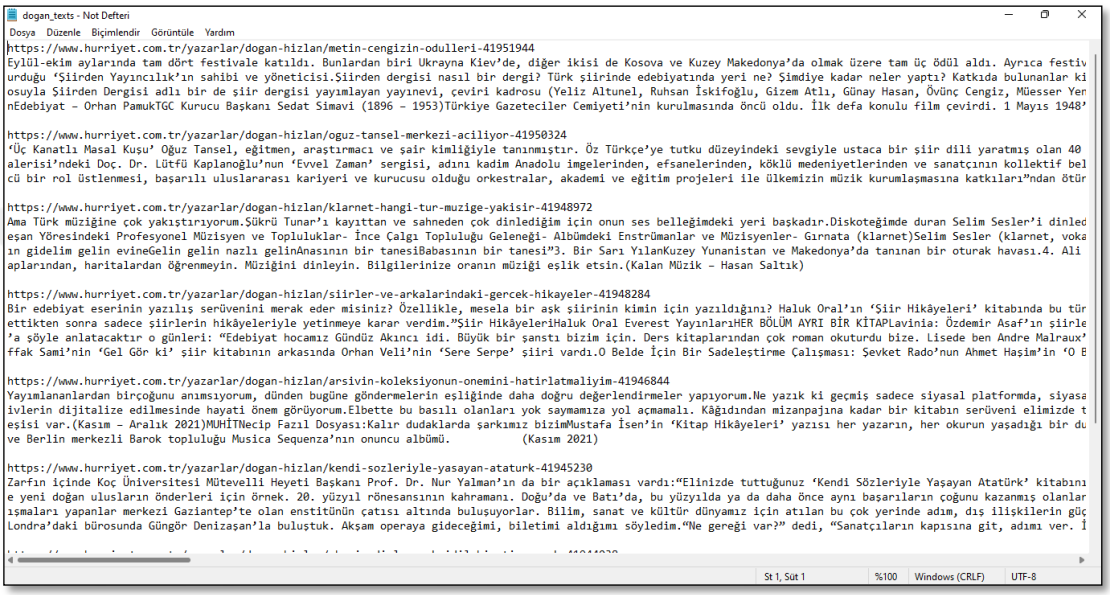
VSC’a entegre edilen Python 3.9.7 aracılığı ile, öncelikle yazarların bağlı olduğu web sitelerinden metin çekme işlemi belirli kodlarla getirilmiştir. Metinleri web sitelerinden çıkarma işlemi için urllib ve BeautifulSoup kütüphanelerinden yararlanılmıştır. Tek bir metin çıktısı Şekil 2.1.’de gösterilmektedir.

Şekil 2.1. Web sitesinden çekilen bir metin örneği.

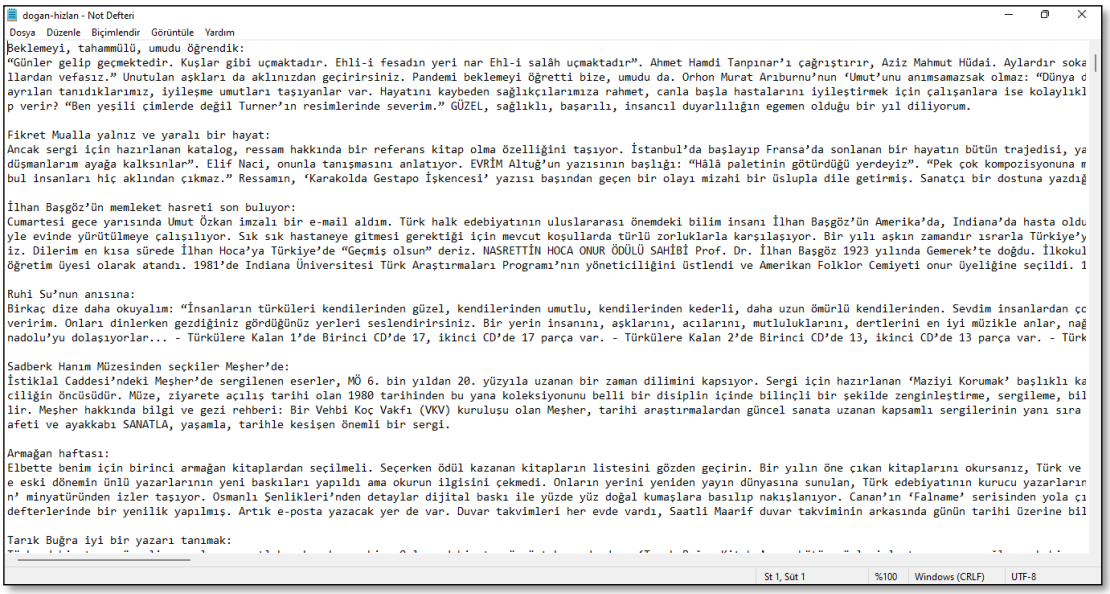
Tek bir metin örneği çalıştıktan sonra bir yazarın bütün yazıları ve bağlı oldukları linklerin çıkarılması ve .txt uzantılı belgelere kaydedilmesi isin kodlar yazılmıştır. Yazılan kodlarda, web sitesi inceleme bölümlerinden tam metin(ler)in içerildiği sınıf (class) “article-content news-text”, “highlighted-box mb20” bölmeleri koda yazılarak gerçekleştirilmiştir. Linkler (Şekil 2.2.) ayrı, linkler ve bağlı oldukları köşe yazıları (Şekil 2.3.) ayrı bir .txt dosyasına kaydedilmiştir. Verilen örneklerde seçilen köşe yazarlarından olan Doğan HIZLAN yazıları ve linkleri bulunmaktadır. Diğer yazarların köşe yazıları da aynı yöntemle web sitelerinden çekilmiştir. Kaydedilen .txt uzantılı dosyalar kontrol edildiğinde bazı metinlerin bozuk çıktığı ve metin içerisinde makine tarafından okunamayacağı düşünülen bazı simgeler görülmüştür. Bu kısımlar da metinden çıkarılmış böylece önişlem adımına uygun hale getirilmiştir (Şekil 2.4.).



Şekil 2.2. Web sitelerinden çekilen linkler örneği.



Şekil 2.3. Web sitelerinden çekilen linkler + metinlerin örnekleri.



Şekil 2.4. Web sitelerinden çekilen metinlerde bozuk kısımların ve simgelerin çıkarıldığı '.txt' örneği.

2.1.3. Üçüncü Aşama

Web sitelerinden çekilen her yazarın köşe yazıları ayrı '.txt' uzantılı dosyalara kaydedilmiştir ve hepsi "author_n_columns" adlı bir dosyada tutulmuştur. Dosya önişleme sokulmadan önce her yazarın karakter, kelime ve köşe yazısı adetlerine kodlarla bakılmıştır (Çizelge 2.1.).

Çizelge 2.1. Yazarların seçilen köşe yazısı adetleri ve köşe yazılarında bulunan karakter/kelime adetleri.

YAZAR	KARAKTER ADEDİ	KELİME ADEDİ	KÖŞE YAZISI ADEDİ
Doğan HIZLAN	576985	73589	160
İlber ORTAYLI	1067766	137298	160
Melike KARAKARTAL	605947	77637	160
Müge AKGÜN	914056	121290	160
Vedat MİLOR	625817	85990	160
Yaşar SÖKMENSÜER	521417	67913	160

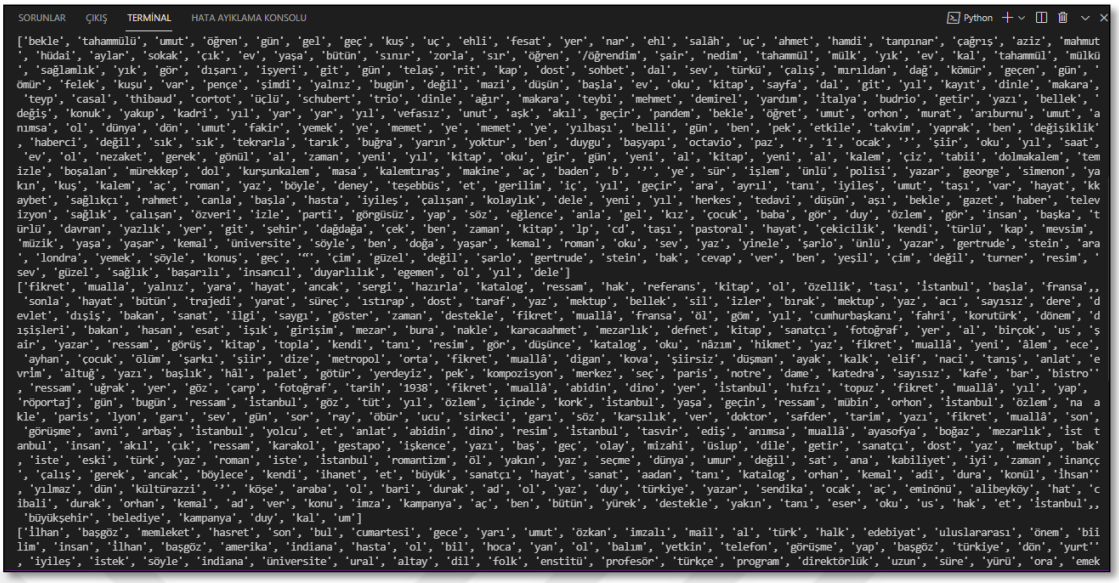
Daha sonra bu dosyalar kodlar aracılığıyla Python üzerinden önişlemden geçirilmiştir. Önişlem adımları uygulanırken "nltk" ve "stanza" kütüphanelerinden yararlanılmıştır. Bu kütüphanelerden faydalanırken, önişlem kodlaması 3 adımdan oluşmuştur. Köşe yazılarında bulunan kelimeler kök haline getirilmiş (lemmaları bulunmuş), metinler küçük harflere dönüştürülmüş, durduran ifadeler (stop-words), sayılar ve noktalama işaretlerinden ayıklanmıştır. Bu işlem 6 yazar için de aynı şekilde gerçekleştirilmiştir.

Önişlemin ilk adımı, köşe yazılarında bulunan kelimeleri kök haline getirme (lemmatization) işlemidir. Bu adımda 'stanza' kütüphanesinden faydalanılmıştır; dili Türkçe seçilmiş, işlem birimi, girdi sınıflandırma (tokenization), çoklu kelime göstergesi (multi-word token), sözcük türü (part-of-speech) ve lemma (kök) seçilmiştir. Bu işlemde öncelikle köşe yazılarındaki cümleler kelimelere bölünmüş, Türkçe dilinde kullanılan alfabe yazılarak Türkçe dilinde olmayan kelimeler ve özel isimler çıkarılmaya çalışılmıştır. Ayrıca 'stanza' kütüphanesinde mevcut, evrensel

sözcük türleri (universal part-of-speech) olan yardımcı fiiller (auxiliary), ön ek ve sok ek (adpositions), sayı (number) ve noktalama ifadeleri (punctuation) de çıkarılmaya çalışılmıştır. Kelimelerin uzunluğu da 1'den fazlaysa o kelimeler tutulmuş, diğerleri de atılmıştır. Son olarak elde edilen lemmalar küçük harflere dönüştürülmüştür (Şekil 2.5.).

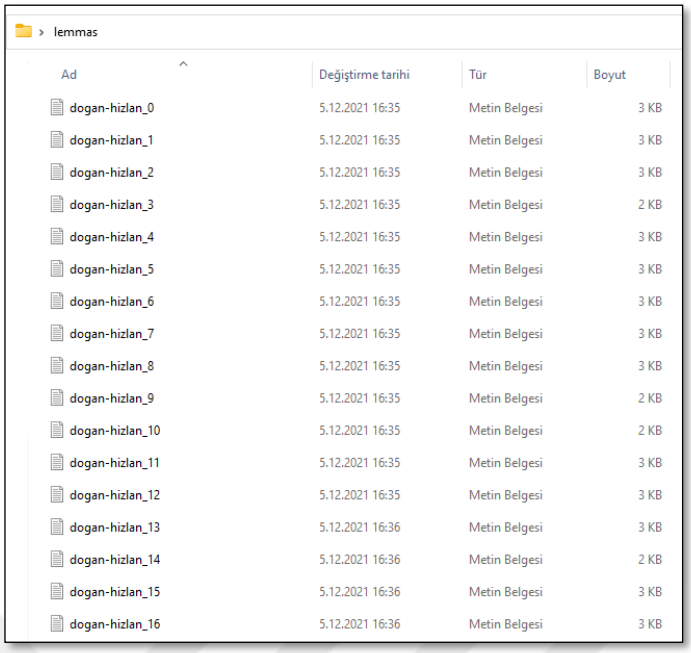
Şekil 2.5. Lemmatization işlemi sonrası çıktı örneği.

Şekil 2.5.'daki çıktı örneğine bakıldığında bazı durduran ifadelerin, noktalama işaretlerinin, sayıların hala var olduğu görülmektedir. Bir sonraki adımda, bu kez 'nltk' kütüphanesi aracılığıyla durduran ifadeler (Şekil 2.6.), sayılar ve noktalama işaretlerinden (Şekil 2.7.) kurtulmak istenmiştir. Ayrıca Türkçe dilinde olmayan bazı harfler ('â' gibi) standart alfabedeki harflere dönüştürülmüştür. Kelimelerin harf sayısı da 1'den küçük ve 1'e eşitse tekrar veri kümesinden çıkarılmıştır (Şekil 2.7.). Köşe yazılarından elde edilen önişlemden geçirilmiş lemmalar algoritmalara sokulmak üzere her bir köşe yazısı için ayrı '.txt' uzantılı dosyalara kaydedilmişlerdir (Şekil 2.8.). Şekil 2.9.'da dosyaların içeriği gösterilmiştir.

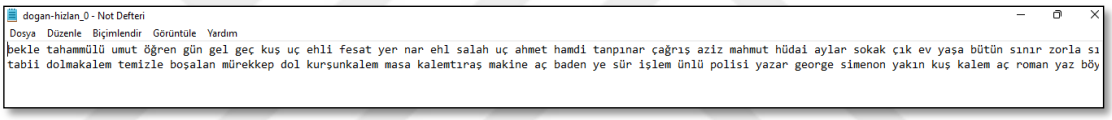


Şekil 2.6. ‘nltk’ kütüphanesinden durduran ifadeler (stop-words) çıkarma işlemi çıktı örneği.

Şekil 2.7. ‘Nltk’ kütüphanesinden sayıların ve noktalama işaretlerinin çıkarılması ve kelimelerdeki bazı harflerin değişimi çıktı örneği.



Şekil 2.8. Her bir köşe yazısı için lemmaların ayrı dosyalara kaydolduğunu gösteren bir örnek.

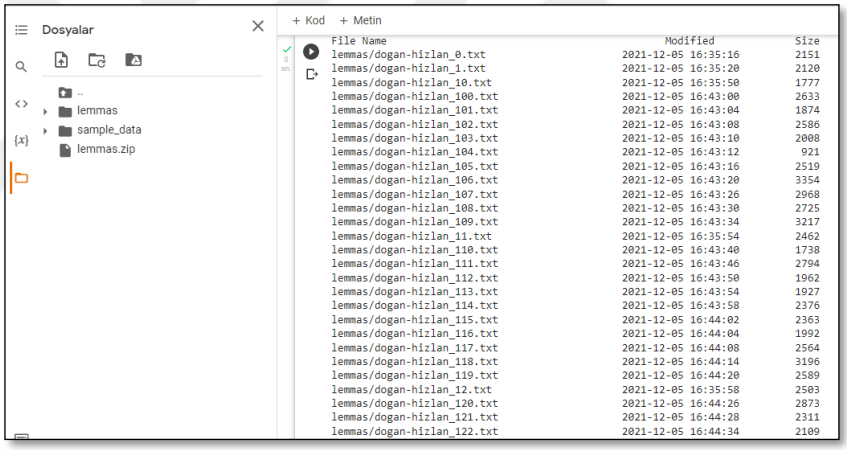


Şekil 2.9. Ayrı dosyalara kaydolmuş bir metinde bulunan lemma örnekleri.

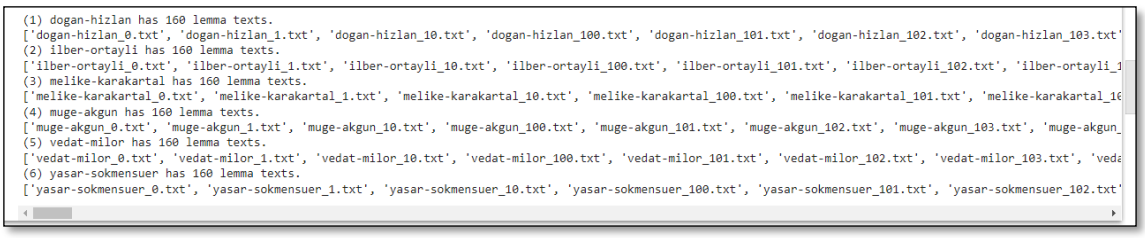
2.1.4. Dördüncü Aşama

Bu veri seti dosyası çalışmanın tamamlanması için makine öğrenme algoritmalarına sokulmalıdır. Veri seti büyük olduğu ve algoritmaların daha başarılı yürütülmesi için online kod işleten Google Colaboratory uygulaması çalışmanın devamı için kullanılmıştır. Google Colaboratory, Jupyter Notebook'un bulut sürümüdür. Hatta site için Jupyter'in önceki adı olan IPython Notebook (.ipynb) dosyalarını kullanır. Kod yazma ve çalıştırma, ilgili belgeleri oluşturma ve grafikleri görüntüleme gibi birçok görevi gerçekleştirmek için Colaboratory kullanılabilir. Colaboratory bir dizi çevrimiçi depolama seçeneğini destekler, böylece Colaboratory Python kodu oluştururken çevrimiçi olarak kullanılabilir (Mueller & Massaron, 2019).

Öncelikle ziplenmiş lemmalar dosyası (Şekil 2.8.) Colaboratory'ye yüklenmiş ve '.zip' dosyasından kurtarılmıştır (Şekil 2.10). Daha sonra her köşe yazarı için ne kadar lemma metni olduğu kontrol edilmiş ve metin dosyaları isimleri çıkarılmıştır (Şekil 2.11.). Ayrıca bu lemmalar dosyasında köşe yazarlarının köşe yazısı başına düşen lemma adetleri ve lemma başına düşen karakter adetleri de istatistiksel iceleme açısından elde edilmiştir (Şekil 2.12.). Şekil 2.13.'e bakıldığında yazarların lemma başına düşen ortalama karakter adetlerinin birbirine yakın olduğu; köşe yazısı başına düşen ortalama lemma adetlerinde de farklılıklar olduğu görülmüştür. İlber ORTAYLI ve Müge AKGÜN'ün köşe yazıları başına düşen ortalama lemma adetleri diğer yazarlara nazaran daha fazladır. Bunun nedeninin işlenmemiş köşe yazılarına bakıldığında İlber ORTAYLI ve Müge AKGÜN'ün yazılarının diğer yazarlara nazaran daha uzun olmasıdır.



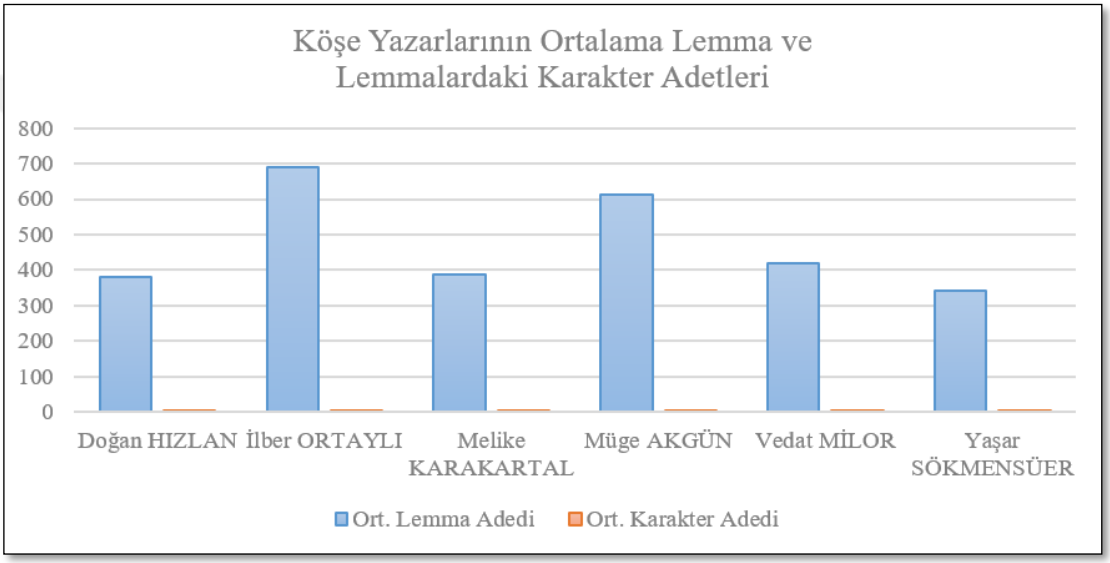
Şekil 2.10. 'lemmas.zip' dosyası '.zip' uzantısından kurtarma ve dosyaları çıkarma örneği.



Şekil 2.11. Tüm köşe yazarları için ne kadar lemma metni bulunduğu ve metinlerin isimlerinin olduğu kod çıktısı.

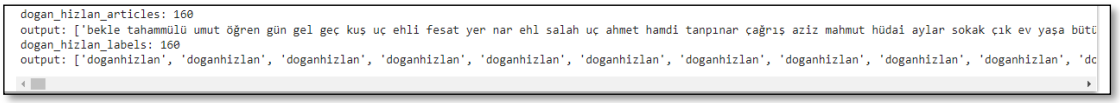
```
dogan-hizlan için köşe yazısı başına düşen ortalama lemma adedi: 379.46875
dogan-hizlan için lemma başına düşen ortalama karakter adedi: 4.987070740344231
ilber-ortayli için köşe yazısı başına düşen ortalama lemma adedi: 688.50625
ilber-ortayli için lemma başına düşen ortalama karakter adedi: 5.182832399851127
melike-karakartal için köşe yazısı başına düşen ortalama lemma adedi: 387.8375
melike-karakartal için lemma başına düşen ortalama karakter adedi: 4.831920585296677
muge-akgun için köşe yazısı başına düşen ortalama lemma adedi: 611.93125
muge-akgun için lemma başına düşen ortalama karakter adedi: 4.957378790509555
vedat-milor için köşe yazısı başına düşen ortalama lemma adedi: 418.625
vedat-milor için lemma başına düşen ortalama karakter adedi: 4.858300985368767
yasar-sokmensuer için köşe yazısı başına düşen ortalama lemma adedi: 342.6125
yasar-sokmensuer için lemma başına düşen ortalama karakter adedi: 4.7482213871356125
```

Şekil 2.12. Köşe yazarlarının köşe yazısı başına düşen lemma adetleri ve lemma başına düşen karakter adetleri kod çıktısı.

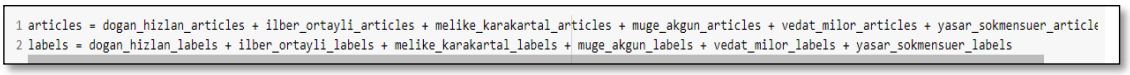


Şekil 2.13. Köşe yazarlarının köşe yazısı başına düşen lemma adetleri ve lemma başına düşen karakter adetleri grafik gösterimi.

Ortalama lemma ve karakter adetleri incelendiktem sonra '.zip' uzantılı dosyadan çıkarılan her köşe yazarına ait lemma dosyaları Şekil 2.14.'de Doğan HIZLAN için yapıldığı gibi her yazar için çıkarılmış ve her lemma dosyası bağlı olduğu köşe yazarı ismi ile etiketlenmiştir. Yazarları ile daha önceden etiketlenen köşe yazıları, köşe yazıları (articles) ve etiketler/yazarlar (labels) olarak ayrı ayrı toplanmıştır (Şekil 2.15.). Veri seti algoritmalara sokulmadan önce vektörlerini bulmak için CountVectorizer ve TfidfVectorizer kullanılmıştır. Daha sonra buna bağlı Öklid mesafesi ve kosinüs benzerliğine bakılmıştır.



Şekil 2.14. Tek yazara ait köşe yazılarının ve etiketleme örneği çıktısı.



Şekil 2.15. Köşe yazılarının "articles" ve bağlı etiketlerinin "labels" olarak ayrı ayrı toplanması.

2.1.4.1. TF-IDF Vektorizer ile Kosinüs Benzerliği Uygulaması

Sklearn kütüphanesinden öznitelik çıkarımı olan TfidfVectorizer ile yazılarda (articles) bulunan tüm lemmalara (Şekil 2.15.) vektör yüklenmiştir ve bu lemmaların dizilişi sağlanmıştır (Şekil 2.16.). Vektörler standartlaştırıldıktan sonra her yazar için yazı aralıkları (Çizelge 2.2.) 50'şer olmak üzere yazılar getirilmiştir. Çekilen yazı aralıkları sorgu yazı (query) aralıklarından farklı olmak üzere Kosinüs benzerliğine (açısal mesafelere) bakmak için her yazarın 50 yazısı için her bir yazarın 7 sorgu yazısı ortalamalarına bakılmış ve birbirine olan açısal mesafeleri çıkarılmıştır (Çizelge 2.3.). Bu mesafelerin 2 boyutlu uzayda gösterilmesi için MDS kullanılmıştır. TfidfVectorizer ile önceden ölçüm yapıldığı için de matris gösterimi buna göre gerçekleştirilmiştir. Matris gösterimi için her yazarın 50 yazısı gösterimine bir renk eklenmiş (Çizelge 2.2.), sorgu yazıları da siyah (black) renkle belirtilmiştir. Yazı aralıkları (50) ve sorgu yazıları (7) sınırlı tutulmuştur çünkü görsel çıktıda fazla örnek fazla renk anlamına gelmektedir, bu da gösterimden sonuç çıkarmayı verimsiz hale getirecektir. Her renk bir yazarı simgelediğinden yazı aralıkları da kodda renklerin bulunduğu kısımlara yazılmıştır.

```
{'bekle': 3086, 'tahammülü': 24467, 'umut': 26445, 'öğren': 29873, 'gün': 10166, 'gel': 9270, 'geç': 9485, 'kuş': 15370, 'uç': 26679, 'ehli': 7194, 'fesat': 84
30700
(960, 30700)
[0. 0. 0. ... 0. 0. 0.]
[0.05828746 0.         0.         ... 0.         0.         0.        ]
[0. 0. 0. ... 0. 0. 0.]
[0. 0. 0. ... 0. 0. 0.]
[0. 0. 0. ... 0. 0. 0.]
[0. 0. 0. ... 0. 0. 0.]
[0. 0. 0. ... 0. 0. 0.]
[0. 0. 0. ... 0. 0. 0.]
[0. 0. 0. ... 0. 0. 0.]
[0. 0. 0. ... 0. 0. 0.]
[0. 0. 0. ... 0. 0. 0.]
[0. 0. 0. ... 0. 0. 0.]
[0. 0. 0. ... 0. 0. 0.]
[0. 0. 0. ... 0. 0. 0.]
[0. 0. 0. ... 0. 0. 0.]
[0. 0. 0. ... 0. 0. 0.]
[0. 0. 0. ... 0. 0. 0.]
[0. 0. 0. ... 0. 0. 0.]
[0. 0. 0. ... 0. 0. 0.]
[0. 0. 0. ... 0. 0. 0.]
[0. 0. 0. ... 0. 0. 0.]
[0. 0. 0. ... 0. 0. 0.]
[0. 0. 0. ... 0. 0. 0.]
[0. 0. 0. ... 0. 0. 0.]
```

Şekil 2.16. TfidfVectorizer ile lemmaları vektörlere çevirme ve dizilişleri.

Çizelge 2.2. Matris gösterimi için yazarların toplam yazı aralıkları ve bağlı renkleri.

Yazarlar	Yazı Aralıkları	Bağlı Olduğu Renkler
Doğan HIZLAN	0-159	kırmızı (red)
İlber ORTAYLI	160-319	mavi (blue)
Melike KARAKARTAL	320-479	yeşil (green)
Müge AKGÜN	480-639	sarı (yellow)
Vedat MİLOR	640-799	mor pembe (magenta)
Yaşar SÖKMENSÜER	800-959	camgöbeği (cyan)

Çizelge 2.3. Seçilen yazıların TfidfVectorizer ile açısal mesafeleri.

Mesafe Adları	Yazar	Karşılaştırılan Yazarlar	Açısal Mesafeler
dist1	Doğan HIZLAN (7 yazı)	Doğan HIZLAN (50 yazı)	0.9668747799221177
dist2		İlber ORTAYLI (50 yazı)	1.2496100475506149
dist3		Melike KARAKARTAL (50 yazı)	1.2418647232504338
dist4		Müge AKGÜN (50 yazı)	1.281062293905369
dist5		Vedat MİLOR (50 yazı)	1.3424590255825775
dist6		Yaşar SÖKMENSÜER (50 yazı)	1.2644258832753692
dist1	İlber ORTAYLI (7 yazı)	Doğan HIZLAN (50 yazı)	1.1394549222525727
dist2		İlber ORTAYLI (50 yazı)	0.8636876296109602
dist3		Melike KARAKARTAL (50 yazı)	1.138304121908154
dist4		Müge AKGÜN (50 yazı)	1.17837838254057
dist5		Vedat MİLOR (50 yazı)	1.2405318994120411
dist6		Yaşar SÖKMENSÜER (50 yazı)	1.1991177936855
dist1	Melike KARAKARTAL (7 yazı)	Doğan HIZLAN (50 yazı)	1.2134495253096311
dist2		İlber ORTAYLI (50 yazı)	1.1902552626007872
dist3		Melike KARAKARTAL (50 yazı)	0.9183288625402322
dist4		Müge AKGÜN (50 yazı)	1.1848960703224183
dist5		Vedat MİLOR (50 yazı)	1.2463495401613696
dist6		Yaşar SÖKMENSÜER (50 yazı)	1.1016689003716575
dist1	Müge AKGÜN (7 yazı)	Doğan HIZLAN (50 yazı)	1.2518729398663238
dist2		İlber ORTAYLI (50 yazı)	1.251435512265189
dist3		Melike KARAKARTAL (50 yazı)	1.2411787473578317
dist4		Müge AKGÜN (50 yazı)	0.8401685645173295
dist5		Vedat MİLOR (50 yazı)	1.07668097782881
dist6		Yaşar SÖKMENSÜER (50 yazı)	1.2692397827584336
dist1	Vedat MİLOR (7 yazı)	Doğan HIZLAN (50 yazı)	1.2497511180295295
dist2		İlber ORTAYLI (50 yazı)	1.2209777734013958
dist3		Melike KARAKARTAL (50 yazı)	1.168995866377791
dist4		Müge AKGÜN (50 yazı)	1.059164321241413
dist5		Vedat MİLOR (50 yazı)	0.9302817773531241
dist6		Yaşar SÖKMENSÜER (50 yazı)	1.2015563103766391
dist1	Yaşar SÖKMENSÜER (7 yazı)	Doğan HIZLAN (50 yazı)	1.352837822290514
dist2		İlber ORTAYLI (50 yazı)	1.3490582969787153
dist3		Melike KARAKARTAL (50 yazı)	1.2731714424621436
dist4		Müge AKGÜN (50 yazı)	1.30187582753053
dist5		Vedat MİLOR (50 yazı)	1.3382886236248333
dist6		Yaşar SÖKMENSÜER (50 yazı)	1.2289827730630367

2.1.4.2. CountVektorizer ile Öklid Mesafesi Uygulaması

CountVectorizer da TfidfVectorizer gibi Sklearn kütüphanesinde olan bir öznitelik çıkarımıdır ve TfidfVectorizer'ın yerine yazılarak çalıştırılabilir. Bu kez CountVectorizer ile yazılarda (articles) bulunan tüm lemmalara (Şekil 2.15.) vektör yüklenmiştir ve bu lemmaların dizilişi sağlanmıştır (Şekil 2.17.). Bir önceki bölümde olduğu gibi Öklid mesafelerine bakmak için Çizelge 2.2.'deki yazı aralıklarından aynı aralıklar seçilmiştir ve Öklid mesafeleri çıkarılmıştır (Çizelge 2.4.). CountVectorizer'a göre yine MDS'de matris gösterimi sağlanmıştır. Bu gösterimde de renkler bir önceki adımla kıyaslanabilmesi için aynı tutulmuştur.

```
{'timsah': 25572, 'gözyaşı': 10061, 'güvencin': 10279, 'çözüm': 29371, 'öneri': 29593, 'daniel': 5480, 't
30700
(960, 30700)
[0 0 0 ... 0 0 0]
[0 0 0 ... 0 0 0]
[0 0 0 ... 0 0 0]
[0 0 0 ... 0 0 0]
[0 0 0 ... 0 0 0]
[0 0 0 ... 0 0 0]
[0 0 0 ... 0 0 0]
[0 0 0 ... 0 0 0]
[0 0 0 ... 0 0 0]
[0 0 0 ... 0 0 0]
[0 0 0 ... 0 0 0]
[0 0 0 ... 0 0 0]
[0 0 0 ... 0 0 0]
[0 0 0 ... 0 0 0]
[0 0 0 ... 0 0 0]
[0 0 0 ... 0 0 0]
[0 0 0 ... 0 0 0]
[0 0 0 ... 0 0 0]
[0 0 0 ... 0 0 0]
[0 0 0 ... 0 0 0]
[0 0 0 ... 0 0 0]
[0 0 0 ... 0 0 0]
```

Şekil 2.17. CountVectorizer ile lemmaları vektörlere çevirme ve dizilişleri

Çizelge 2.4. Seçilen yazıların CountVectorizer ile Öklid Mesafeleri.

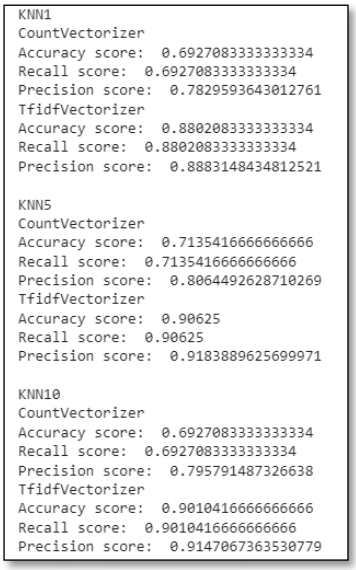
Mesafe Adları	Yazar	Karşılaştırılan Yazarlar	Öklid Mesafeleri
dist1	Doğan HIZLAN (7 yazı)	Doğan HIZLAN (50 yazı)	0.6584419124824873
dist2		İlber ORTAYLI (50 yazı)	0.9565346461143837
dist3		Melike KARAKARTAL (50 yazı)	0.953943426937654
dist4		Müge AKGÜN (50 yazı)	1.001099724368994
dist5		Vedat MİLOR (50 yazı)	1.068478417421163
dist6		Yaşar SÖKMENSÜER (50 yazı)	0.964409394317842
dist1	İlber ORTAYLI (7 yazı)	Doğan HIZLAN (50 yazı)	0.8686886640272696
dist2		İlber ORTAYLI (50 yazı)	0.5891578934981582
dist3		Melike KARAKARTAL (50 yazı)	0.8721517793097023
dist4		Müge AKGÜN (50 yazı)	0.9066328897233361
dist5		Vedat MİLOR (50 yazı)	0.9607237399348467
dist6		Yaşar SÖKMENSÜER (50 yazı)	0.9161989961481288
dist1	Melike KARAKARTAL (7 yazı)	Doğan HIZLAN (50 yazı)	0.9261470658853729
dist2		İlber ORTAYLI (50 yazı)	0.8619622444062732
dist3		Melike KARAKARTAL (50 yazı)	0.5452983121839339
dist4		Müge AKGÜN (50 yazı)	0.889537817179295
dist5		Vedat MİLOR (50 yazı)	0.919379518728025
dist6		Yaşar SÖKMENSÜER (50 yazı)	0.7290672905859356
dist1	Müge AKGÜN (7 yazı)	Doğan HIZLAN (50 yazı)	0.9141236839489495
dist2		İlber ORTAYLI (50 yazı)	0.893796105643497
dist3		Melike KARAKARTAL (50 yazı)	0.903514735535176
dist4		Müge AKGÜN (50 yazı)	0.5085161475033939
dist5		Vedat MİLOR (50 yazı)	0.7729316611556455
dist6		Yaşar SÖKMENSÜER (50 yazı)	0.9163809940244008
dist1	Vedat MİLOR (7 yazı)	Doğan HIZLAN (50 yazı)	0.904354782678015
dist2		İlber ORTAYLI (50 yazı)	0.8268970277902069
dist3		Melike KARAKARTAL (50 yazı)	0.7874501536966919
dist4		Müge AKGÜN (50 yazı)	0.7183567164139688
dist5		Vedat MİLOR (50 yazı)	0.5315507742430929
dist6		Yaşar SÖKMENSÜER (50 yazı)	0.7920349105438869
dist1	Yaşar SÖKMENSÜER (7 yazı)	Doğan HIZLAN (50 yazı)	1.033968192349704
dist2		İlber ORTAYLI (50 yazı)	0.9908149136393444
dist3		Melike KARAKARTAL (50 yazı)	0.9099613584504636
dist4		Müge AKGÜN (50 yazı)	0.9517281535940452
dist5		Vedat MİLOR (50 yazı)	1.0029692352667683
dist6		Yaşar SÖKMENSÜER (50 yazı)	0.8524768852984964

2.1.4.3. KNN Algoritması Uygulaması

Şekil 2.15.'te köşe yazıları ve bağlı olduğu etiketler test ve eğitim verisi olarak bölünmüştür. Yazıların %20'si test, %80'i eğitim verileri olarak alınmıştır (Şekil 2.18.). Yani algoritma yazıların %80'i ile eğitilip, diğer %20'si ile test edilecektir ve buna göre bir sonuca varılacaktır. Öncelikle en yakın 'k' komşu algoritması uygulanmıştır. Bu algoritma uygulanırken 'sklearn' kütüphanesinden faydalanılmıştır. Metinlerin hem CountVektorizer hem de TfidfVectorizer ile vektörleri bulunmuştur. Bu algoritmada 'k' değeri sırasıyla '1, 5, 10' olarak seçilmiştir. Dolayısıyla metinlerde en yakın vektör olarak çevresinde sırasıyla '1, 5, 10' vektöre bakılmış ona göre bir karar verilmiştir. Karar verme aşamasında CountVectorizer ve TfidfVectorizer için doğruluk (accuracy), hassasiyet (recall) ve kesinlik (precision) skorları çıkarılmıştır (Şekil 2.19.). Daha sonra 20'ye kadar olan her 'k' değeri ve iki Vectorizer için hata oranına ve doğruluk skoruna bakılmıştır (Şekil 3.3.).

```
1 from sklearn.model_selection import train_test_split
2 from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
3
4 X_train, X_test, y_train, y_test = train_test_split(articles, labels, random_state=1, test_size=0.20)
5
6 cv = CountVectorizer()
7 X_train_cv = cv.fit_transform(X_train)
8 X_test_cv = cv.transform(X_test)
9
10 tv = TfidfVectorizer()
11 X_train_tv = tv.fit_transform(X_train)
12 X_test_tv = tv.transform(X_test)
```

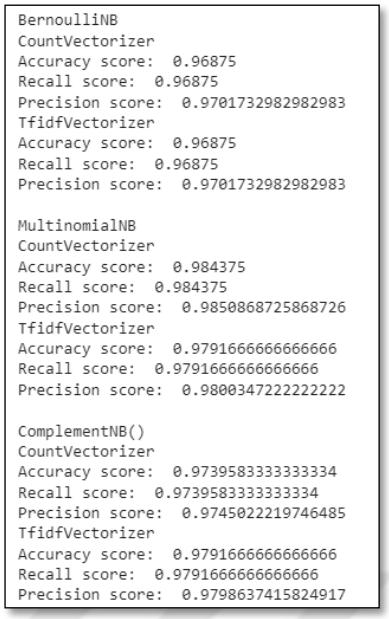
Şekil 2.18. Metinlerin eğitim ve test verilerine bölünmesi; CountVectorizer/TfidfVectorizer uygulanması.



Şekil 2.19. KNN Skorları.

2.1.4.4. Naive Bayes Algoritması Uygulaması

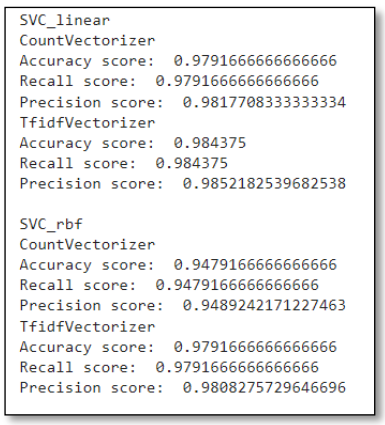
Şekil 2.18.'de bölünmüş veriler bu kez Naive Bayes (NB) algoritmasına sokulmuştur. Bu algoritma kullanılırken sklearn kütüphanesinden Katlıterim (Multinomial) NB, Bernoulli NB, Tümleyici (Complement) NB sınıflarından yararlanılmıştır. Kategorik ve Gaussian NB sınıfları kodda hata verdiği kullanılamamıştır. Daha sonra KNN algoritmasında olduğu gibi CountVectorizer ve TfidfVectorizer için ve seçilen Naive Bayes sınıfları için doğruluk (accuracy), hassasiyet (recall) ve kesinlik (precision) skorları çıkarılmış (Şekil 2.20.), en son da kaç hata olduğunda dair hata matrisine bakılmıştır (Şekil 3.4.).



Şekil 2.20. Naive Bayes Skorları.

2.1.4.5. SVM Algoritması Uygulaması

Eğitim ve test olarak ayrılan veriler (Şekil 2.18.) sklearn kütüphanesinden SVM algoritmasına sokulmuştur. SVC olan sınıflandırıcısıyla çekirdek (kernel) parametresi hem doğrusal (linear) hem de varsayılan olan radyal temelli fonksiyon (rbf) seçilmiştir. Bu her iki parametrede CountVectorizer ve TfidfVectorizer için sınıflandırıcının doğruluk (accuracy), hassasiyet (recall) ve kesinlik (precision) skorlarına bakılmıştır (Şekil 2.21). Daha sonra hata matrisi düzenlenmiştir (Şekil 3.5.).



Şekil 2.21. SVM Skorları.

2.1.4.6. Decision Tree Classifier ve MLP Algoritmaları Uygulamaları

Son olarak Şekil 2.18.'deki veriler sklearn kütüphanesinden MLP ve DecisionTree algoritmalarına sokulmuştur. Her iki algoritmada da CountVectorizer ve TfidfVectorizer kullanılmış, daha sonra doğruluk (accuracy), hassasiyet (recall) ve kesinlik (precision) skorlarına bakılmıştır (Şekil 2.22.).

```
MLP
CountVectorizer
Accuracy score:  0.984375
Recall score:  0.984375
Precision score:  0.9845629691007295
TfidfVectorizer
Accuracy score:  0.9895833333333334
Recall score:  0.9895833333333334
Precision score:  0.9899340628507295

DecisionTree
CountVectorizer
Accuracy score:  0.796875
Recall score:  0.796875
Precision score:  0.8058659527029203
TfidfVectorizer
Accuracy score:  0.7447916666666666
Recall score:  0.7447916666666666
Precision score:  0.7507060162591772
```

Şekil 2.22. MLP ve DecisionTree Skorları.

2.1.4.7. LSTM Algoritması Uygulaması

Bir derin öğrenme algoritması olan LSTM algoritması kullanılırken ise tensorflow, numpy ve keras kütüphanelerinden yararlanılmıştır. Öncelikle LSTM için zorunlu bir adım olan Tokenizer (Simgeleştirici) sınıfı metinleri tamsayılara çevirmek için getirilmiştir. Daha sonra Ardışık (Sequential) modeli içerisinden Dense (Yoğun), Flatten (Düzleştirme), LSTM, Dropout (Bırakma), Activation (Aktivasyon), Embedding (Gömme), Bidirectional (Çift Yönlü) katmanları getirilmiştir. En iyileştirici (Optimizer) olarak da Adam seçilmiştir. Kelime büyüklüğü (vocab_size) 5000 olarak seçilmiş, veri seti içerisinde en yaygın kullanılan 5000 kelime hedeflenmiştir. Gömme boyutu (embedding_dim) 256 seçilmiş, köşe yazısı başına düşen ortalama lemma adedi (max_length) 500 seçilmiştir. Metinleri kesme (truncation) ve doldurma (padding) işlemi sondan (post)

olacak şekilde seçilmiştir. Kelime listesi dışındaki simgeler "OOV" şeklinde gösterilecektir. Veri setinin %80'i eğitim verisine verilip eğitilmiştir (training_portion). Bırakma oranı (dropout_rate) ise %90 seçilmiştir, böylece veri seti sürekli incelendiğinde veri setinin %90'ı her defasında bırakılacaktır ve aynı veriler üzerinde çalışılmayacaktır. 6 yazar olduğu için de sınıf sayısı da (num_classes) 6 olarak not edilmiştir. Verilerin başta not edilmesinin amacı LSTM verileri üzerinde değişiklik yapılacaksa tek bir blokta yapılmasını sağlamaktır (Şekil 2.23.).

Etiketler (labels) bağlı oldukları yazılar (articles) ile değiştirilmeden birbiriyle karıştırılmıştır, böylece LSTM algoritması uygulandığında yazarların sırası her 160 yazıda sabit olarak kalmayacaktır. Karıştırılan yazılar eğitim (%80) ve test verileri (%20) olarak bölünmüştür. Eğitim büyüklüğü (train_size) yazıların %80'ini miktarına kadar eğitim makaleleri (train_articles), %80'ininden sonraki miktar ise test makaleleri (test_articles) olarak belirlenmiştir. Aynı durum eğitim ve test yazılarının bağlı olduğu etiketler (yazar isimleri) için de gerçekleştirilmiştir. Şekil 2.24.'de eğitim kümesinin boyutu 768, test kümesinin boyutu da 192 olarak çıkmıştır. Ayrıca kontrol amaçlı eğitim makalelerinin ve eğitim etiketlerinin ilki (0'ıncısı) da yazdırılmıştır.

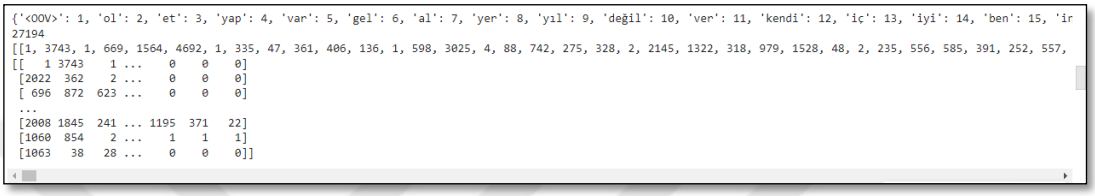
```python
import tensorflow as tf
import numpy as np
import random
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, LSTM, Dropout, Activation, Embedding, Bidirectional
from tensorflow.keras.optimizers import Adam
vocab_size = 5000
embedding_dim = 256
max_length = 500 #ortalama lemma adedi
trunc_type = "post" #pre
padding_type = "post"
oov_tok = "<OOV>"
training_portion = 0.8
dropout_rate = 0.9
num_classes = 6
```

Şekil 2.23. LSTM Algoritmasının kurulumu ve yazılan değerler.

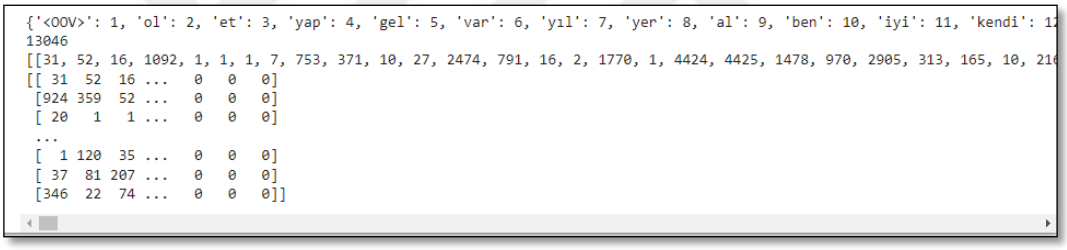
```
eğitim kümesinin boyutu 768
test kümesinin boyutu 192
timsah gözyaşı güvencin çözüm öneri daniel boulud yıldız şef ortak özellik artık çokuluslu şirket ceo yap lokanta gastronomik açı ilginç ol ağla bebek anne süt ka
vedatmilor
```

Şekil 2.24. Eğitim ve test kümelerinin boyutu; eğitim kümesinin ilk öğesinin ve yazarının çıktısı.

Şekil 2.25. ve Şekil 2.26'da en sık geçen 5000 kelime ve bunlar dışındaki kelimeler (OOV) eğitim ve test makalelerinde işaretlenmiş ve sözcük dizinleri (word_index) verilmiştir. Ayrıca bu dizinlerin dizilişi doldurma, kesme ve maksimum uzunluk değerlerine (Şekil 2.15.) göre çıkarılmıştır. Aynı durum eğitim ve test yazılarının bağlı olduğu eğitim ve test etiketleri (yazarlar) için de yapılmıştır; hepsine birer dizin atanmıştır (Şekil 2.27.; Şekil 2.28.). 6 yazar olsa da ilk dizin [0] ile son dizin [5] ile gösterilmiştir.

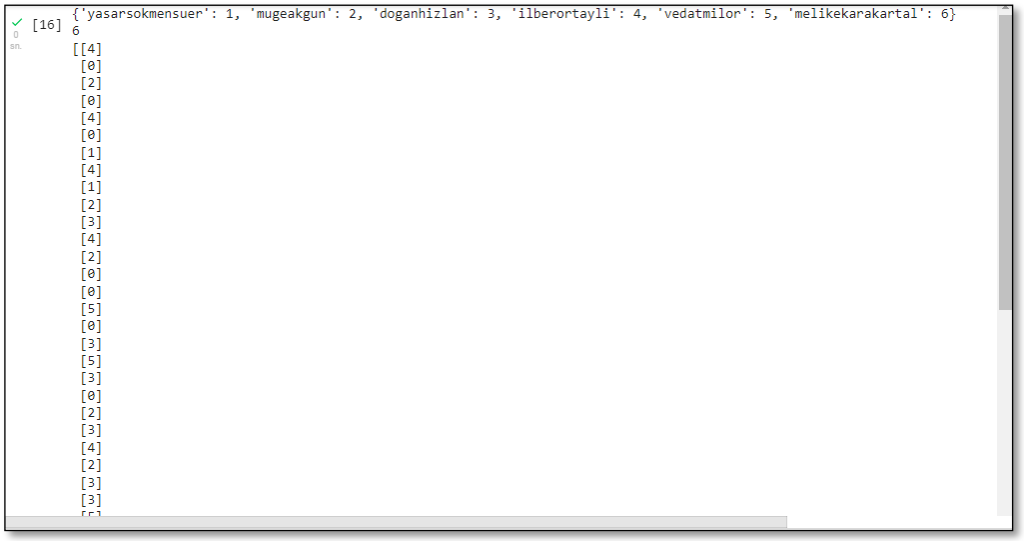


Şekil 2.25. Eğitim makalelerinde sözcük dizini atanması ve bu dizinin diziliş çıktısı.



Şekil 2.26. Test makalelerinde sözcük dizini atanması ve bu dizinin diziliş çıktısı.

Şekil 2.27. Eğitim makalelerinin bağlı olduğu eğitim etiketlerine dizin atanması ve sıralanması.



Şekil 2.28. Test makalelerinin bağlı olduğu test etiketlerine dizin atanması ve sıralanması.

Ardışık (Sequential) model uygulanırken 4 katman değerleri verilmiştir. Bu katmanlar "add()" fonksiyonuyla getirilmiştir. Gömme (Embedding) katmanında 5000 kelime için gömme boyutu (embedding_dim) 256 olarak seçilmiştir. Bırakma (Dropout) katmanında bırakma oranı (%90) verilmiş, çift yönlü katman 256 gömme boyutuyla eklenmiştir. Yoğun (Dense) katmanda ise 6 yazar olduğundan sınıf sayısı (num_classes) 6 seçilmiş, aktivasyon ise olasılık dağılımının görülmesi için "softmax" fonksiyonu getirilmiştir. Katmanların eklendiği model özetlenince "model.summary()" çıktı şekli (output shape) ve parametreler Çizelge 2.5.'te gösterilmiştir. Gömme (Embedding) katmanında 256 boyutta eğitilen parametreler 1.280.000, bırakma (dropout) katmanında 0 olarak not edilmiştir. Çift yönlü (Bidirectional) katmanda ise girdi hem sağdan sola hem de soldan sağa okunduğu için 256 boyutu ikiye katlanmış, 512 boyutta 1.050.624 parametre eğitilmiştir. Yoğun (Dense) katmanda da 6 sınıf için 3078 parametre eğitilmiştir. Toplamda 2.333.702 parametre eğitilmiş, eğitilmeyen parametre bulunmamaktadır.

Çizelge 2.5. Ardışık modelin eklenen katmanlar sonrası çıkan özeti.

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_4 (Embedding)     (None, None, 256)         1280000

 dropout_4 (Dropout)         (None, None, 256)         0

 bidirectional_4 (Bidirectio  (None, 512)              1050624
 nal)

 dense_4 (Dense)             (None, 6)                 3078

=================================================================
Total params: 2,333,702
Trainable params: 2,333,702
Non-trainable params: 0
_________________________________________________________________
```

Optimizer olarak Adam kullanılmıştır ve Adam'da öğrenme hızı (learning_rate) varsayılan değer 0.001 olarak seçilmiş, bozunma oranı (decay) ise varsayılan değere yakın '1e-6' (0,0000001) olarak kullanılmıştır. Model derlenirken de kayıp (loss) fonsiyonu 2'den fazla sınıf olduğu için "sparse_categorical_crossentropy" (seyrek kategorik çapraz etkinlik ölçümü) olarak seçilmiş, metrik doğruluk (accuracy) olarak getirilmiştir.

Geçmiş nesnesi üzerinde "model.fit()" yapılırken x değişkeni eğitim verisinin dizilişi (train_padded), y değişkeni eğitim verilerinin bağlı olduğu etiket verilerinin dizilişidir. Devir sayısı (num_epochs) 20 olarak belirlenmiştir, yani veri 20 kez eğitilecektir. Doğrulama verisi (validation_data) ise test verisinin dizilişi ve bağlı olduğu etiket sırası olarak verilmiştir. Ayrıntı (Verbose), 2 değerinde getirilmiştir, böylece çıktıda devir başına bir satır gösterimi sağlanmıştır.

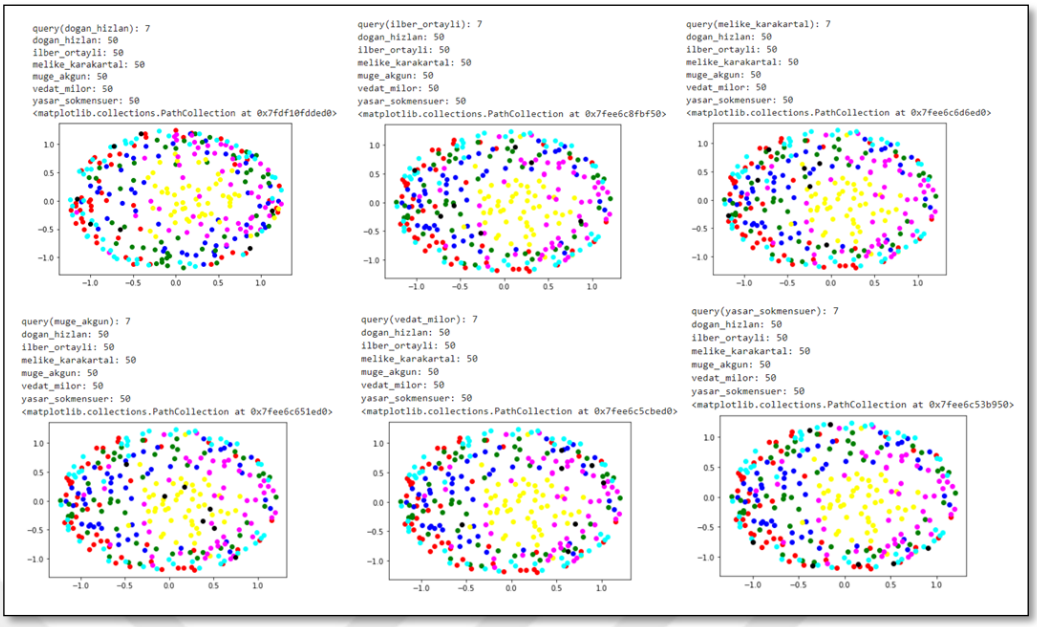
3. BULGULAR

3.1. Verilerin Analizi

Bu bölümde uygulanan tüm algoritma ve parametrelerin analizleri ve yorumları yapılmıştır. Hepsinden çıkan sonuçlarla bir genelleme yapılmaya çalışılmıştır.

3.1.1. TF-IDF Vektorizer ile Kosinüs Benzerliği Analizi

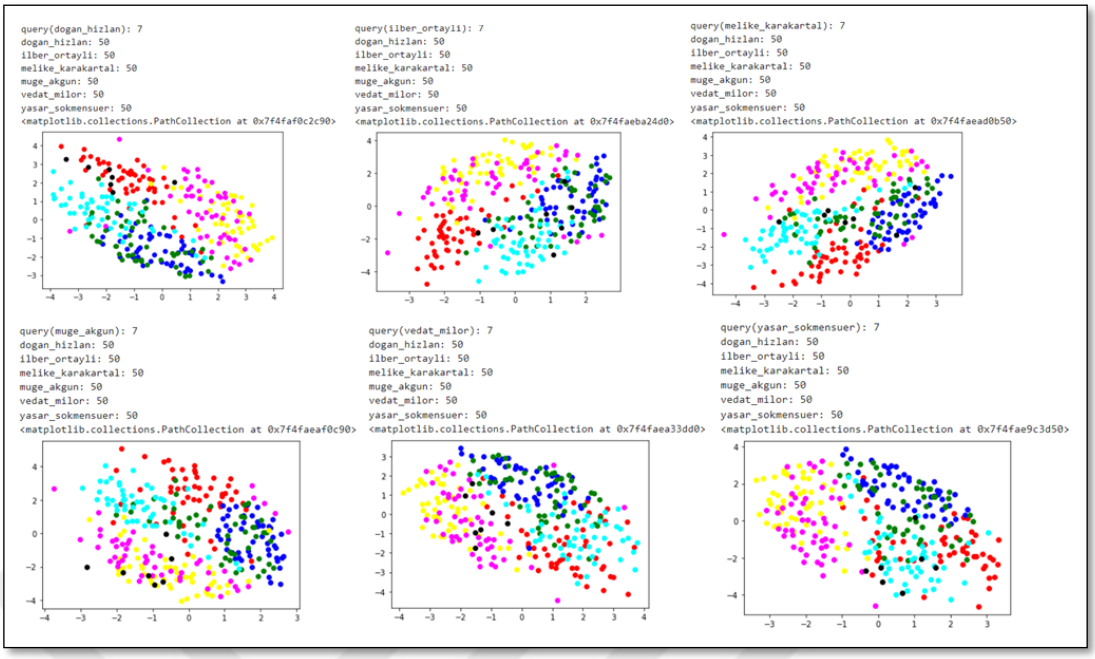
Çizelge 2.3.'te TfidfVectorizer ile metinler arasında açısal mesafelere bakıldığında yazar kendi yazıları ile kıyaslandığında açısal mesafenin, diğer yazarların yazılarına göre daha düşük olduğu gözlemlenmiştir. Bu durum tüm yazarlar için aynıdır. O halde her bir sorgu yazısının her bir yazarın yazılarıyla kıyaslanması ile yazıların düşük olan açısal mesafeleri için bu yazılar birbirine yakındır çıkarımı yapılabilir. Şekil 3.1.'e bakıldığında her bir yazarın hem kendisi hem de diğer yazarlarla karşılaştırılmasıyla görsel bir çıktı elde edilmiştir. Buna göre yazılar arası ayırt edici bir durum söz konusu değildir çünkü açısal mesafeler düşük olsa da diğer yazarların yazılarıyla olan açısal mesafeler arasında keskin bir değer görülmemektedir.



Şekil 3.1. Yazılar arası TfidfVectorizer ile açısal mesafelerin görsel matrisi.

3.1.2. CountVektorizer ile Öklid Mesafesi Analizi

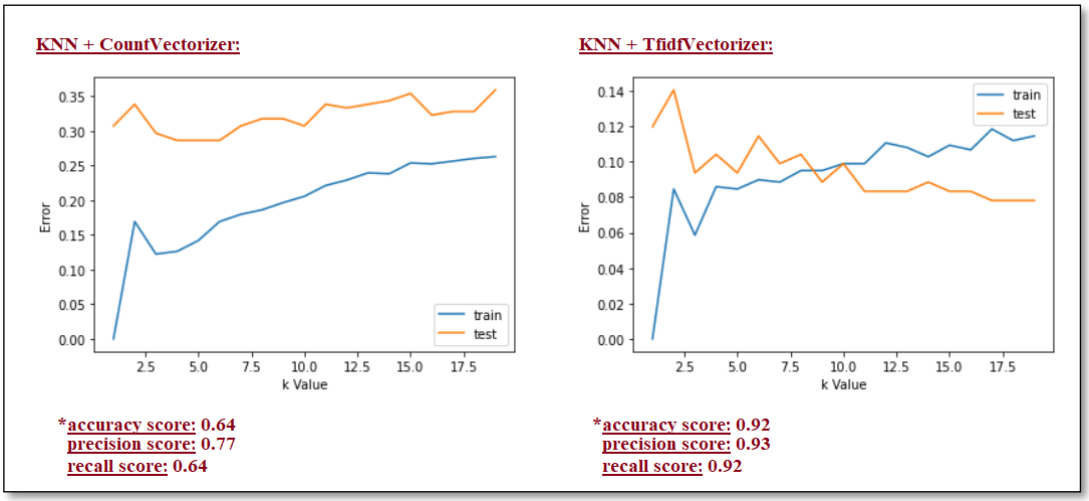
Bir önceki adımda olduğu gibi bu kez Çizelge 2.4.'de CountVectorizer ile yazıların Öklid mesafelerine bakılmıştır. Öklid mesafelerinin açısal mesafelere göre daha düşük olduğu görülmektedir. Her yazarın kendi yazılarıyla olan Öklid mesafeleri diğer yazarların yazılarına göre burada da daha düşük olmuştur. Aynı olgu TfidfVectorier ile açısal mesafelerde de yer aldığı için tesadüfi bir durum olarak görülmeyebilir. Burdan hareketle yazılar arasında Öklid ve açısal mesafeler doğru orantılı olarak düşük olduğundan, söz konusu yazılar birbirine benzer çıkarımı yapılabilir. Şekil 3.2.'de Öklid mesafelerin görsel matrisi bulunmaktadır. Nokta dağılımları bu matriste Şekil 3.1.'deki matrise göre bir nebze daha ayırt edici olabilir çünkü noktalar çok belirgin olmasa da belli bölgelerde yığılmışlardır.



Şekil 3.2. Yazılar arası CountVectorizer ile Öklid mesafelerin görsel matrisi.

3.1.3. KNN Algoritması Analizi

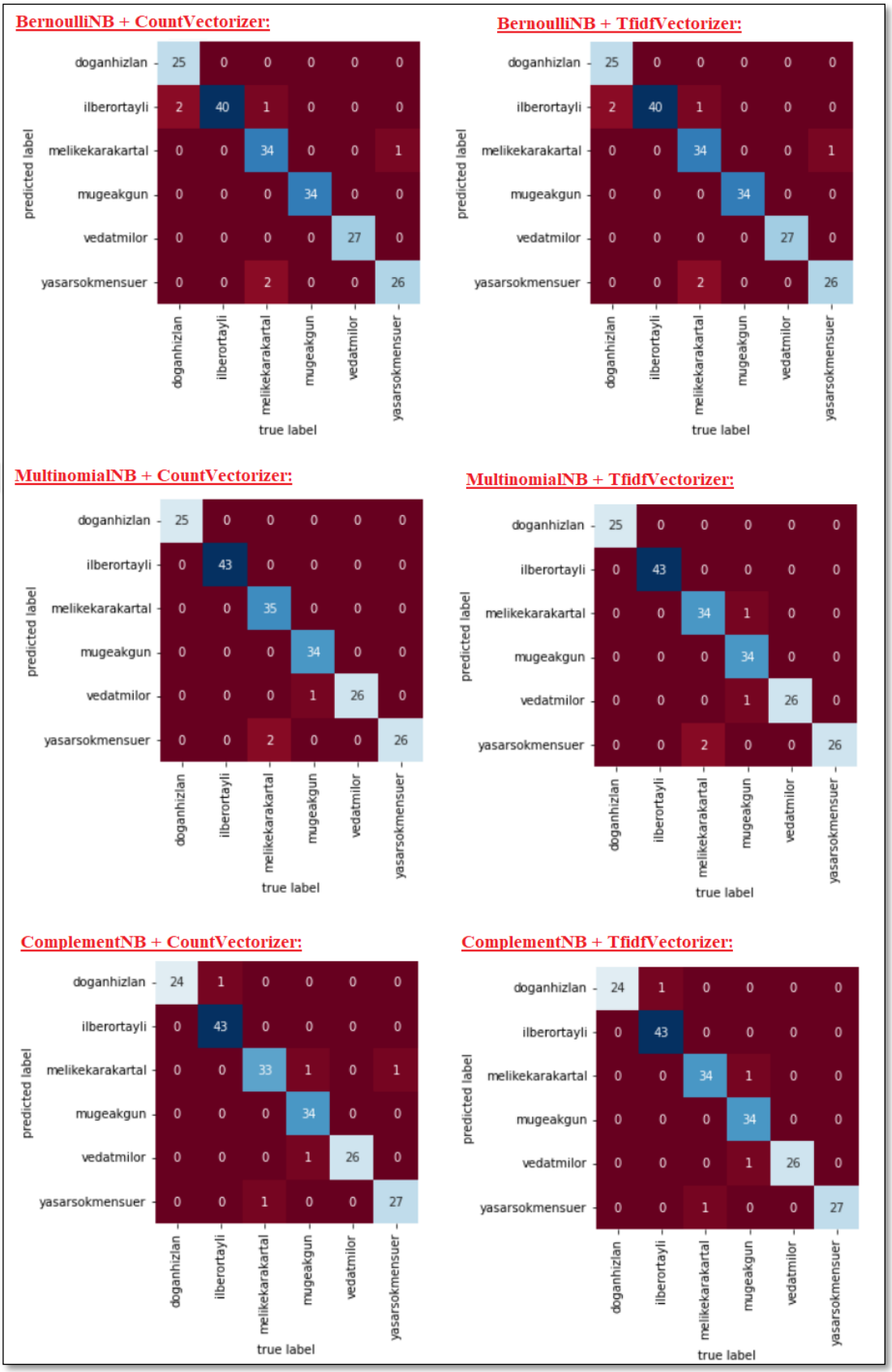
Şekil 2.19.'daki KNN algoritması skorlarına göre, "k=1, 5, 10" olduğunda TfidfVectorizer ile çıkarılan vektörlerde doğruluk, hassasiyet ve kesinlik skoru CountVectorizer ile çıkarılan vektörlerin skorlarına oranla daha yüksek çıkmıştır. O halde TfidfVectorizer ile KNN algoritması daha iyi performans göstermiştir denebilir. Ayrıca "k" değeri arttıkça bu skorlar da doğru orantılı olarak artmıştır. Şekil 3.3.'e göre "k" değeri 1'den 20'ye kadar arttıkça eğitim ve test verisindeki hata minimum oranlarda görülmektedir. Test verisindeki hata oranı TfidfVectorizer'da gittikçe azalırken, CountVectorizer'da bu hata oranı %30- %35 aralığındadır. Bu grafiğe göre doğruluk, hassasiyet ve kesinlik skorları da belirlenmiş, bütün bu skorlar CountVectorizer'a göre TfidfVectorizer'da fazla olduğu tespit edilmiştir.



Şekil 3.3. CountVectorizer/TfidfVectorizer ile k-değeri için hata grafiği ve skorları.

3.1.4. Naive Bayes Algoritması Analizi

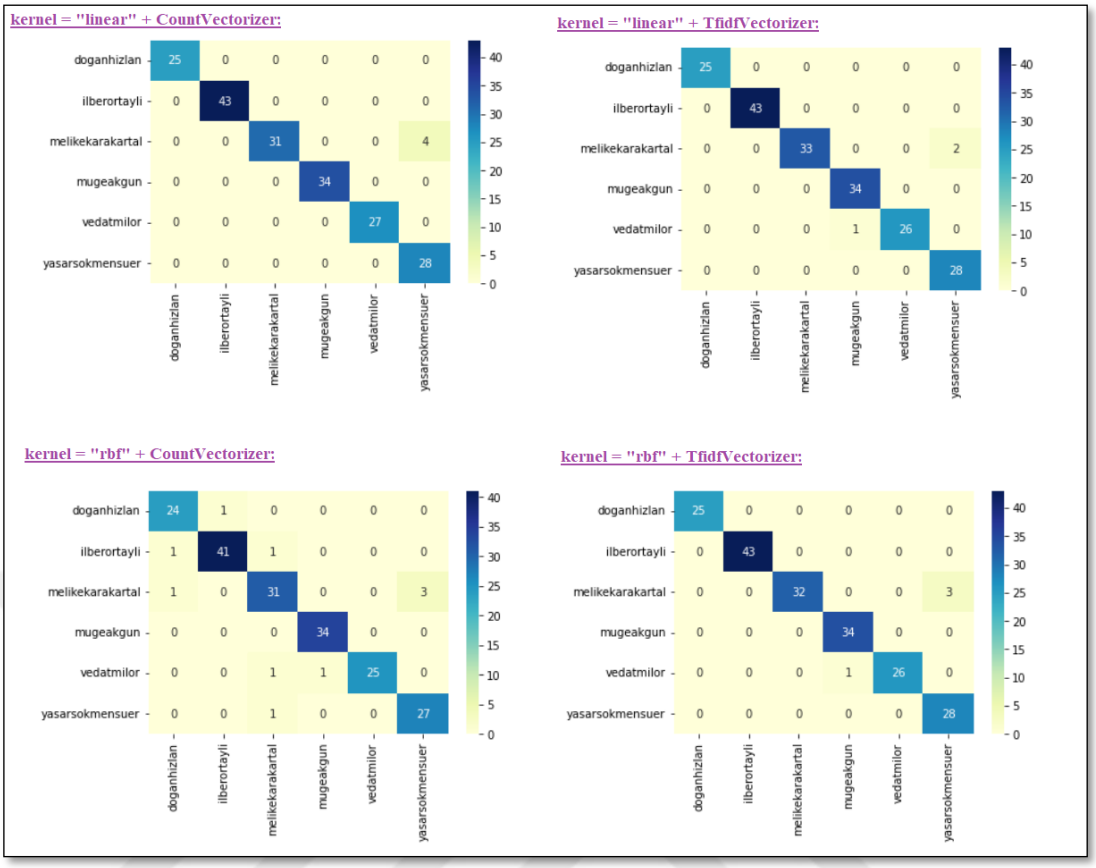
Şekil 2.20.'deki skorlara göre BernoulliNB ve ComplementNB sınıflarında CountVectorizer ve TfidfVectorizer doğruluk, hassasiyet ve kesinlik skorları birbirine oldukça yakındır ve yüksek çıkmıştır. MultinomialNB sınıfında ise CountVectorizer skorları TfidfVectorizer skorlarından daha yüksek çıkmıştır. MultinomialNB sınıfında CountVectorizer ile çalışmak daha iyi bir fikir olabilir. Algoritma skorları sonucunda 3 Naive Bayes (BernoulliNB, MultinomialNB, ComplementNB) sınıfı dahil olmak üzere hem CountVectorizer hem de TfidfVectorizer için Şekil 3.4.'de hata matrisi düzenlenmiştir. Buna göre MultinomialNB sınıfı CountVectorizer ile test verisi (192 veri) üzerinde en az hatayı (3 hata) yapmıştır. Tfidfvectorizer ile MultinomialNB ve ComplementNB sınıfı test verisi üzerinde aynı ölçüde tahmine sahiptir. BernoulliNB sınıfında ise diğer sınıflara göre hata sayısı artmıştır. Sonuç olarak bu matris tüm sınıfların çok az hata ile gayet iyi bir tahmine sahip olduklarını göstermektedir. O halde bu algoritma başarılı olmuştur denebilir.



Şekil 3.4. Naive Bayes hata matrisi.

3.1.5. SVM Algoritması Analizi

Şekil 2.21.'deki skorlara bakıldığında, SVC çekirdeği doğrusal (kernel="linear") seçildiğinde çekirdeğin varsayılan değerinden (kernel="rbf") daha yüksek bir doğruluk, hassasiyet ve kesinlik skoru vermektedir. Ayrıca her iki çekirdek fonksiyonunda da CountVectorizer ve TfidfVectorizer kullanımı skorlar açısından büyük bir fark yaratmamıştır. İki çekirdek fonksiyonunda da TfidfVectorizer skorları daha yüksek çıkmıştır. SVM sonucunda çekirdek (kernel) fonksiyonu için 2, Vectorizer'lar için 2 olmak üzere 4 farklı faktör için hata matrisine bakılmıştır (Şekil 3.5.). TfidfVectorizer ile çekirdek "linear" veya "rbf" olarak seçildiğinde test verisinde daha az veri yanlış tahmin edilmiştir, yani CountVectorizer'a göre TfidfVectorizer daha iyi bir sonuç vermiştir. Çekirdeğin "linear" olarak seçilmesi, "rbf" olarak seçilmesinden daha iyi bir sonuç çıkarmıştır çünkü test verisinde TfidfVectorizer için 'kernel="linear"' olması sadece 3 veriyi yanlış; 189 veriyi doğru tahmin etmiştir.



Şekil 3.5. SVM hata matrisi.

3.1.6. Decision Tree Classifier ve MLP Algoritmaları Analizleri

Şekil 2.22.'deki skorlar incelendiğinde Karar Ağaçları sınıflandırıcısı CountVectorizer için TfidfVectorizer'dan daha yüksek doğruluk, hassasiyet ve kesinlik skoru vermiştir. MLP algoritması ise her iki Vectorizer için birbirine yakın doğruluk, hassasiyet ve kesinlik skoru vermiştir. Karar Ağaçlarına göre MLP skorları oldukça yüksektir. Buna göre MLP algoritması Karar Ağaçları sınıflandırıcısından çok daha iyi çalışmıştır.

Uygulanan bütün sklearn algoritmalarına bakıldığında performansın bu veri seti için oldukça yüksek olduğu söylenebilir (Çizelge 3.1.). "random_state"in 1'den 11'e kadar seçilmesi ile doğruluk, hassasiyet ve kesinlik skorları 10 kez yenilenmiştir ve her bir algoritma ve varsa fonksiyonları için bu 10 değerin ortalaması alınmıştır. Böylece kullanılan tüm sklearn algoritmalarının nihai doğruluk, hassasiyet ve

kesinlik skorları çıkarılmıştır. Bu skorlar incelendiğinde doğruluk ve hassasiyet skorlarının birbirine eşit ve kesinlik skorlarına oldukça yakın oldukları saptanmıştır. Doğruluk/hassasiyet ve kesinlik skorları en yüksek MLP algoritmasında çıkmıştır. Daha sonra SVM algoritmasının doğrusal fonksiyonu 2. yüksek skorlara sahiptir. 3. yüksek skorlar ise Naive Bayes algoritması Bernoulli sınıfı tarafından çıkarılmıştır. Algoritmalar arasından en düşük skorlar ise KNN ve Karar Ağaçları algoritmaları sonucunda çıkmıştır. KNN algoritmasında yine "k" değeri arttıkça skorlar da artmıştır ve diğer algoritmalara nazaran kesinlik skorları doğruluk/hassasiyet skorlarından oldukça yüksektir. İki farklı vectorizer kullanımı da skorları etkileyen bir diğer unsurdur. TfidfVectorizer KNN, SVM ve MLP algoritmalarında daha yüksek skorlara sahipken, CountVectorizer Naive Bayes ve Karar Ağacı algoritmalarında daha yüksek skorlara sahip olmuştur.

Çizelge 3.1. Sklearn algoritmalarının toplam ortalama skorları.

Algoritmalar		Doğruluk/Hassasiyet Skoru		Kesinlik Skoru	
		TfidfVectorizer	CountVectorizer	TfidfVectorizer	CountVectorizer
KNN	n_neighbors = 1	0.86	0.68	0.87	0.77
	n_neighbors = 5	0.87	0.72	0.88	0.8006
	n_neighbors = 10	0.88	0.72	0.89	0.8048
Naive Bayes	BernoulliNB	0.9739	0.9739	0.9764	0.9764
	MultinomialNB	0.93	0.9791	0.94	0.9799
	ComplementNB	0.9604	0.9729	0.9624	0.9744
SVM	kernel = "linear"	0.9817	0.9770	0.9827	0.9785
	kernel = "rbf"	0.975	0.9578	0.9769	0.9598
Decision Tree		0.7604	0.78	0.7693	0.79
MLP		0.9890	0.9807	0.9895	0.9816

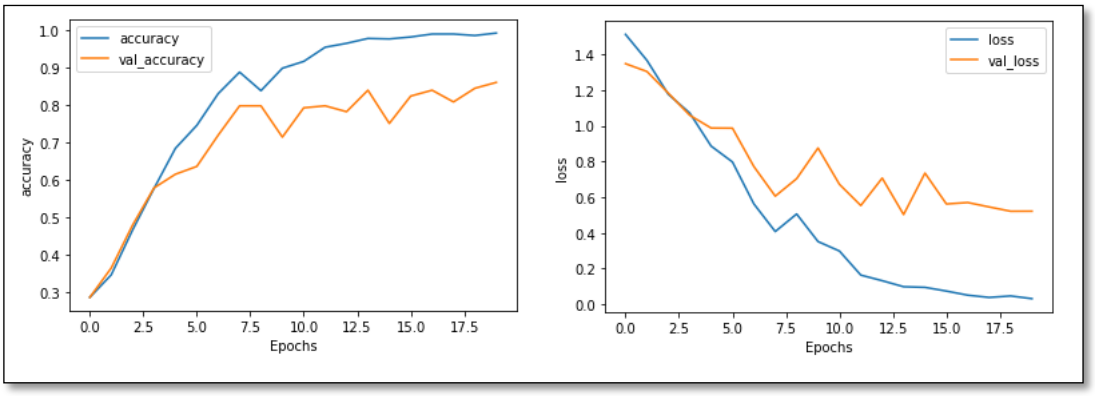
3.1.7. LSTM Algoritması Analizi

LSTM algoritması sonucundaki çıktıya bakıldığında (Şekil 3.6.), 20 devir sonunda dereceli olarak, kayıp (loss) %3'e yakın olacak şekilde azalmış, doğruluk (accuracy) %99'a kadar artmış, doğrulama verisi kaybı (val_loss) %50'ye kadar azalmış ve doğrulama verisinin doğruluğu %85'e kadar artmıştır. Bu da LSTM algoritmasının veri seti üzerinde oldukça iyi çalıştığını göstermektedir. Şekil

3.7.'deki grafiğe bakıldığında doğruluk arttıkça, doğrulama verisinin doğruluğu da artmış; kayıp azaldıkça da doğrulama verisinin kaybı azalmaktadır.

```
Epoch 1/20
24/24 - 11s - loss: 1.5110 - accuracy: 0.2865 - val_loss: 1.3472 - val_accuracy: 0.2865 - 11s/epoch - 472ms/step
Epoch 2/20
24/24 - 7s - loss: 1.3659 - accuracy: 0.3464 - val_loss: 1.3034 - val_accuracy: 0.3646 - 7s/epoch - 307ms/step
Epoch 3/20
24/24 - 7s - loss: 1.1761 - accuracy: 0.4661 - val_loss: 1.1823 - val_accuracy: 0.4792 - 7s/epoch - 311ms/step
Epoch 4/20
24/24 - 7s - loss: 1.0713 - accuracy: 0.5768 - val_loss: 1.0587 - val_accuracy: 0.5781 - 7s/epoch - 306ms/step
Epoch 5/20
24/24 - 7s - loss: 0.8862 - accuracy: 0.6836 - val_loss: 0.9870 - val_accuracy: 0.6146 - 7s/epoch - 312ms/step
Epoch 6/20
24/24 - 7s - loss: 0.7970 - accuracy: 0.7448 - val_loss: 0.9863 - val_accuracy: 0.6354 - 7s/epoch - 307ms/step
Epoch 7/20
24/24 - 7s - loss: 0.5625 - accuracy: 0.8294 - val_loss: 0.7717 - val_accuracy: 0.7188 - 7s/epoch - 311ms/step
Epoch 8/20
24/24 - 7s - loss: 0.4077 - accuracy: 0.8867 - val_loss: 0.6059 - val_accuracy: 0.7969 - 7s/epoch - 312ms/step
Epoch 9/20
24/24 - 7s - loss: 0.5060 - accuracy: 0.8372 - val_loss: 0.7036 - val_accuracy: 0.7969 - 7s/epoch - 306ms/step
Epoch 10/20
24/24 - 7s - loss: 0.3517 - accuracy: 0.8971 - val_loss: 0.8752 - val_accuracy: 0.7135 - 7s/epoch - 307ms/step
Epoch 11/20
24/24 - 7s - loss: 0.2987 - accuracy: 0.9154 - val_loss: 0.6720 - val_accuracy: 0.7917 - 7s/epoch - 311ms/step
Epoch 12/20
24/24 - 7s - loss: 0.1640 - accuracy: 0.9531 - val_loss: 0.5533 - val_accuracy: 0.7969 - 7s/epoch - 307ms/step
Epoch 13/20
24/24 - 7s - loss: 0.1326 - accuracy: 0.9635 - val_loss: 0.7067 - val_accuracy: 0.7812 - 7s/epoch - 305ms/step
Epoch 14/20
24/24 - 7s - loss: 0.0989 - accuracy: 0.9766 - val_loss: 0.5026 - val_accuracy: 0.8385 - 7s/epoch - 306ms/step
Epoch 15/20
24/24 - 7s - loss: 0.0954 - accuracy: 0.9753 - val_loss: 0.7347 - val_accuracy: 0.7500 - 7s/epoch - 305ms/step
Epoch 16/20
24/24 - 7s - loss: 0.0746 - accuracy: 0.9805 - val_loss: 0.5621 - val_accuracy: 0.8229 - 7s/epoch - 311ms/step
Epoch 17/20
24/24 - 7s - loss: 0.0516 - accuracy: 0.9883 - val_loss: 0.5700 - val_accuracy: 0.8385 - 7s/epoch - 306ms/step
Epoch 18/20
24/24 - 7s - loss: 0.0389 - accuracy: 0.9883 - val_loss: 0.5453 - val_accuracy: 0.8073 - 7s/epoch - 311ms/step
Epoch 19/20
24/24 - 7s - loss: 0.0473 - accuracy: 0.9844 - val_loss: 0.5214 - val_accuracy: 0.8438 - 7s/epoch - 305ms/step
Epoch 20/20
24/24 - 7s - loss: 0.0321 - accuracy: 0.9909 - val_loss: 0.5216 - val_accuracy: 0.8594 - 7s/epoch - 306ms/step
```

Şekil 3.6. LSTM Algoritması sonucu çıktısı.



Şekil 3.7. LSTM Algoritması sonucu çıktısının grafik gösterimi.

4. TARTIŞMA

Bu çalışmanın bulgularında her sklearn algoritmasının sonucu doğruluk, hassasiyet ve kesinlik skorlarının fazla olması algoritmanın ezbere sonuç verip vermediği hususunda şüphe doğurmuştur. O nedenle her algoritma 10 kere denenip 10 sonucun ortalamasıyla tekrar değerlendirilmiştir. Ancak yine de elde edilen sonuçlar oldukça yüksektir. O yüzden derin öğrenme algoritması olan LSTM algoritması kullanılmıştır. LSTM algoritması başarılı görünse de algoritmaya verilen veriler yani 6 yazarın 160'ar yazı veri seti bir derin öğrenme algoritması için yeterli görünmeyebilir.

Çalışmanın ilk adımlarından olan önişlemde ise Türkçe dili birçok engele sebep olmaktadır. İngilizce dili için bulunan Python kütüphaneleri Türkçe dili için yeterince zengin değillerdir. Türkçe'de kelimeleri kök haline getirme, durduran ifadeleri çıkarma, ön ek ve son ek çıkarma işlemleri veri setinde %100 başarılı çalışmaması durumu algoritmalarda çıkan sonuçları da etkilemiş olabilir.

5. SONUÇ VE ÖNERİLER

Bu çalışmanın sonucunda seçilen 6 yazarın da yazıları test verileri üzerinden sorgulandığında, doğru yazar ile eşleştiği görülmektedir. Bu eşleşme yapılan önişlem adımlarına, yazıların vektörlerine ayrılıp algoritmalara sokulmasıyla elde edilmiştir. En başarılı algoritma %98 skorlarıyla MLP iken LSTM algoritmasında doğrulama verisinin skoru %85 olarak not edilmiştir. Genel olarak bakılacak olursa elde edilen ML algoritma skorlarının test verisi için oldukça yüksek olması çalışmayı başarılı kılmaktadır.

Kurulan bu yapı ile bu çalışma daha da genişletilebilir ve her koşulda test edilebilir bir konumdadır. Yeni bir yazar ve yazıları eklendiğinde de çalışabilirliğini sürdürmesi beklenen bir yapı oluşturulmuştur. O halde bu yapı ile yazıların hangi yazara ait olabileceği bulunabilirken intihali önleme hususuna katkı sağlayacağı savunulmaktadır. Üstelik sadece intihal değil ayrıca bir takım siber suçları engellemede de yardımcı olabilir. Bunun nedeni yazılan bir yazının o yazara ait olup olmadığı gösteren bir sistem olmasıdır. Dolayısıyla Bilişim sistemleri kullanımı içerip özellikle Türkçe dilinde Adli konular üzerinde hizmet edeceği düşünülmekte ve disiplinler arası Adli Bilişim bölümüne katkı sağlaması beklenmektedir.

ÖZET

Köşe Yazılarında Yazarlık Analizinin Adli Bilişim'e Katkısı

Bu araştırma, farklı konularda 6 yazarın 160'ar köşe yazısı üzerinden yazarlık analizini kapsamaktadır. Her yazarın kendine özgü yazı alışkanlığı olduğunu gösteren bu çalışma öncelikle Doğal Dil İşleme yöntemiyle tüm yazıların önişlemden geçirilmesini ve daha sonra derin öğrenme de olmak üzere makine öğrenme algoritmalarına sokulmasını içermektedir. Ayrılan eğitim ve test verileri üzerinden yazıların yazarlara ait olup olmadığı tartışılmış, algoritmalar sonucu çıkan doğruluk, hassasiyet ve kesinlik skorları ile çalışmanın başarılı olduğu görülmüştür. En başarılı algoritma %98 skorlarla MLP'dir. Bir derin öğrenme algoritması olan LSTM algoritmasında ise doğrulama verisinin skoru %85 olarak not edilmiştir. Çıkarılan verilere göre online metinler üzerinden yazarlık analizinin Adli Bilişim'e katkısı olacağının yanı sıra intihali ve yazarlık analizi içeren bilişim suçlarını engellemede yardımcı olacağı öngörülmektedir.

Anahtar Sözcükler: Adli Bilişim, Adli Dilbilim, Doğal Dil İşleme (DDİ), İntihal, Makine Öğrenme, Yazarlık Analizi.

SUMMARY

Contribution of Authorship Analysis to Computer Forensics in Columns

This research includes the analysis of authorship over 160 columns of 6 authors on different subjects. This study, which shows that each writer has an idiosyncratic writing habit, firstly includes a preprocessing step of all texts with Natural Language Processing method and then inserting them into machine learning algorithms, including deep learning. It was discussed whether the articles belonged to the authors, based on the training and test data allocated, and it was observed that the study was successful with the accuracy, recall and precision scores obtained as a result of the algorithms. The most successful algorithm is MLP with 98% scores. In the LSTM algorithm, which is a deep learning algorithm, the score of the validation data was noted as 85%. According to the data obtained, it has been predicted that the analysis of authorship over online texts will contribute to Computer Forensics, as well as helping to prevent plagiarism and cybercrimes including authorship analysis.

Key Words: Authorship Analysis, Computer Forensics, Forensic Linguistics, Machine Learning, Natural Language Processing (NLP), Plagiarism.

KAYNAKLAR

AGRAWAL S (2019). Sentiment Analysis Using LSTM (Step-By-Step Tutorial). Erişim Adresi: [https://towardsdatascience.com/sentiment-analysis-using-lstm-step-by-step-50d074f09948]. Erişim Tarihi: 11/10/2021.

BEM D, FELD F, HUEBNER E, BEM O (2008). Computer forensics-past, present and future. *Journal of Information Science and Technology*. **5:** 43-59.

BİLGİN AO (2018). Metin Madenciliği Yöntemleri ile Yazar Tanıma: Divan Edebiyatı Örneği. *Yüksek Lisans Tezi, Karadeniz Teknik Üniversitesi, Türkiye*.

BJÖRNSON E (2019). Is Plagiarism an Increasing Problem?. Erişim Adresi: [https://ma-mimo.ellintech.se/2019/01/01/is-plagiarism-an-increasing-problem]. Erişim Tarihi: 5/9/2021.

BURNAP P, COLOMBO G, SCOURFIELD J (2015). Machine classification and analysis of suicide-related communication on twitter. *Proceedings of the 26th ACM conference on hypertext & social media*. 75-84.

CHOLLET F (2020). The Sequential Model. Erişim Adresi: [https://keras.io/guides/sequential_model/]. Erişim Tarihi: 11/10/2021.

COULTHARD M, GRANT T, KREDENS K (2009). Forensic Linguistics. 1-7.

DINH TL (2014). Authorship Identification for Online Texts. *Yüksek Lisans Tezi, University of Groningen, Netherlands.*

EDUREKA (2020). What is String in Python: Everything You Need to Know. Erişim Adresi: [https://www.edureka.co/blog/what-is-string-in-python/]. Erişim Tarihi: 9/9/2021.

EKİNCİ E (2013). Elektronik Postaların Adli Analizinde Yazar Analizi Tekniklerinin Kullanılması. *Yüksek Lisans Tezi, Gebze Yüksek Teknoloji Enstitüsü, Türkiye*.

EKIZER AH (2014). Bilişim Suçları (Siber Suçlar). Erişim Adresi: [https://www.ekizer.net/bilisimsuclari-sibersuclar/]. Erişim Tarihi: 5/9/2021.

GALL R (2018). What is LSTM?. Erişim Adresi: [https://hub.packtpub.com/what-is-lstm/]. Erişim Tarihi: 11/10/2021.

GEITGEY A (2018). Natural Language Processing is Fun!. Erişim Adresi: [https://medium.com/@ageitgey/natural-language-processing-is-fun-9a0bff37854e]. Erişim Tarihi: 20/9/2021.

GRIANTI A (2020). Euclidean Distance Matrix. Erişim Adresi: [https://medium.com/swlh/euclidean-distance-matrix-4c3e1378d87f]. Erişim Tarihi: 20/9/2021.

GURAV S (2020). Label Encoder and OneHot Encoder in Python. Erişim Adresi: [https://towardsdatascience.com/label-encoder-and-onehot-encoder-in-python-83d32288b592]. Erişim Tarihi: 20/9/2021.

INDIAN AI PRODUCTION (2019). Python Matplotlib Tutorial - Mastery In Matplotlib Library. Erişim Adresi: [https://indianaiproduction.com/python-matplotlib-tutorial/]. Erişim Tarihi: 5/9/2021.

JAIN P (2021). Basics of CountVectorizer. Erişim Adresi: [https://towardsdatascience.com/basics-of-countvectorizer-e26677900f9c]. Erişim Tarihi: 5/9/2021.

JOHNSON D (2021). What is TensorFlow? How it Works? Introduction & Architecture. Erişim Adresi: [https://www.guru99.com/what-is-tensorflow.html]. Erişim Tarihi: 5/9/2021.

KADAM D. (2018). Academic integrity and plagiarism: The new regulations in India. *Indian Journal of Plastic Surgery*. **51:** 109-110.

KALINOWSKI T, ALLAIRE JJ, CHOLLET F. Text Tokenization Utility. Erişim Adresi: [https://keras.rstudio.com/reference/text_tokenizer.html]. Erişim Tarihi: 5/9/2021.

KANANI B (2019). TfidfVectorizer for Text Classification. Erişim Adresi: [https://studymachinelearning.com/tfidfvectorizer-for-text-classification/]. Erişim Tarihi: 9/10/2021.

KAUL S (2021). Text Feature Extraction (2/3): TF-IDF Model. Erişim Adresi: [https://medium.com/geekculture/text-feature-extraction-2-3-tf-idf-model-c3a8f7a92bc9]. Erişim Tarihi: 9/10/2021.

KERAS- Accuracy Metrics. Erişim Adresi: [https://keras.io/api/metrics/accuracy_metrics/#accuracy-class]. Erişim Tarihi: 5/9/2021.

KERAS- Adam. Erişim Adresi: [https://keras.io/api/optimizers/adam/]. Erişim Tarihi: 5/9/2021.

KERAS- Bidirectional Layer. Erişim Adresi: [https://keras.io/api/layers/recurrent_layers/bidirectional/]. Erişim Tarihi: 5/9/2021.

KERAS- Dropout Layer. Erişim Adresi: [https://keras.io/api/layers/regularization_layers/dropout/]. Erişim Tarihi: 5/9/2021.

KERAS - Flatten Layer. Erişim Adresi: [https://keras.io/api/layers/reshaping_layers/flatten/]. Erişim Tarihi: 5/9/2021.

KERAS- Layer Activation Functions. Erişim Adresi: [https://keras.io/api/layers/activations/#softmax-function]. Erişim Tarihi: 5/9/2021.

KERAS- Losses. Erişim Adresi: [https://keras.io/api/losses/]. Erişim Tarihi: 5/9/2021.

KERAS- Metrics. Erişim Adresi: [https://keras.io/api/metrics/]. Erişim Tarihi: 5/9/2021.

KERAS- Optimizers. Erişim Adresi: [https://keras.io/api/optimizers/]. Erişim Tarihi: 5/9/2021.

KERAS- Sparse Categorical Crossentropy Class. Erişim Adresi: [https://keras.io/api/losses/probabilistic_losses/#sparsecategoricalcrossentropy-class]. Erişim Tarihi: 5/9/2021.

KERAS- Text Data Preprocessing. Erişim Adresi: [https://keras.io/api/preprocessing/text/]. Erişim Tarihi: 5/9/2021.

KERAS- Timeseries Data Preprocessing. Erişim Adresi: [https://keras.io/api/preprocessing/timeseries/]. Erişim Tarihi: 5/9/2021.

KILIÇ S (2019). Adli Bilişim Nedir?. Erişim Adresi: [https://kernelblog.org/2019/11/adli-bilisim-nedir/]. Erişim Tarihi: 5/9/2021.

KINGMA DP, BA JL (2015). Adam: A method for stochastic optimization. *ICLR 2015*. 1-15.

KORSTANJE J (2021). The K-Nearest Neighbors (KNN) Algorithm in Python. Erişim Adresi: [https://realpython.com/knn-python/]. Erişim Tarihi: 9/10/2021.

KUZU RS (2010). Authorship Recognition in Online Social Platforms. *Yüksek Lisans Tezi, Boğaziçi Üniversitesi, Türkiye.*

KUZUCULAR Ş (2014). Türkçe'nin Ekleri ve Özellikleri. Erişim Adresi: [https://edebiyatvesanatakademisi.com/yazim-imla-ses-kelime/turkce-nin-ekleri-ve-ozellikleri/1103]. Erişim Tarihi: 9/10/2021.

MOHAMMED M, KHAN MB, BASHIER EBM (2016). Machine Learning: Algorithms and Applications. *Crc Press*.

MUELLER JP, MASSARON L (2019). What is Google Colaboratory?. Erişim Adresi: [https://www.dummies.com/programming/python/what-is-google-colaboratory/]. Erişim Tarihi: 9/10/2021.

MURNANE K (2016). What is Deep Learning and How is it Useful?. Erişim Adresi: [https://www.forbes.com/sites/kevinmurnane/2016/04/01/what-is-deep-learning-and-how-is-it-useful/?sh=33fea86bd547]. Erişim Tarihi: 9/10/2021.

OLSSON J. (2008). Forensic linguistics: An introduction to language, crime, and the law. 2nd Ed. Chapter 1.

O'REILLY MEDIA, INC. (2015). Web Scraping with Python. Erişim Adresi: [https://www.oreilly.com/library/view/web-scraping-with/9781491910283/ch01.html]. Erişim Tarihi: 5/9/2021.

PRABHAKARAN S (2018). Cosine Similarity - Understanding the Math and How it Works (With Python Codes). Erişim Adresi: [https://www.machinelearningplus.com/nlp/cosine-similarity/]. Erişim Tarihi: 5/9/2021.

PRATIK (2017). NumPy and Pandas Tutorial - Data Analysis with Python. Erişim Adresi: [https://cloudxlab.com/blog/numpy-pandas-introduction/]. Erişim Tarihi: 5/9/2021.

ROBINSON S (2017). What is Natural Language Processing?. Erişim Adresi: [https://stackabuse.com/what-is-natural-language-processing/]. Erişim Tarihi: 5/9/2021.

RSTUDIO- Train a Keras Model. Erişim Adresi: [https://keras.rstudio.com/reference/fit.html#see-also]. Erişim Tarihi: 21/10/2021.

SAKAL M (2020). Makine Öğrenmesi Algoritmaları Kısa Açıklamaları. Erişim Adresi: [http://muratsakal.com/?p=230]. Erişim Tarihi: 5/9/2021.

SCIKIT- LEARN. Multi-Dimensional Scaling (MDS). Erişim Adresi: [https://scikit-learn.org/stable/modules/manifold.html#multidimensional-scaling]. Erişim Tarihi: 5/9/2021.

SCIKIT- LEARN. Support Vector Machines, Decision Trees, Naïve Bayes, MLP Classifier. Erişim Adresi: [https://scikit-learn.org/stable/modules/classes.html]. Erişim Tarihi: 5/9/2021.

SHAKEEL I (2003). Introduction to Computer Forensics & Digital Investigation. *InfoSec Institute*. 1-79.

SHIFFMAN D (2012). The Nature of Code: Simulating Natural Systems with Processing. s.: 43-44.

SONI GK (2018). Plagiarism Detection and Prevention: A Study. *International Journal of Library & Information Science (IJLIS)*. **7:** 1-6.

STANZA- A Python NLP Package for Many Human Languages. Erişim Adresi: [https://stanfordnlp.github.io/stanza/]. Erişim Tarihi: 5/9/2021.

SUBRAMANIAN D (2019). A Simple Introduction to K-Nearest Neighbors Algorithm. Erişim Adresi: [https://towardsdatascience.com/a-simple-introduction-to-k-nearest-neighbors-algorithm-b3519ed98e]. Erişim Tarihi: 5/9/2021.

TARCAN A, ÇAKAR F (2008). Bilgisayarlı Dil Tanımlamada Dilbilimsel Yaklaşımlar ve Bir Yazılım Denemesi. *Elektronik Sosyal Bilimler Dergisi*. **7:** 64-70.

TOMIĆ KD (2019). Evidentiality Strategies as Distinguishing Markers in Forensic Authorship Analysis. *Анали Филолошког факултета*. **31:** 161-190.

TRACY R (2021). How to Compare Two Strings Using Sklearn's TfidfVectorizer and Cosine Similarity. Erişim Adresi: [https://medium.com/geekculture/how-to-compare-two-strings-using-sklearns-tdidfvectorizer-and-cosine-similarity-21e8b42371be]. Erişim Tarihi: 5/9/2021.

TROPINA T, BOYD MS (2017). Unit 1 Introduction to the Language of Cybercrime: Definitions and Discussion, EJTN and Cybercrime Investigations Capacity Building. *Handbook: The Language of Cybercrime.*

TUPA S (2017). What is Plagiarism?. Erişim Adresi: [https://www.plagiarism.org/article/what-is-plagiarism]. Erişim Tarihi: 9/10/2021.

TUTORIALSPOINT- Machine Learning with Python Algorithms. Erişim Adresi: [https://www.tutorialspoint.com/machine_learning_with_python/machine_learning_with_python_algorithms.htm]. Erişim Tarihi: 9/10/2021.

TUTORIALS POINT (I) PVT. LTD. (2019). Keras. s.: 1

TUTORIALS POINT (I) PVT. LTD. (2019). Scikit-Learn. s.:1

TÜRKOĞLU F, DIRI B, AMASYALI FM (2007). Author attribution of Turkish texts by feature mining. *International Conference on Intelligent Computing*. 1086-1093.

UÇAK NÖ, BIRINCI HG (2008). Bilimsel Etik ve İntihal. *Türk Kütüphaneciliği*. **22**: 187-204.

UNIVERSITY OF EDINBURGH (2016). What Is Informatics?. Erişim Adresi: [https://www.ed.ac.uk/files/atoms/files/what20is20informatics.pdf]. Erişim Tarihi: 21/9/2021.

UNRUH A (2017). What is the TensorFlow Machine Intelligence Platform?. Erişim Adresi: [https://opensource.com/article/17/11/intro-tensorflow]. Erişim Tarihi: 5/9/2021.

VERSLOOT C (2021). Build An LSTM Model with TensorFlow 2.0 and Keras. Erişim Adresi: [https://www.machinecurve.com/index.php/2021/01/07/build-an-lstm-model-with-tensorflow-and-keras/]. Erişim Tarihi: 5/9/2021.

VİKİPEDİ- Ankara. Erişim Adresi: [https://tr.wikipedia.org/wiki/Ankara]. Erişim Tarihi: 5/9/2021.

VIKIPEDI- Visual Studio Code. Erişim Adresi: [https://tr.wikipedia.org/wiki/Visual_Studio_Code]. Erişim Tarihi: 5/9/2021.

WIKIPEDIA- Computer Forensics. Erişim Adresi: [https://en.wikipedia.org/wiki/Computer_forensics]. Erişim Tarihi: 5/9/2021.

WIKIPEDIA- Decision Tree Learning. Erişim Adresi: [https://en.wikipedia.org/w/index.php?title=Decision_tree_learning&oldid=950332127]. Erişim Tarihi: 5/9/2021.

WIKIPEDIA- Machine Learning. Erişim Adresi: [https://en.wikipedia.org/wiki/Machine_learning]. Erişim Tarihi: 5/9/2021.

WIKIPEDIA- Multilayer Perceptron. Erişim Adresi: [https://en.wikipedia.org/wiki/Multilayer_perceptron]. Erişim Tarihi: 5/9/2021.

WIKIPEDIA- Natural Language Processing. Erişim Adresi: [https:/f/en.wikipedia.org/wiki/Natural_language_processing]. Erişim Tarihi: 5/9/2021.

WINGATE J (2020). NLP Tutorial Using Python Nltk, Urllib and BeautifulSoup. Erişim Adresi: [https://www.engineeringbigdata.com/nlp-using-python-example-with-urllib-and-beautifulsoup/]. Erişim Tarihi: 9/10/2021.

ZENDE MA, TUPLONDHE MB, WALUNJ SB, PARULEKAR SV (2016). Text Mining Using Python. *International Journal of Current Engineering and Scientific Research (IJCESR)*. **3:** 54-56.

EKLER

(Bu bölümde çalışmaya dair kodlar sıralanmış olup 2.1. Verilerin Toplanması ve 3.1. Verilerin Analizi bölümlerini kapsamaktadır.)

Ek-1: 2.1.2. İkinci Aşama Kodları

```python
#constructing link database and texts

from urllib.request import urlopen
from bs4 import BeautifulSoup
link = "https://www.hurriyet.com.tr/yazarlar/dogan-hizlan/ataturkun-bilinmeyen-dunyasi-41686500"
def get_text(link):
    html = urlopen(link).read()
    soup = BeautifulSoup(html,"html.parser")
    div_tag = soup.find("div", {"class":"article-content news-text"})
    paragraphs = div_tag.findChildren("p" , recursive=False)
    document = ""
    for par in paragraphs:
        document = document + par.text
    return document

link = "https://www.hurriyet.com.tr/yazarlar/dogan-hizlan/ataturkun-bilinmeyen-dunyasi-41686500"
out = get_text(link)
print(out)
```

```python
def get_links(news_list_link):
    html = urlopen(news_list_link).read()
    print(news_list_link)
    soup = BeautifulSoup(html, "html.parser")
    divs = soup.find_all("div", {"class":"highlighted-box mb20"})
    news_links = []
    for d in divs:
        a = d.find("a")
        l = a["href"]
        news_links.append(l)
    return news_links

link_database = []

for i in range(1,54):
    news_list_link = "https://www.hurriyet.com.tr/yazarlar/dogan-hizlan/?p={}".format(i)
    n_ls = get_links(news_list_link)
    for l in n_ls:
        updated_link = "https://www.hurriyet.com.tr" + l
        link_database.append(updated_link)

with open("links.txt", "w") as target:
    for link in link_database:
        target.write(link + "\n")

results = []
for link in link_database:
    print(link)
    out = get_text(link)
    results.append((link, out))
    print(out)
    print("\n\n----\n\n")

with open("dogan_texts.txt", "w", encoding='utf-8') as target:
    for link, out in results:
        target.write(link + "\n" + out + "\n\n")
```

Ek-2: 2.1.3. Üçüncü Aşama Kodları

```python
import os

text_directory = "C:\\Users\\cemre\\Desktop\\author_n_columns"
text_files = os.listdir(text_directory)
text_files.sort()

authors = []
contents = []
for text_file in text_files:
    if os.path.isfile(text_directory + "/" + text_file):
        content = open(text_directory + "/" + text_file, encoding="utf-8").read()
        contents.append(content)
        authors.append(text_file)

print("Karakter adetleri:")
for i in range(len(authors)):
    print(len(contents[i]))
print()
print("Kelime adetleri:")
for i in range(len(authors)):
    print(contents[i].count(" ") + 1)

num_authors = len(contents)
raw_texts = []
for i in range(num_authors):
    raw_texts.append([])

for index, content in enumerate(contents):
  lines = content.splitlines()
  text = ""
  for line in lines:
    if line != "":
        text += line + " "
    else:
        raw_texts[index].append(text)
        text = ""
  raw_texts[index].append(text)

print("Köşe yazısı adetleri")
for i in range(len(authors)):
    print(len(raw_texts[i]))
```

```python
import stanza

alphabet = "abcçdefgğhıijklmnoöprsştuüvyz"
nlp = stanza.Pipeline(lang="tr", processors='tokenize,mwt,pos,lemma')

for i in range(len(authors)):
 for text_no, text in enumerate(raw_texts[i]):
   doc = nlp(text)

   lemmas = ""
   for sentence in doc.sentences:
     for word in sentence.words:
            for c in word.lemma:
                if c not in alphabet:
                    if word.upos == "PROPN":
                        continue

            if word.upos not in ("PUNCT", "NUM", "AUX", "ADP")  and len(word.lemma) > 1:
                lemmas += word.lemma.lower() + " "
   print(lemmas)
```

```
stop_words = set(stopwords.words('turkish'))
word_tokens = word_tokenize(lemmas)
filtered_sentence = [w for w in word_tokens if not w.lower() in stop_words]
filtered_sentence = []
for w in word_tokens:
    if w not in stop_words:
        filtered_sentence.append(w)
#print(word_tokens)
print(filtered_sentence)
```

```
tokens = []
for token in filtered_sentence:
    if token.isdigit():
        continue
    if token in punctuation:
        continue
    if len(token) <= 1:
        continue
    token = token.replace("", "")
    if len(token) <= 1:
        continue
    if not token.isalpha():
        continue

    token = token.replace("â", "a").replace("î", "i").replace("û", "u")
    tokens.append(token)

print(tokens)

author_name = authors[i][:-4]
with open(f"lemmas/{author_name}_{text_no}.txt", "w", encoding='utf-8') as target:
    for token in tokens:
        target.write(token + " ")
```

Ek-3: 2.1.4. Dördüncü Aşama Kodları

```
from zipfile import ZipFile

dosya = 'lemmas.zip'

with ZipFile(dosya,'r') as zip_file:
          zip_file.printdir()
          zip_file.extractall()
```

```
import os
authors = ["dogan-hizlan", "ilber-ortayli", "melike-karakartal", "muge-akgun", "vedat-milor", "yasar-sokmensuer"]
for i, author in enumerate(authors):
  dir_path = "lemmas"
  text_files = os.listdir(dir_path)
  text_files.sort()
  authors_text_files = []
  for text_file in text_files:
    if text_file.startswith(author + "_"):
      authors_text_files.append(text_file)
  print(f"({i+1}) {author} has {len(authors_text_files)} lemma texts.")
  print(authors_text_files)
```

```
authors = ["dogan-hizlan", "ilber-ortayli", "melike-karakartal", "muge-akgun", "vedat-milor", "yasar-sokmensuer"]
number_of_texts = 160

for author in authors:
  total_lemmas = 0
  total_characters = 0
  for text_no in range(0, number_of_texts):
    f = open(f"lemmas/{author}_{text_no}.txt", mode="r")
    content = f.read()
    f.close()

    if content == "":
      continue

    if content[-1] == " ":
      content = content[:-1]
    lemmas = content.split(" ")
    total_lemmas += len(lemmas)
    total_characters += len(content) - content.count(" ")
  average_lemmas = total_lemmas / number_of_texts
  average_characters_in_a_lemma = total_characters / total_lemmas
  print(author, "için köşe yazısı başına düşen ortalama lemma adedi:", average_lemmas)
  print(author, "için lemma başına düşen ortalama karakter adedi:", average_characters_in_a_lemma)
```

```
dogan_hizlan_articles = []
dogan_hizlan_labels = []
author = "dogan-hizlan"
for article_no in range (0, 160):
  with open (f"/content/lemmas/{author}_{article_no}.txt" , "r") as f:
    text = f.read()
    dogan_hizlan_articles.append(text)
    dogan_hizlan_labels.append(author.replace("-", ""))

print("dogan_hizlan_articles:", len(dogan_hizlan_articles))
print("output:", dogan_hizlan_articles)
print("dogan_hizlan_labels:", len(dogan_hizlan_labels))
print("output:", dogan_hizlan_labels)
```

```
ilber_ortayli_articles = []
ilber_ortayli_labels = []
author = "ilber-ortayli"
for article_no in range (0, 160):
  with open (f"/content/lemmas/{author}_{article_no}.txt" , "r") as f:
    text = f.read()
    ilber_ortayli_articles.append(text)
    ilber_ortayli_labels.append(author.replace("-", ""))

melike_karakartal_articles = []
melike_karakartal_labels = []
author = "melike-karakartal"
for article_no in range (0, 160):
  with open (f"/content/lemmas/{author}_{article_no}.txt" , "r") as f:
    text = f.read()
    melike_karakartal_articles.append(text)
    melike_karakartal_labels.append(author.replace("-", ""))

muge_akgun_articles = []
muge_akgun_labels = []
author = "muge-akgun"
for article_no in range (0, 160):
  with open (f"/content/lemmas/{author}_{article_no}.txt" , "r") as f:
    text = f.read()
    muge_akgun_articles.append(text)
    muge_akgun_labels.append(author.replace("-", ""))

vedat_milor_articles = []
vedat_milor_labels = []
author = "vedat-milor"
for article_no in range (0, 160):
  with open (f"/content/lemmas/{author}_{article_no}.txt" , "r") as f:
    text = f.read()
    vedat_milor_articles.append(text)
    vedat_milor_labels.append(author.replace("-", ""))

yasar_sokmensuer_articles = []
yasar_sokmensuer_labels = []
author = "yasar-sokmensuer"
for article_no in range (0, 160):
  with open (f"/content/lemmas/{author}_{article_no}.txt" , "r") as f:
    text = f.read()
    yasar_sokmensuer_articles.append(text)
    yasar_sokmensuer_labels.append(author.replace("-", ""))
```

```
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer(min_df=0)
vectorizer.fit(articles)
print(vectorizer.vocabulary_)
print(len(vectorizer.vocabulary_))

matrix = vectorizer.transform(articles).toarray()
print(matrix.shape)

vectors = []
for row in matrix:
  vectors.append(row)
  print(row)
```

```python
import numpy as np

def unit_vector(vector):
    return vector / np.linalg.norm(vector)

def angular_distance(v1, v2):
    v1_u = unit_vector(v1)
    v2_u = unit_vector(v2)
    return np.arccos(np.clip(np.dot(v1_u, v2_u), -1.0, 1.0))

def find_average_vector(vectors):
  num_dims = len(vectors[0])
  average = np.zeros(num_dims)
  for vector in vectors:
    average += vector
  average /= len(vectors)
  return average

query = find_average_vector(vectors[10:17])
average_author1 = find_average_vector(vectors[50:100])
average_author2 = find_average_vector(vectors[200:250])
average_author3 = find_average_vector(vectors[350:400])
average_author4 = find_average_vector(vectors[500:550])
average_author5 = find_average_vector(vectors[700:750])
average_author6 = find_average_vector(vectors[800:850])

dist1 = angular_distance(query, average_author1)
dist2 = angular_distance(query, average_author2)
dist3 = angular_distance(query, average_author3)
dist4 = angular_distance(query, average_author4)
dist5 = angular_distance(query, average_author5)
dist6 = angular_distance(query, average_author6)
print(dist1)
print(dist2)
print(dist3)
print(dist4)
print(dist5)
print(dist6)
```

```python
from sklearn.manifold import MDS

matrix_normalized = np.zeros(matrix.shape)
for index, row in enumerate(matrix):
  row_normalized = unit_vector(row)
  matrix_normalized[index] = row_normalized

matrix_of_angular_distances = np.zeros((matrix.shape[0], matrix.shape[0]))
for row in range(matrix.shape[0]):
  for col in range(matrix.shape[0]):
    matrix_of_angular_distances[row, col] = angular_distance(matrix_normalized[row], matrix_normalized[col])

embedding = MDS(n_components=2, dissimilarity='precomputed')
matrix_transformed = embedding.fit_transform(matrix_of_angular_distances)

print(matrix_transformed.shape)
```

```python
import matplotlib.pyplot as plt

x_black = []
y_black = []
for row in matrix_transformed[10:17]:
  x_black.append(row[0])
  y_black.append(row[1])

x_red = []
y_red = []
for row in matrix_transformed[50:100]:
  x_red.append(row[0])
  y_red.append(row[1])

x_blue = []
y_blue = []
for row in matrix_transformed[200:250]:
  x_blue.append(row[0])
  y_blue.append(row[1])

x_green = []
y_green = []
for row in matrix_transformed[350:400]:
  x_green.append(row[0])
  y_green.append(row[1])

x_yellow = []
y_yellow = []
for row in matrix_transformed[500:550]:
  x_yellow.append(row[0])
  y_yellow.append(row[1])

x_magenta = []
y_magenta = []
for row in matrix_transformed[700:750]:
  x_magenta.append(row[0])
  y_magenta.append(row[1])

x_cyan = []
y_cyan = []
for row in matrix_transformed[800:850]:
  x_cyan.append(row[0])
  y_cyan.append(row[1])
```

```python
print("query(dogan_hizlan):", len(x_black))
print("dogan_hizlan:", len(x_red))
print("ilber_ortayli:", len(x_blue))
print("melike_karakartal:", len(x_green))
print("muge_akgun:", len(x_yellow))
print("vedat_milor:", len(x_magenta))
print("yasar_sokmensuer:", len(x_cyan))

plt.scatter(x_red, y_red, c="red")
plt.scatter(x_blue, y_blue, c="blue")
plt.scatter(x_green, y_green, c="green")
plt.scatter(x_yellow, y_yellow, c="yellow")
plt.scatter(x_magenta, y_magenta, c="magenta")
plt.scatter(x_cyan, y_cyan, c="cyan")
plt.scatter(x_black, y_black, c="black")
```

```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score

classifiers = [KNeighborsClassifier(n_neighbors = 1), KNeighborsClassifier(n_neighbors = 5),
               KNeighborsClassifier(n_neighbors = 10)]
classifier_names = ["KNN1", "KNN5", "KNN10"]

for classifier, classifier_name in zip(classifiers, classifier_names):
  print(classifier_name)
  classifier.fit(X_train_cv, y_train)
  predictions = classifier.predict(X_test_cv)
  print("CountVectorizer")
  print("Accuracy score: ", accuracy_score(y_test, predictions))
  print("Recall score: ", recall_score(y_test, predictions, average='weighted'))
  print("Precision score: ", precision_score(y_test, predictions, average='weighted'))

  classifier.fit(X_train_tv, y_train)
  predictions = classifier.predict(X_test_tv)
  print("TfidfVectorizer")
  print("Accuracy score: ", accuracy_score(y_test, predictions))
  print("Recall score: ", recall_score(y_test, predictions, average='weighted'))
  print("Precision score: ", precision_score(y_test, predictions, average='weighted'))
  print()
```

```python
error1= []
error2= []
for k in range(1,20):
    knn= KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train_cv,y_train)
    y_pred1= knn.predict(X_train_cv)
    error1.append(np.mean(y_train!= y_pred1))
    y_pred2= knn.predict(X_test_cv)
    error2.append(np.mean(y_test!= y_pred2))
plt.plot(range(1,20),error1,label="train")
plt.plot(range(1,20),error2,label="test")
plt.xlabel('k Value')
plt.ylabel('Error')
plt.legend()
```

```python
from sklearn import metrics
knn= KNeighborsClassifier(n_neighbors=k)
knn.fit(X_train_cv,y_train)
y_pred= knn.predict(X_test_cv)
metrics.accuracy_score(y_test,y_pred)
```

```python
from sklearn import metrics
knn= KNeighborsClassifier(n_neighbors=k)
knn.fit(X_train_cv,y_train)
y_pred= knn.predict(X_test_cv)
metrics.precision_score(y_test,y_pred, average='weighted')
```

```python
from sklearn.naive_bayes import MultinomialNB, BernoulliNB, ComplementNB
from sklearn.metrics import accuracy_score, precision_score, recall_score
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer

classifiers = [BernoulliNB(), MultinomialNB(), ComplementNB()]
classifier_names = ["BernoulliNB", "MultinomialNB", "ComplementNB()"]

all_predictions = []
for classifier, classifier_name in zip(classifiers, classifier_names):
  print(classifier_name)
  classifier.fit(X_train_cv, y_train)
  predictions = classifier.predict(X_test_cv)
  print("CountVectorizer")
  print("Accuracy score: ", accuracy_score(y_test, predictions))
  print("Recall score: ", recall_score(y_test, predictions, average='weighted'))
  print("Precision score: ", precision_score(y_test, predictions, average='weighted'))
  all_predictions.append(predictions)

  classifier.fit(X_train_tv, y_train)
  predictions = classifier.predict(X_test_tv)
  print("TfidfVectorizer")
  print("Accuracy score: ", accuracy_score(y_test, predictions))
  print("Recall score: ", recall_score(y_test, predictions, average='weighted'))
  print("Precision score: ", precision_score(y_test, predictions, average='weighted'))
  print()
  all_predictions.append(predictions)
```

```python
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
i = 5  # 0 -> bernoulli count, 1 -> bernoulli tfidf, 2 -> multinomial count,
       # 3 -> multinomial tfidf, 4 -> complement count, 5 -> complement tfidf
cm = confusion_matrix(y_test, all_predictions[i])
sns.heatmap(cm, square=True, annot=True, cmap="RdBu", cbar=False,
            xticklabels=["doganhizlan", "ilberortayli", "melikekarakartal", "mugeakgun",
                         "vedatmilor", "yasarsokmensuer"],
            yticklabels=["doganhizlan", "ilberortayli", "melikekarakartal", "mugeakgun",
                         "vedatmilor", "yasarsokmensuer"])
plt.xlabel("true label")
plt.ylabel("predicted label")
```

```python
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer

classifiers = [SVC(kernel="linear"), SVC(kernel="rbf")]
classifier_names = ["SVC_linear", "SVC_rbf"]

all_predictions = []
for classifier, classifier_name in zip(classifiers, classifier_names):
  print(classifier_name)
  classifier.fit(X_train_cv, y_train)
  predictions = classifier.predict(X_test_cv)
  print("CountVectorizer")
  print("Accuracy score: ", accuracy_score(y_test, predictions))
  print("Recall score: ", recall_score(y_test, predictions, average='weighted'))
  print("Precision score: ", precision_score(y_test, predictions, average='weighted'))
  all_predictions.append(predictions)

  classifier.fit(X_train_tv, y_train)
  predictions = classifier.predict(X_test_tv)
  print("TfidfVectorizer")
  print("Accuracy score: ", accuracy_score(y_test, predictions))
  print("Recall score: ", recall_score(y_test, predictions, average='weighted'))
  print("Precision score: ", precision_score(y_test, predictions, average='weighted'))
  print()
  all_predictions.append(predictions)
```

```python
import pandas as pd
import seaborn as sns
from sklearn.metrics import confusion_matrix

i = 0  # 0 -> linear count, 1 -> linear tfidf, 2 -> rbf count, 3 -> rbf tfidf

cm = confusion_matrix(y_test, all_predictions[i], labels=["doganhizlan", "ilberortayli", "melikekarakartal",
                                                          "mugeakgun", "vedatmilor", "yasarsokmensuer"],)
cm_matrix = pd.DataFrame(data=cm, columns=["doganhizlan", "ilberortayli", "melikekarakartal",
                                           "mugeakgun", "vedatmilor", "yasarsokmensuer"],
                         index=["doganhizlan", "ilberortayli", "melikekarakartal",
                                "mugeakgun", "vedatmilor", "yasarsokmensuer"])
sns.heatmap(cm_matrix, annot=True, fmt='d', cmap='YlGnBu')
```

```python
from sklearn.neural_network import MLPClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer

classifiers = [MLPClassifier(), DecisionTreeClassifier()]
classifier_names = ["MLP", "DecisionTree"]

for classifier, classifier_name in zip(classifiers, classifier_names):
  print(classifier_name)
  classifier.fit(X_train_cv, y_train)
  predictions = classifier.predict(X_test_cv)
  print("CountVectorizer")
  print("Accuracy score: ", accuracy_score(y_test, predictions))
  print("Recall score: ", recall_score(y_test, predictions, average='weighted'))
  print("Precision score: ", precision_score(y_test, predictions, average='weighted'))

  classifier.fit(X_train_tv, y_train)
  predictions = classifier.predict(X_test_tv)
  print("TfidfVectorizer")
  print("Accuracy score: ", accuracy_score(y_test, predictions))
  print("Recall score: ", recall_score(y_test, predictions, average='weighted'))
  print("Precision score: ", precision_score(y_test, predictions, average='weighted'))
  print()
```

```python
random.seed(0)

articles = dogan_hizlan_articles + ilber_ortayli_articles + melike_karakartal_articles + muge_akgun_articles + vedat_milor_articles + yasar_sokmensuer_article
labels = dogan_hizlan_labels + ilber_ortayli_labels + melike_karakartal_labels + muge_akgun_labels + vedat_milor_labels + yasar_sokmensuer_labels

all = list(zip(articles, labels))
random.shuffle(all)
articles, labels = zip(*all)

train_size = int(len(articles) * training_portion)

print("eğitim kümesinin boyutu", train_size)
print("test kümesinin boyutu", len(articles) - train_size)

train_articles = articles[:train_size]
test_articles = articles[train_size:]
train_labels = labels[:train_size]
test_labels = labels[train_size:]

print(train_articles[0])
print(train_labels[0])
```

```python
tokenizer = Tokenizer(num_words=vocab_size, oov_token=oov_tok)
tokenizer.fit_on_texts(train_articles)
print(tokenizer.word_index)
print(len(tokenizer.word_index))

train_sequences = tokenizer.texts_to_sequences(train_articles)
print(train_sequences)

train_padded = pad_sequences(train_sequences, padding = padding_type, truncating=trunc_type, maxlen=max_length)
print(train_padded)
```

```
tokenizer = Tokenizer(num_words=vocab_size, oov_token=oov_tok)
tokenizer.fit_on_texts(test_articles)
print(tokenizer.word_index)
print(len(tokenizer.word_index))

test_sequences = tokenizer.texts_to_sequences(test_articles)
print(test_sequences)
test_padded = pad_sequences(test_sequences, maxlen=max_length, padding=padding_type, truncating=trunc_type)
print(test_padded)
```

```
label_tokenizer = Tokenizer()
label_tokenizer.fit_on_texts(train_labels)
print(label_tokenizer.word_index)
print(len(label_tokenizer.word_index))

train_label_sequences = np.array(label_tokenizer.texts_to_sequences(train_labels))
train_label_sequences -= 1
print(train_label_sequences)
```

```
label_tokenizer = Tokenizer()
label_tokenizer.fit_on_texts(test_labels)
print(label_tokenizer.word_index)
print(len(label_tokenizer.word_index))

test_label_sequences = np.array(label_tokenizer.texts_to_sequences(test_labels))
test_label_sequences -= 1
print(test_label_sequences)
```

```
model = Sequential()
model.add(Embedding(vocab_size, embedding_dim))
model.add(Dropout(dropout_rate))
model.add(Bidirectional(LSTM(embedding_dim)))
model.add(Dense(num_classes, activation="softmax"))
model.summary()
```

```
optimizer = Adam(learning_rate=0.001, decay=1e-6)
model.compile(optimizer=optimizer, loss="sparse_categorical_crossentropy", metrics=["accuracy"])
```

```
num_epochs = 20
history = model.fit(x=train_padded, y=train_label_sequences, epochs=num_epochs,
                    validation_data=(test_padded,test_label_sequences), verbose=2)
```

Ek-4: 3.1. Verilerin Analizi Kodları

```
USE_TFIDF = True # True olursa TfidfVectorizer kullanılır, False olursa CountVectorizer kullanılır

from tqdm import tqdm
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.naive_bayes import MultinomialNB, BernoulliNB, ComplementNB
from sklearn.metrics import accuracy_score, precision_score, recall_score
from sklearn.svm import SVC
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
#np.random.seed(0)
accuracies = {"KNN1":0, "KNN5":0, "KNN10":0, "MLP":0, "Tree":0, "BernoulliNB":0, "MultinomialNB":0,
              "ComplementNB":0, "SVC_linear":0, "SVC_kernel":0}
recalls = {"KNN1":0, "KNN5":0, "KNN10":0, "MLP":0, "Tree":0, "BernoulliNB":0, "MultinomialNB":0,
           "ComplementNB":0, "SVC_linear":0, "SVC_kernel":0}
precisions = {"KNN1":0, "KNN5":0, "KNN10":0, "MLP":0, "Tree":0, "BernoulliNB":0, "MultinomialNB":0,
              "ComplementNB":0, "SVC_linear":0, "SVC_kernel":0}
for seed in tqdm(range (1, 11)):
  X_train, X_test, y_train, y_test = train_test_split(articles, labels, random_state=seed,
                                                      test_size=0.20)

  cv = CountVectorizer()
  X_train_cv = cv.fit_transform(X_train)
  X_test_cv = cv.transform(X_test)

  tv = TfidfVectorizer()
  X_train_tv = tv.fit_transform(X_train)
  X_test_tv = tv.transform(X_test)

  classifiers = [KNeighborsClassifier(n_neighbors = 1), KNeighborsClassifier(n_neighbors = 5),
                 KNeighborsClassifier(n_neighbors = 10), MLPClassifier(), DecisionTreeClassifier(),
                 BernoulliNB(), MultinomialNB(), ComplementNB(), SVC(kernel="linear"),
                 SVC(kernel="rbf")]
  classifier_names = ["KNN1", "KNN5", "KNN10", "MLP", "Tree", "BernoulliNB", "MultinomialNB",
                      "ComplementNB", "SVC_linear", "SVC_kernel"]

  for classifier, classifier_name in zip(classifiers, classifier_names):

    if USE_TFIDF:
      #print(classifier_name)
      classifier.fit(X_train_tv, y_train)
      predictions = classifier.predict(X_test_tv)
      accuracies [classifier_name] += accuracy_score(y_test, predictions)
      recalls [classifier_name] += recall_score(y_test, predictions, average="weighted")
      precisions [classifier_name] += precision_score(y_test, predictions, average="weighted")

    else:
      classifier.fit(X_train_cv, y_train)
      predictions = classifier.predict(X_test_cv)
      accuracies [classifier_name] += accuracy_score(y_test, predictions)
      recalls [classifier_name] += recall_score(y_test, predictions, average="weighted")
      precisions [classifier_name] += precision_score(y_test, predictions, average="weighted")

for key, value in recalls.items():
  recalls[key] /=10

for key, value in precisions.items():
  precisions[key] /=10

for key, value in accuracies.items():
  accuracies[key] /=10

print("TfidfVectorizer kullanarak" if USE_TFIDF else "CountVectorizer kullanarak")
print(accuracies)
print(recalls)
print(precisions)
```

```
import matplotlib.pyplot as plt

def plot_graphs(history, string):
  plt.plot(history.history[string])
  plt.plot(history.history['val_'+string])
  plt.xlabel("Epochs")
  plt.ylabel(string)
  plt.legend([string, 'val_'+string])
  plt.show()

plot_graphs(history, "accuracy")
plot_graphs(history, "loss")
```