**T.C.**
**SAKARYA ÜNİVERSİTESİ**
**BİLGİSAYAR VE BİLİŞİM BİLİMLERİ FAKÜLTESİ**

**BSM 498 BİTİRME ÇALIŞMASI**

# BİLGİSAYARLI GÖRÜ YÖNTEMLERİ KULLANILARAK BİR ANTİBOT SİSTEMİNİ GEÇEBİLEN SİSTEMİN TASARLANMASI

**B161210074 – Ahmet Yasir AKBAL**

**Fakülte Anabilim Dalı : BİLGİSAYAR MÜHENDİSLİĞİ**
**Tez Danışmanı : Dr. Öğr. Üyesi M. Fatih ADAK**

**2019-2020 Yaz Dönemi**

**T.C.**
**SAKARYA ÜNİVERSİTESİ**
**BİLGİSAYAR VE BİLİŞİM BİLİMLERİ FAKÜLTESİ**

# BİLGİSAYARLI GÖRÜ YÖNTEMLERİ KULLANILARAK BİR ANTİBOT SİSTEMİNİ GEÇEBİLEN SİSTEMİN TASARLANMASI

**BSM 498 - BİTİRME ÇALIŞMASI**

**Ahmet Yasir AKBAL**

**Fakülte Anabilim Dalı : BİLGİSAYAR MÜHENDİSLİĞİ**

**Bu tez .. / .. / … tarihinde aşağıdaki jüri tarafından oybirliği / oyçokluğu ile kabul edilmiştir.**

| ……. | ……. | ……. |
|---|---|---|
| **Jüri Başkanı** | **Üye** | **Üye** |

# ÖNSÖZ

Günümüzde Yapay Zekanın, sağlık, iletişim, güvenlik, otomotiv gibi bir çok sektörde kullanımı gün geçtikçe yaygınlaşmaktadır. Bir çok şirket ve devlet yatırımlarını bu alanda arttıkmakta, çoğu üniversitede bu ders zorunlu olarak verilmektedir. Bu çalışmada Bulanık Mantık, Yapay Sinir Ağları/Derin Öğrenme, Makine Öğrenmesi, Genetik Algoritmalar, Uzman Sistemler gibi alt başlıklara ayrılan Yapay Zekanın Derin Öğrenme alt başlığına ait yöntemleri kullanacağız. Bu alt alan son günlerde yaşanan teknolojik gelişmeler sonucu donanım maliyetinin azalması, veri miktarının artması ve bu verinin işlenme ihtiyacı, donanım performansının artması, donanıma erişimin kolaylaşması ve yeni oluşturulan daha iyi sonuçlar veren modeller ile patlama yaşamıştır.

# ÖZET

Anahtar kelimeler: Veri, Yapay Zeka, Makine Öğrenmesi, Derin Öğrenme, Bilgisayarlı Görü

Veri bir yapay zeka sisteminin tasarımındaki en kritik noktadır. Yapay Zeka sistemleri verilere göre insanlar tarafından kurallaştırılır veya öğrenirler. Verinin olmadığı bir durumda bir yapay zeka sistemi tasarlamak olanaksızdır.

Yapay Zeka, bilgisayarlara insanların yaptığı gibi bir olayı algılayıp, bu olay üzerinde düşünüp sonuç çıkarma kabiliyetinin kazandırılmasını hedefler. Burada yapay zekanın algıladığı şey yapay zekanın üzerinde işlem yapacağı veridir. Bu verilere örnek olarak görüntü, ses veya metin verilebilir. Yapay zeka aldığı bu veriyi işleyerek bir sonuca varır. Örnek olarak en optimal yolun bulunması, resimdeki nesnenin tespit edilmesi, resmin sınıflandırılması verilebilir.

Makine öğrenmesi, yapay zekanın bir alt dalıdır. Bu alt dalda sisteme yapay zeka özelliğinin kazandırılması işlemi tasarımcılar tarafından yazılan kurallar(örneğin : durağan yapay zeka, bulanık mantık) ile değil de kullanılan makine öğrenme algoritmasının verileri işleyerek, öğrenek bir model çıkarması sonucu olmaktadır.

Derin öğrenme ise uzun yıllar önce de var olan yapay sinir ağlarının genişletilmesi ile karşımıza çıkmıştır. Yakın zamanda ve gelecekte veri sayısının artması ve bu büyük boyutlardaki verileri işlemeye olan ihtiyacı , daha güçlü donanımların tasarlanmış olması ve donanım maliyetinin azalması derin öğrenme alanının doğmasına sebep olmuştur.

Bilgisayarlı görü ise insanın görme duyusu sayesinde yaptığı algılamaları makinelere yaptırmaktır. Eskiden yapay zekanın bir alt dalı olarak görülse de günümüzde başlı başına bir alan haline gelmiştir. Bilgisayarlı görüde de yapay zeka teknikleri kullanılır.

# BÖLÜM 1. GİRİŞ

Yapay zeka, kabaca; bir bilgisayarın ya da bilgisayar denetimli bir makinenin, genellikle insana özgü nitelikler olduğu varsayılan akıl yürütme, anlam çıkartma, genelleme ve geçmiş deneyimlerden öğrenme gibi yüksek zihinsel süreçlere ilişkin görevleri yerine getirme yeteneği olarak tanımlanmaktadır.[1]

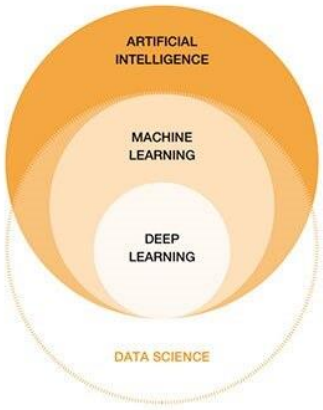
## 1.1. Yapay Zekanın Kısa Bir Tarihçesi

Yapay zeka için 1884 yılı oldukça önemlidir. Bu tarihte Charles Babbage, zeki davranışlar gösterecek bir mekanik makine üzerinde çalışma yapmıştır. Fakat çalışmaları sonucu insanlar kadar zeki davranışlar sergileyecek bir makine üretilemeyeceğine karar vermiş ve çalışmalarına ara vermiştir. 1950 yılında Alan Turing bir makinenin zeki olmadığına karar veren meşhur Turing Testi'ni oluşturmuştur.[2] O dönemde bu testi geçebilen makinelerin zeka seviyesi yeterli sayılmaktaydı. 1956 yılında yapay zeka resmi olarak ilk kez İngilteredeki bir konferansda John McCarty öncülüğünde ortaya atılmıştır. Daha sonra 1957 yılında John McCarty yapay zeka uygulamaları için fonksiyonel LISP(List Processing Language) dilini geliştirmiştir.

1965-1970 yılları arası yapay zeka için karanlık bir dönem olarak adlandırabilir. Ortaya çıkan gerçekçi olmayan beklentiler nedeniyle oluşan aceleci ve iyimser tutum insanlar kadar zeki olabilecek makinelerin üretilmesinin kolay olacağı düşüncesini oluşturmuştur. 1970-1975 yılları arası ise yapay zekanın ivme kazandığı bir dönemdir. 1975-1980 yılları arasında ise psikoloji başta olmak üzere başka bilim dallarından yapay zekaya katkı sağlanabileceği düşüncesi oluşmuştur. 1980 yıllardan günümüze kadar ise yapay zeka gelişimine devam etmiştir. Günümüzden örnek olarak sürücüsüz araçlar, nesne tespiti, ses tanıma, yüz tanıma, doğal dil işleme verilebilir.

## 1.2. Yapay Zekanın Alt Dalları

Yapay Zeka kendi içinde Makine Öğrenmesi, Derin Öğrenme, Bulanık Mantık, Genetik Algoritmalar gibi alt alanlara ayrılır. Günümüzde makine öğrenmesi ve onun alt dalı olan derin öğrenme hayatımızın bir çok alanında karşımıza çıkmakta ve bizlere fayda sağlamaktadır. Özellikle büyük verinin modellenme ihtiyacının arttığı günümüzde derin öğrenme modellerinin önemi çok fazladır.




Şekil 1.1. Yapay zeka ve kesiştiği ilgili alanlar

Şekil 1.1'de Yapay Zeka, Makine Öğrenmesi, Derin Öğrenme ve Veri Bilimi dalları arasında ilişki gösterilmiştir. Görüldüğü gibi bu 4 alanında birbiri ile en azından belli bölümlerde ilişkisi bulunmaktadır.

Veri bilimi, makine öğrenmesi tekniklerini de için de barındırır. Veri biliminde amaç sadece veriyi bir makine öğrenme algoritması ile modellemek olmadığı için veri bilimi makine öğrenmesinden daha büyük bir alandır diyebiliriz. Veri biliminde verinin istatiksel olarak incelenmesi, verideki değişkenlerin aralarında nasıl bağlantı olduğunun incelenmesi, görselleştirilmesi, sebep-sonuç ilişkisinin çıkarılması gibi işlemler de yapılır.

Yapay zekanın bir alt dalı olan makine öğrenmesi klasik makine öğrenme algoritmalrını ve derin öğrenme algoritmalarını içermektedir. Günümüzde her ne kadar derin öğrenme daha popüler olsa da temel makine öğrenme algoritmaları

diyebileceğimiz(derin öğrenme dışında tuttuklarımız) hala çok iyi başarımlar vermekte ve kullanılmaktadır. Özellikle yakın zaman duyurulmuş GBM'ler(Graident Boosting Machines) bazı problemlerde en iyi başarımı veren algoritmalardan olmuştur.

Derin öğrenme yıllardır var olan yapay sinir ağlarınındaki ara katman sayılarının genişletilmesi ile daha geniş ağlar oluşturulması ile ortaya çıkmıştır. Bu katmanlar sadece nöronlara benzetilen birimlerden oluşan katmanlar değildir. Konvolüsyonel katmanlar, pooling katmanları, recurrent katmanları vb. olabilir. Teknoloji geliştikçe donanım maliyetinin azalması ve erişebilirliğinin kolaylaşması, veri boyutunun artması sonucu temel makine öğrenme algoritmalarının bu büyük verileri modellemede yetersiz kalması derin öğrenmenin doğmasına sebep olmuştur.

# BÖLÜM 2. VERİ

Yapay zeka sistemlerinde verinin yeri, önemi ve kullanımından bahsetmeden önce veri ve veri ile ilgili temel tanımlamalardan bahsedelim. **Değişken**, nicel ya da nitel anlamda bir özelliğin belirgin olarak bir durumdan diğerine farklılık göstermesi olarak tanımlanabilir. Değişkenle ilgili denek ya da objenin değerine ise **veri** denir. Birey ya da objenin belli bir özelliğe sahip olması miktar olarak açıklanabiliyorsa bu tür değişkenlere **nicel değişken** denir. Akademik başarı puanı, ağırlık ölçüsü, zeka puanı, gelir miktarı, kütüphanedeki kitap sayısı, bir ailenin sahip olduğu çocuk sayısı nicel değişkenlere örnek olarak verilebilir. **Nitel değişken** ise birey ya da objelerin sahip olunan belli bir özellik açısından sınıflara ayrılmasını gösterir. Cinsiyet, yerleşim birimi, öğrenim görülen bölüm gibi değişkenler nitel değişkenlerdir (veri analizi el kitabı pegem). Nicel değişkenler sonraki bölümde bahsedilecek olan veri çeşitlerinden olan sayısal değişkenlere, nitel değişkenler ise kategorik değişkenlere karşılık gelecektir.

Değişkenlerin aldığı değerler göre değişkenler sürekli ve süreksiz olarak sınıflandırılır. **Sürekli değişkenler** değişkenin tanım aralığında sonsuz sayıda değer alabilen değişkenlerdir. Örnek olarak bir nesnenin uzunluğunu belirten bir değişkenin 5-10 cm arasında bir değer aralığı olsun. Değişkenimiz sürekli olduğu için bu aralık arasında sonsuz sayıda değer olacaktır. **Süreksiz değişkenler** ise kesikli değerler alan değişkenlerdir. Bir kitaptaki sayfa sayısı, bir kişinin doğum yeri gibi değerler örnek olarak verilebilir. Bu örneklerden görüldüğü üzere süreksiz değişkenler nicel(sayısal) veya nitel(kategorik) olabilirler.

Değişkenler ayrıca **bağımlı** ve **bağımsız değişkenler** olarak da gruplandırılabilmektedir. Bu tür değişkenler arasında bir neden sonuç ilişkisi vardır; bağımlı değişkenin alacağı değeri, bağımsız değişkenin aldığı değerler belirler. Örneğin “y = ax+b” denkleminde “y” değeri “x”’in alabileceği değerlere bağlı olarak değişeceği için bir bağımlı değişkendir. Buradaki a ve b değerlerini sabit olduğu

bağımlı veya bağımsız değişken olmadığını da belirtelim. Makine öğrenmesi algoritmalarında çoğu zaman(denetimli öğrenme algoritmalarında) amacımız bu bağımsız değişkenler ile bağımlı değişkenin değerini bize verebilecek bir model oluşturmak olacaktır. Veri ile ilgili temel tanımlamalar verildiğine göre veri ve algoritma ilişkisini daha yakından inceyelim.

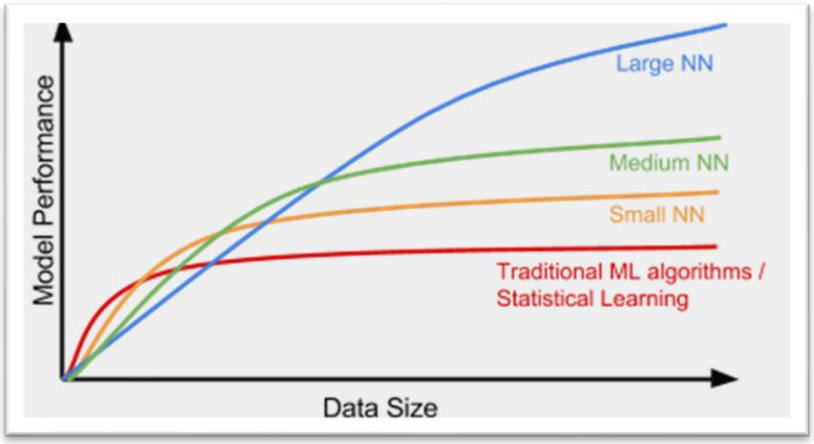
Bir yapay zeka sisteminden bahsediyorsak, veri bu sistemdeki en önemli ögedir. Elimizde ne kadar zeki bir sistem olursa olsun onu veri ile eğitecek, programlayacağız. Sistemimizde kullanacağımız veri tutarlı, yeterli miktarda, probleme uygun değilse sistemimize zeka niteliği kazandırmak zorlaşacaktır.

Verideki tutarsızlığa örnek verecek olursak elimizde bulunan atmosferik bilgilerden hava sıcaklığı tahmini yapıldığını düşünelim. Veri setimizdeki bir satırdaki bilgilerden çıktı olarak hesapladığımız hava sıcaklığı 30C olsun. Yine başka satırda bu girdilere çok yakın veya aynı girdilerle hesaplanan çıktının da bu bu değere yakın olmasını bekleriz. Fakat böyle olmazsa veride dolayısıyla sistemimizde tutarsızlıklar meydana gelecek ve sisteme zeka özelliği kazandırmak zorlaşacaktır. (Burada not olarak eklememiz gereken ise bu örnekteki girdilerdeki ufak bir değişikliğin bazı problemlerde çıktıyı gerçekten de çok ciddi boyutlarda değiştirebileceğidir. Bu durumlar probleme özgüdür). Aynı şekilde verisetinde aynı girdiler için farklı çıktıların olduğu satırların olması da tutarsızlıktır.

Verisetimizdeki bazı değerler verisetinin kalanına göre çok farklı olabilir. İstatistikte bu durum aykırı değer(outlier) olarak adlandırılır ve daha sonra değinilecek olan veri önişleme süreçlerinde çözülmesi gereken bir sorundur. Bunun yanında bazı makine öğrenme algoritmalarının bu aykırı değerlere karşı bağışıklıklı olduğunu ekliyorum.


Şekil 2.1. Outlier veri

Verinin miktarının başarıma etkisinden bahsedecek olursak genel olarak tüm makine öğrenmesi algoritmalarında veri miktarının artması başarımı bir noktaya kadar iyi yönde etkileyecek olsa da bu durum kullanılan makine öğrenmesi algoritmasına göre de değişecektir. Klasik makine öğrenme algoritmaları genelde veri boyutu azken, yapay sinir ağları/derin öğrenme algoritmalarına göre bir nebze daha iyi başarım verirken veri boyutu arttıkça yapay sinir ağları özellikle de derin sinir ağları daha iyi başarım vermektedir. Yetersiz veri sorununu çözmek için kullanılan transfer learning, sentetik veri oluşturma, veri zenginleştirme gibi yöntemler ise ileride anlatılacaktır.



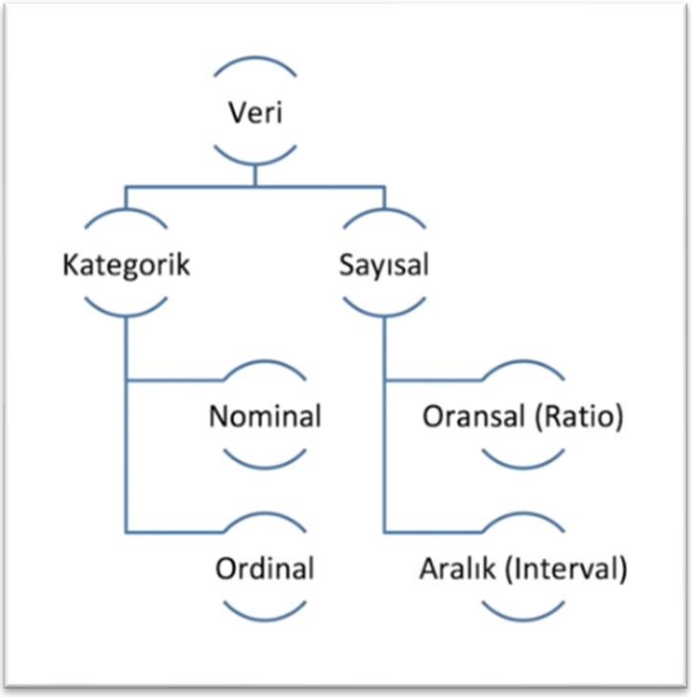

Şekil 2.2. Klasik makine öğrenmesi ile yapay sinir ağlarının karşılaştırılması

Problemimizde kullanacağımız veri kapsayıcı olmalıdır. Olabildiğince fazla senaryoda veri örneği içererek modelin genelleştirme yapmasına olanak sağlamalıdır. Elimizdeki verisetinde bireyin cinsiyetini belirten bir girdi değişkenin yanında çeşitli değişkenler ve çıktı/çıktılarımız olsun. Modelimizi sadece bu verisetindeki cinsiyeti erkek olan örnekler ile eğitip daha sonra modelden cinsiyeti kadın olan bir veri örneğini verip tahmin yapmasını istediğimizde model muhtemelen istenilen başarıda olmayan bir sonuç verecektir. (Bu örnekte cinsiyet değişkeninin çıktıya etkisinin hatrı sayılır bir seviyede olduğunu farzettik).

Verisetimiz probleme uygun olmalıdır. Örneğin biraz uç bir örnekle açıklayacak olursak gözetimli öğrenme algoritmasında girdi olarak kişinin saç rengini, hobilerini, göz rengi gibi girdileri verip kişinin belirlenen bir hastalığa sahip olup olmadığını tahmin etmeye çalışmak yanlış olacaktır. Verisetinin probleme uygun olup olmadığını saptamak daha çok problemi anlamakla alakalı olup problemin konusunda

uzmanlaşmış biri tarafından değerlendirilmesi mantıklı olacaktır. Bunun yanında verisetinde bulunan değişkenlerden hangilerini modele dahil edeceğimizi belirlememize yarayan backward/forward elimination gibi yöntemlerden ilerleyen kısımlarda bahsedilecektir.

## 2.1. Veri Çeşitleri




Şekil 2.3. Veri çeşitleri[3]

Genel olarak veriyi 2 farklı kategoriye ayırabiliriz; kategorik ve sayısal veriler. Bu iki veri çeşidi de kendi arasında ikiye ayrılır. Bunları ayrı ayrı başlıklar altında inceleyelim.

### 2.1.1. Kategorik Veriler

Elimizdeki veriyi gruplara ayırabiliyorsak bu veri kategorik bir veridir. Örnek olarak canlı türlerinin olduğu bir veriseti elimizde olsun. Buradaki veriler hayvan, bitki, insan gibi bir grubu temsil edecektir.

#### 2.1.1.1. nominal veriler

Birbirleri üzerinde büyüklük, küçüklük ilişkisi olmayan verilerdir. Bir fakültenin bölümlerini düşünelim. Bu bölümler arasında A bölümü B bölümünden daha büyüktür veya küçüktür gibi ilişkiler kurmak anlamsız olacaktır. Kısacası bu tür veriler birbiri arasında sıralanamazlar.

#### 2.1.1.2. ordinal veriler

Nominal verilerden farklı olarak birbirleri arasında sıralanabilen verilerdir. Örnek olarak askeri rütbeleri(binbaşı, onbaşından daha üst bir seviyededir), bir okuldaki öğrencilerinin sınıflarını(8.sınıf, 7.sınıftan daha yukarıda olacaktır) verebiliriz. Burada önemli olan kısım ise buradaki büyüklük-küçüklük ilişkisinin tam olarak matematikte gördüğümüz "8 sayısı 4 sayısından 2 kat büyüktür" gibi net bir ilişki olmadığıdır.

### 2.1.2. Sayısal veriler

Bir sayı ile ifade edilen verilerdir. Oransal ve aralık veriler olarak ikiye ayrılırlar.

#### 2.1.2.1. oransal(ratio) veriler

Birbirleri arasında mantıklı bir oransal ilişki kurulabilen verilerdir. Örneğin "A tamsayısı B tamsayısından 2 kat büyüktür" cümlesinde olduğu gibi 2 tamsayı arasında oransal bir ilişki kurabiliriz ve bu doğrudur. Aynı şekilde bir canlının yaşı arasında da oransal bir ilişki kurmak mümkündür ve doğrudur.

#### 2.1.2.2. aralık(interval) veriler

Birbiriyle oransal bir ilişki kurulamayan(kurulması mantıksız olan) veri çeşitledir. Sıcaklık(Celcius, Fahrenayt sıcaklık birimleri), pH değeri örnek olarak verilebilir. Örneklerden birini açıklayacak olursak: 40C, 20C'den 2 kat daha sıcaktır gibi bir orantısal ilişki kurmak doğru olmayacaktır.

Tablo 2.1. Veri çeşitlerine göre yapılması doğru olan/olmayan işlemler

| Yapılması doğrudur | Nominal | Ordinal | Interval | Ratio |
|---|---|---|---|---|
| Frekans dağılımı | Evet | Evet | Evet | Evet |
| Medyan alma | Hayır | Evet | Evet | Evet |
| Ekleme ve Çıkarma | Hayır | Hayır | Evet | Evet |
| Ortalama, standart sapma, standart hata | Hayır | Hayır | Evet | Evet |
| Oran alma, değişim katsayısı | Hayır | Hayır | Hayır | Evet |

## 2.2. Veri Önişleme Süreçleri

Veri ön işleme süreci verimiz makine öğrenme algoritmasına verilmeden önce yapılan işlemlerdir. Veri madenciliği, büyük veri gibi alanlarda bu ön işleme adımı daha da genişlemektedir. Bu yazıda elimizde hazır bir veri dosyasının olduğu(örneğin kaggledan indirilmiş bir veriseti, şirketin hazırlayıp bize bir model oluşturmamız için verdiği bir veriseti) zaman hangi ön işleme adımlarının yapılacağından bahsedilecektir.

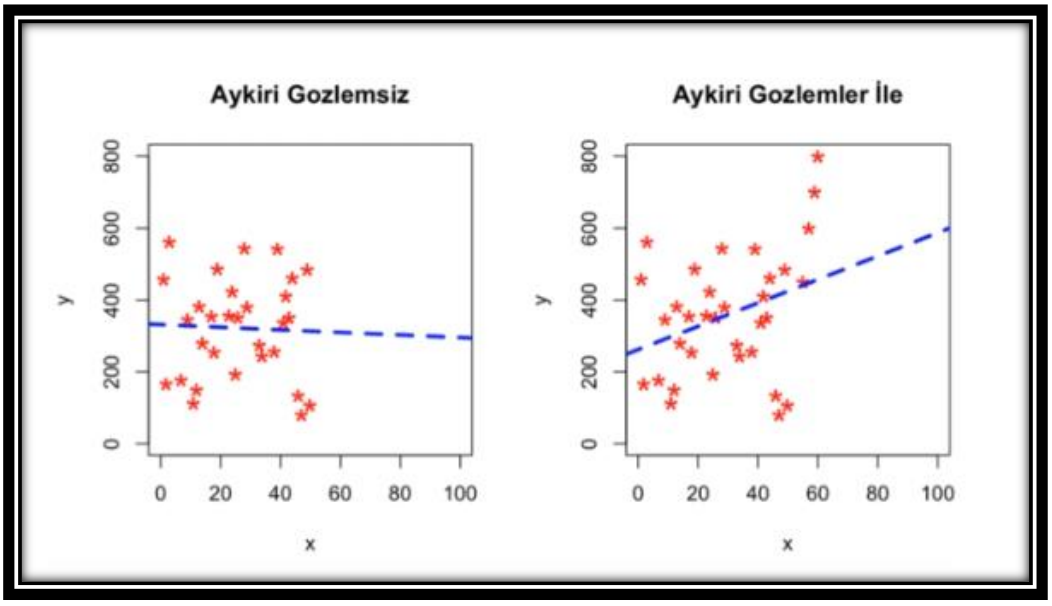
- Veri Temizleme (Data Cleaning / Cleasing)
    - Aykırı Gözlem Analizi (Outlier Analysis)
    - Eksik Veri Analizi (Missing Data Analysis)
- Veri Ölçekleme (Data Scaling)
    - Normalizasyon
    - Standartlaştırma
- Değişken Dönüşümü
    - Label Encoding
    - One-Hot Encoding
    - Dummy(Tuzak) Değişken
    - Sürekli değişkeni kategorik değişkene çevirme
    - Biri/Birkaçı 1, Diğerleri 0 Dönüşümü
- Değişken(Variable) Seçimi

### 2.2.1. Veri Temizleme

Kullanacağımız veriler çoğu zaman elimize kirli bir şekilde gelecektir. Bu bahsedilen kirliliklerin neler olduğu alt başlıklar halinde incelenmiştir.

#### 2.2.1.1. Aykırı gözlem analizi(outlier analysis)

Veride genel eğilimin oldukça dışına çıkan ya da diğer gözlemlerden oldukça farklı olan gözlemlere aykırı gözlem denir. Aykırı gözlemler genelleme yapmak amacıyla oluşturulan kural setlerini(örn; bulanık mantıkta tanımladığımız kurallar, karar ağaçlarındaki koşullar), fonksiyonları(örn; regresyon modellerindeki fonksiyonlar) yanıltarak modelimizin başarımını ciddi seviyede düşürebilir. Toplam örnek sayısına oranla çok daha az sayıda olsa bile aykırı gözlemler modeli ciddi boyutlarda değiştirebilmektedir. Bu yüzden veri önişleme sürecinde bu sorunun çözülmesi çok önemlidir.




Şekil 2.4. Aykiri gözlemlerin modele etkisi[4]

Yukarıdaki şekilde bir girdisi ve bir çıktısı olan regresyon modeli sol tarafta aykırı gözlem bulunmayan bir veriseti gösterilmiştir. Aynı verisetine bazı aykırı gözlemler eklenerek eğitim gerçekleştirildiğinde oluşan model sağdadır. Şekli yorumlayacak olursak aykırı gözlemin modelin ne denli değiştirdiği görülmektedir. Modelin eğiminde ciddi bir değişim söz konusudur.

Aykırı gözlemin ne olduğundan bahsettik şimdi de aykırı gözlemleri tespit etmek için kullanılan yöntemlerden bahsedelim.

### (a) Sektör bilgisi

Aykırı gözlem tespitinde kullanılan en sık yöntemlerden birisidir. Problemin ait olduğu sektör hakkında bilinen bilgiler doğrultusunda uzman/lar tarafından aykırı gözlemlerin tespit edilmesidir. Bu yöntemde aykırı gözlemi tespit eden direkt olarak insandır. Bu yöntemi bir örnekle açıklamaya çalışırsak: bir bitkinin yaprak uzunluğu,genişliği vb. bilgilerden bitkinin türünü tahmin etmeye çalışalım. Örneklerden bazılarında yaprak genişliği diğerlerine göre çok daha büyük olsun. Verisetini inceleyen uzman kişi bu gözlemlerin olağandışı, tekrar gerçekleşme olasılığının çok düşük olduğunu tespit ederek bu değerleri aykırı gözlem olarak işaretleyebilir. Eğer eğitilecek model genelleme kaygısı ile eğitiliyorsa(çoğu zaman böyle olur) bu aykırı gözlemleri ya verisetinen çıkarmalı ya da bir yöntem kullanılarak manipule edilmelidir. Bu yöntemlerden ilerleyen bölümde bahsedilecektir.

### (b) Standart Sapma Yaklaşımı

Aykırı gözlem analizi yapılacak değişkenin ortalamasından standart sapması(ya da standart sapmanın 2 veya 3 katı) eklenip çıkarılarak bir aralık belirlenir. Bu aralığın dışında kalan gözlemler aykırı gözlem olarak belirlenir.
Örneğin;
$\mu$ = ortalama, $\sigma$ = standart sapma

$$\mu - 2.\sigma < x < \mu + 2.\sigma \qquad \textbf{(2.1)}$$

Yukarıdaki eşitliği sağlamayan x değerleri aykırı gözlem olarak seçilebilir.

### (c) Z Skoru Hesaplama

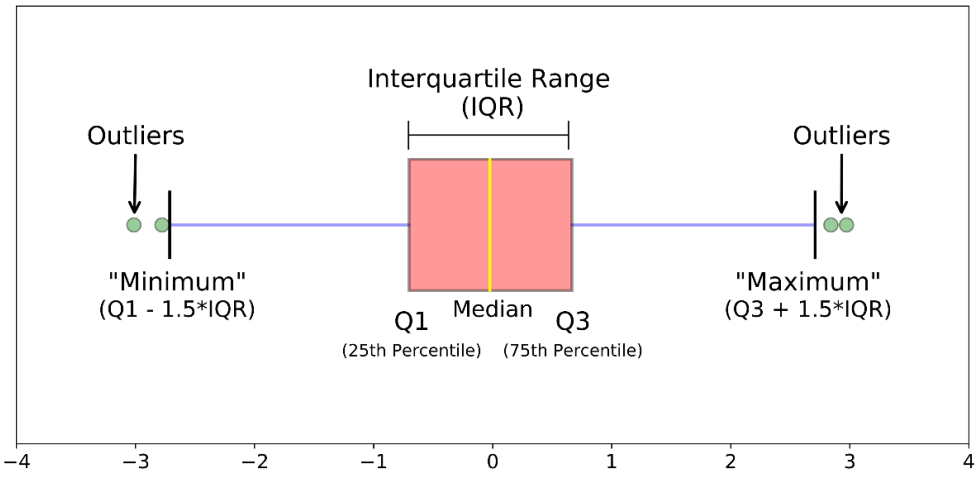
Standart sapma yaklaşımına benzer olarak bir değer hesaplanıp ortalamadan çıkartılıp eklenerek bir aralık belirlenir. Burada hesaplanan değer ise Z skorudur. Hesaplanan Z skoru işleme sokulmadan önce 2.5 katsayısı ile çarpılır.

$$Z = (x - \mu) / \sigma \quad \textbf{(2.2)}$$

$$\mu - 2.5 * Z < x < \mu + 2.5 * Z \quad \textbf{(2.3)}$$

### (d) Boxplot(interquartile range-IQR) Yöntemi

Aykırı gözlem tespiti yapılırken kullanılan en sık yöntemlerden biridir. Boxplot yöntemini uygulanırken hesaplanan değerler ile bir boxplot grafiği de çizilebilir. Bu grafik veri analizinde veri hakkında bilgi edinmek amacıyla sıkça kullanılmaktadır.




Şekil 2.5. Boxplot grafiği

Bu yöntemde öncelikle aykırı değerlerini tespit etmek istediğimiz değişkenin değerleri küçükten büyüğe doğru sıralanır. Verinin öyle bir değeri seçilirki verinin sağında ve solunda kalan örnek sayısı birbirine eşit olsun. Bu istatistikten bildiğimiz medyan değeridir. Bu değer bu yöntemde ayrıca **Q2** değeri olarak da adlandırılır. Q2 değerinin solundaki veri kümesinin medyan değeri bulunur bu değer **Q1** olarak adlandırılır. Bu değeri şöyle açıklayabiliriz: analizi yapılan bu değişkendeki veriler küçükten büyüğe doğru sıralandığında Q1 değerine kadar olan değerler bize verisetinin yüzde 25'lik kısmını vermektedir.

Aynı şekilde Q2 değerinin sağında kalan veriler içinde bir medyan hesaplandığında bu da bize veriseti küçükten büyüğe doğru sıralandığında hangi değere kadar olan verilerin bize verisetinin yüzde 75'lik kısmını vereceğidir. Bu değer **Q3** değeri olarak

adlandırılır. Şekil 2.5'deki grafikte Q1 ve Q3 değerleri temsil edilmiştir. Median değeri olarak gösterilen değer ise Q2 değeridir.
Q1 ve Q3 değerleri hesaplandıktan sonra bir aralık belirlenip bu aralık dışında kalan değerler aykırı gözlem olarak belirlenecektir. Bunu yapmak için yapılacak ilk işlem **IQR** değerini hesaplamaktır:

$$\mathrm{IQR} = 1.5 * (\mathrm{Q3} - \mathrm{Q1}) \quad \textbf{(2.4)}$$

Hesaplanan bu IQR değeri yardımıyla alt eşik değeri(minimum nokta) ve üst eşik değeri(maksimum nokta) hesaplanır. Bu iki noktanın oluşturduğu aralık dışında kalan değerler aykırı gözlem olarak karşımıza çıkar.

$$\mathrm{MIN} = \mathrm{Q1} - \mathrm{IQR} \quad \textbf{(2.5)}$$

$$\mathrm{MAX} = \mathrm{Q3} + \mathrm{IQR} \quad \textbf{(2.6)}$$

Artık aykırı gözlemleri tespit edebildiğimize göre bu problemi çözmek için neler yapılabilir ondan bahsedelim. Akla gelecek en basit yöntem bu **aykırı gözlem içeren satırları verisetinden kaldırmaktır**. Bir diğer yöntem ise aykırı gözlem içermeyen **verilerin ortalamasını alıp** bu ortalama değerini aykırı gözlem içeren tüm değişkenlere atamaktır.

Ortalama ile doldurma yönteminde aykırı değer ile onun doldurulacağı değer arasındaki fark çok fazla olabilir. Bu durumda ortalama yöntemini kullanmaktansa bir başka yöntem olan **baskılama yöntemini** kullanmak daha akıllıca olacaktır. Baskılama yönteminde: aykırı değer, alt eşik değerinden(Q1 veya bizim belirlediğimiz bir değer) daha küçükse alt eşik değeriyle, üst eşik değerinden büyük ise üst eşik değeri ile doldurulur.

Buraya kadar anlatılanlar tek değişkenli aykırı gözlem analizyle alakalı kısımlardı. Tek değişkeni inceleyerek aykırı gözlem tespiti yapmak bazı aykırılıkların gözden kaçmasına sebep olabilmektedir. Bu yüzden aykırı gözlem analizi yaparken birden fazla değişken ile analiz yapılması da gerekebilmektedir. Örneğin, bir verisetinde kişinin yaşı, boyu vb. değişkenler olsun. Biz burada sadece boy değişkenini ele alarak tek değişkenli bir aykırı gözlem analizi yaparsak boyu olandışı derece kısa

olanları(örneğin 140cm altını seçelim), olağandışı seviyede yüksek olanları(örneğin 210cm üstünü seçelim) aykırı gözlem olarak kabul edebiliriz. Fakat verisetimizde yaşı 15 olup boyu 190cm olan biri olduğunu düşünelim. Böyle bir durumda sadece boy değişkenini hesaba katarak tek değişkenli aykırı gözlem analizi yaptığımızda bu olağandışı durumu yakalayamayacağız. Bu tip durumları atlamamak için aykırı gözlem analizini yeri geldiğinde çok değişkenli olarak da yaparız. Python'da sklearn kütüphanesi içinde bulunan LocalOutlierFactor sınıfı ile bu işlem yapılabilmektedir.

#### 2.2.1.2. Eksik veri analizi(missing data analysis)

Verisetlerindeki veriler her zaman eksiksiz olarak karşımıza çıkmaz. Makine öğrenme algoritmaları eksik veri ile çalışamadığı için veriyi sunmadan önce verisetimizi eksin verilerden arındırmak zorundayız. Bunun için öncelikle eksik verileri tespit etmeliyiz. Eksik veriler genelde verisetinde "NaN", "Na","NULL","-" gibi ifadelerle karşımıza çıkar. Eksik verilerin tespiti ve bu sorunun çözümü hakkında detaya girmeden önce eksik verilerin hangi sebeplerden dolayı oluşabileceğini inceleyelim:

- Donanımsal bozukluklar
- Uyuşmazlık yüzünden silinen veriler
- Anlaşılamayan verilerin girilmiş olması
- Veri girişi sırasında veriye önem verilmemiş olması
- Verideki değişikliklerin kaydedilmemiş olması

Eksik verilerin çözülmesi için kullanılabilecek yöntemler aşağıdaki şekilde listelenmiştir:

- **Silme Yöntemleri**
  - Gözlem ya da değişken silme yöntemi
  - Liste bazında silme yöntemi (Listwise Method)
  - Çiftler bazında silme yöntemi (Pairwise Method)
- **Değer Atama Yöntemleri**
  - Ortanca, ortalama, medyan
  - En Benzer Birime Atama (hot deck)
  - Dış Kaynaklı Atama
- **Tahmine Dayalı Yöntemler**
  - Makine Öğrenmesi
  - EM
  - Çoklu Atama Yöntemi

Şekil 2.6. Eksik veri probleminin çözülmesi için kullanılan yöntemler

Silme yöntemlerinde eksik verinin bulunduğu gözlemi(satırı) silebileceğimiz gibi direkt olarak eksik veri içeren değişkeni(sütunu) de silebiliriz. Bu işlemleri yapmadan önce bir eşik oran değeri belirleriz. Örneğin bir satırdaki değeri eksik olan değişkenlerin sayısı tüm değişkenlerin sayısının yüzde 50sinden fazlaysa bu satırı direkt olarak silebiriz. Benzer mantıkla bir değişkenin verisetinde eksik olduğu satır sayısı tüm satır sayısınının yüzde 60'ından fazlaysa o değişkeni direkt olarak silebiliriz. Bu noktada hayati öneme sahip nokta ise silinecek değişkenin rastgele bir durumdan dolayı ortaya çıkan bir eksiklik mi olduğu yoksa yapısal bir eksiklik mi olduğudur. Konuya giriş kısmında listelenen eksik verinin oluşma nedenlerini rastgele olarak oluşmuş bozukluklar olarak ele alabiliriz. Yapısal bozukluklardan ortaya çıkan eksiklikler ise verisetindeki diğer değişkenlerle bağlantılı olarak ortaya çıkan eksikliklerdir. Örneğin elimizde hayvanları listelendiği bir veriseti olsun. Verisetinde hayvan ismi, ayak sayısı vb. değişkenlerimiz olsun. Verisetindeki "yılan" hayvanına ait satırdaki ayak sayısı değerinin daha önce "Na","NaN","Null","-" gibi eksik veriyi ifade eden bir değerle doldurulmuş olması olasıdır. Bu durumda buradaki eksiklik rastgele olarak değil, verisetinin yapısı gereği bir başka değişkenden(burada hayvan değişkeninden kaynaklı olarak: yılanların ayağı olmaz) kaynaklı olarak oluşmuş bir eksikliktir. Bu durumda eksik verinin bulunduğu bu satırı komple silmek doğru olmayacaktır. Bir başka durum olarak ise elimizde kişinin kredi kartına sahip olma durumu, kredi kartı ile yaptığı aylık harcama tutarı bulunan bir veriseti olsun. Burada kişi kredi kartına sahip olduğu halde o ayda kredi kartıyla herhangi harcama yapmamış olabilir. Verisetimizde de harcama tutarı 0 olarak değil de yine "Na","NaN","Null","-" gibi eksik veriyi ifade eden bir değerle ifade edilmiş olabilir. Bu durumda da eksik veri içeren bu satırı direkt olarak silmek mantıklı olmayacaktır. Onun yerine bu değeri 0 ile doldurabiliriz. Bu örnekte yılan örneğinde olduğu gibi tam anlamıyla verisetindeki başka bir değişkenle ilişkili olarak ortaya çıkan yapısal bir eksiklik olmadığının, biraz daha farklı bir durumun söz konusu olduğunu belirtmek istiyorum.

Değer atama yöntemlerinde ise istatiksel yöntemler(ortanca, ortalama, medyan), eksik gözlem içeren satırın kendisine en yakın diğer bir satıra atandığı hot deck yöntemi, bir uzman tarafından eksik gözlemlerin doldurulduğu dış kaynaklı atama yöntemi bulunmaktadır.

Tahmine dayalı yöntemler eksik gözlem sorunun çözmede kullanılan ileri seviye yöntemlerdir. Bu yöntemlerde eksikliğe sahip değişkenler bir bağımlı değişken olarak kabul edilerek bu eksik değerlerin hangi değer ile doldurulacağı kullanılan algoritma tarafından belirlenir.

### 2.2.2. Veri Ölçekleme

Veri ölçekleme işlemleri verinin dağılımını değiştirmeden değer aralıklarının değiştirilmesi işlemidir. Kategorik olmayan verilere uygulanır. Bir çok makine öğrenme algoritması çoğu durumda değişkenler ölçeklendiğinde daha yüksek başarım verir ve daha hızlı yakınsama gösterir. Bu makine öğrenme algoritmalarından bahsedecek olursak:

- Lineer ve lojistik regresyon
- En yakın komşular(KNN)
- Sinir ağları
- Support Vector Machine-Radial kernel fonksiyonu ile
- Gradient descent kullanan algoritmalar(burada sıralananların çoğunu kapsar)
- Principal Compenent Analysis
- Lineer Discrimant Analysis

Kullanacağımız verisetleri içinde bulunan değişkenler farklı aralıklarda olabilir. Örnek bir verisetinde bireyin yaşını belirten sütun 0 ile 100 arasında değerler alırken, aylık gelirini belirten bir başka değişken 0 ile 1.000.000 arasında bir değer alabilir. Böyle bir durumda verilerde bir ölçekleme yapmadan direkt olarak veriyi bir yapay sinir ağına verirsek aralığı fazla olan değişkenin modele etkisi daha fazla olacak, diğer değişkenleri baskılayacaktır. Her ne kadar kullanılan optimizasyon algoritması aralığı fazla olan bu değişkenin ağırlık değerini azaltarak modeli dengelemeye çalışsa da bu çoğu zaman yetersiz olacak, ağımızın daha düşük başarım vermesini ve daha geç yakınsamasını sağlayacaktır. Bu sebepten ötürü elimizdeki farklı değişkenlerin aralıkları birbirinden çok farklıysa verilerimizi ölçeklemek gerekecektir. Veri ölçekleme normalizasyon ve standartizasyon olmak üzere 2 farklı yöntemle yapılabilir.

#### 2.2.2.1. Normalizasyon

Değişkenin değer aralığının [0,1] veya tasarımcı tarafından belirlenen bir [a,b] aralığına sıkıştırılmasıdır.

Normalizasyon 2 şekilde yapılır.

- Değişkenin aldığı değerler 0 ile 1 arasına sıkıştırılır.

$$X = (x - x_{\min}) \,/\, (x_{\max} - x_{\min}) \tag{2.7}$$

- Değişkenin aldığı değerler a ile b arasına sıkıştırılır.

$$X = a + \big((x - x_{\min}).(b - a)\big)/(x_{\max} - x_{\min}) \tag{2.8}$$

Min-max(0-1) normalizasyonunda değişkene ait minumum değer o değişkene ait diğer değerlerden çok çok daha küçükse normalizasyon sonucu oluşan değerler 1 değerine doğru, değişkene ait maksimum değer o değişkene ait diğer değerlerden çok çok daha büyükse normalizasyon sonucu oluşan değerler 0 değerine doğru yığılacaktır. Bu aşırı yığılma sonucu veriler birbirine çok yakın virgüllü değerler olabilir. Bu yığılmalar eğitilecek modelin başarısını direkt olarak olumsuz etkileyebilir. Ayrıca kullanılan veri tipinin hassasiyeti bu noktada yetersiz kalarak kırpılmalar sonucu veri kaybı söz konusu olabilir.

#### 2.2.2.2. Standartizasyon

Standartizasyon yöntemlerinde genelde değişkenden bir değer çıkarılıp sonuç bir başka değere bölünür. Buradaki değerlerin farklılığı standartizasyon yöntemini belirler. Burada standartlaştırma yöntemlerinden olan Z dönüşümü ile standartizasyon ve robust scaler incelenecektir.

- **Z dönüşümü ile standartizasyon(diğer adıyla standart scaler)**

$$X = (x - \mu) \,/\, \sigma \tag{2.9}$$

Bu yöntemde değişkene ait değerden o değişkene ait aritmetik ortalama değeri çıkarılır ver çıkan sonuç o değişkene ait standart sapmaya bölünür. Z dönüşümü ile standartizasyon yapıldığında ortalamaya eşit değerler sıfır, ortalamadan küçük

değerler negatif, ortalamadan büyük değerler ise pozitif olur. Bu yöntem aykırı gözlem içeren bir sütuna uygulandığında aykırı gözlem sorunu aynen devam edecektir. Z dönüşümü uygulunan değişkenin ortalaması 0, standart sapması 1 olacaktır.

- **RobustScaler**

Bir diğer standartizasyon çeşididir. Aykırı değerlerin etkisini bir nebze düşürebilmektedir. RobustScaler sonucu oluşan değişkenin aralık değeri MinMax ve Standart Scalere göre daha büyüktür.

$$\mathrm{IQR} = (\mathrm{Q3} - \mathrm{Q1}) \quad \textbf{(2.10)}$$

$$\mathrm{X} = (\mathrm{x} - \text{medyan}) \,/\, IQR \quad \textbf{(2.11)}$$

Bu noktadan sonra aklımıza normalizasyon mu yoksa standartizasyon yöntemlerini kullanmalıyım sorusu gelebilir. Bu sorunun cevabı olacak katı ve net bir kural olmasa da şunlardan bahsedebiliriz:

- Verilimizin normal dağılıma uygun dağıldığını düşünüyorsak standartizasyon(Z dönüşümü metodunu)
- Verimizin hangi dağılıma sahip olduğuyla ilgilenmeyen bir makine öğrenmesi algoritması(KNN, yapay sinir ağları gibi) kullanıyorsak normalizasyonu

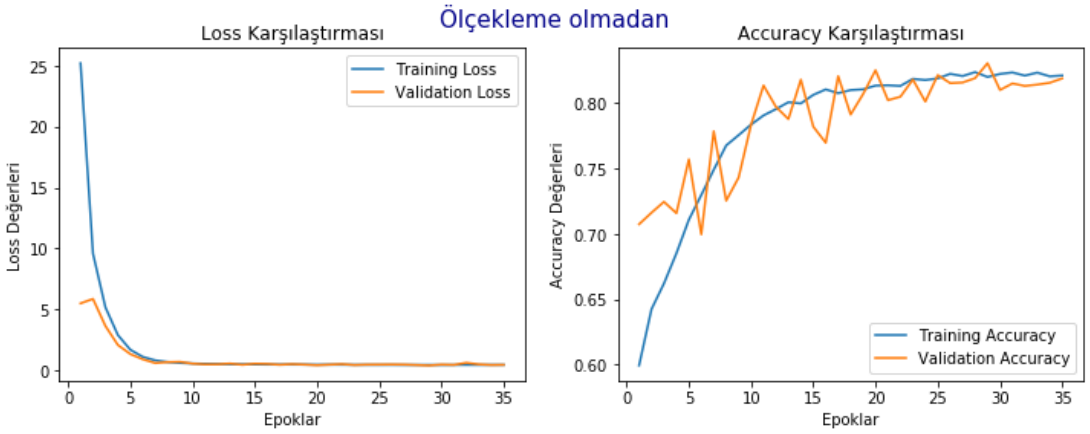
kullanabiliriz. Fakat günün sonunda bunlar kesin olarak her zaman bize doğru seçimi yapmamızı sağlamayacak ifadeler olup seçim yapmamız gerektiğinde her iki ölçekleme metodu da uygulayarak hangisinin daha iyi başarım verdiği gözlemlemek en garanti yol olacaktır.

Şimdi de ölçekleme işleminin bir yapay sinir ağı modelinin başarımını nasıl etkilediğini görmek adına bir örnek yapalım.

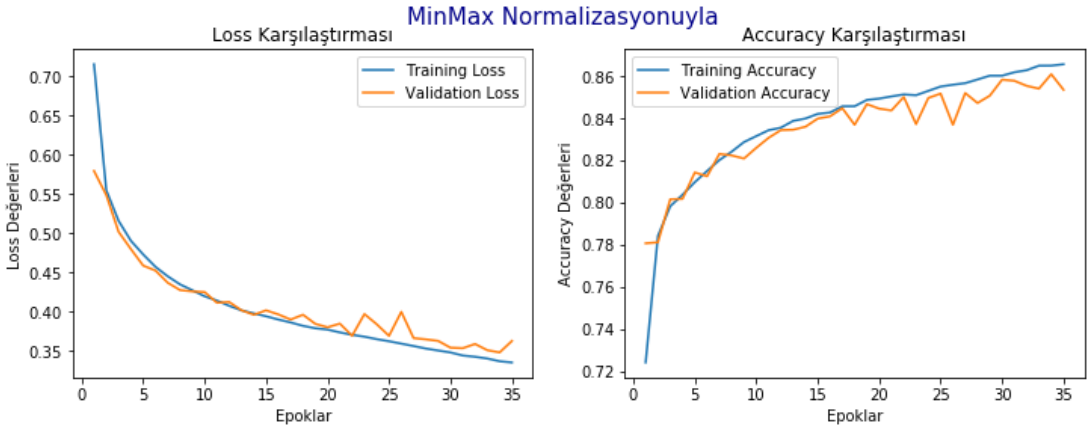
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | ... | 45 | 46 | 47 | 48 | 49 | 50 | 51 | 52 | 53 | 54 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 2596 | 51 | 3 | 258 | 0 | 510 | 221 | 232 | 148 | 6279 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 5 |
| **1** | 2590 | 56 | 2 | 212 | -6 | 390 | 220 | 235 | 151 | 6225 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 5 |
| **2** | 2804 | 139 | 9 | 268 | 65 | 3180 | 234 | 238 | 135 | 6121 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 |
| **3** | 2785 | 155 | 18 | 242 | 118 | 3090 | 238 | 238 | 122 | 6211 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 |
| **4** | 2595 | 45 | 2 | 153 | -1 | 391 | 220 | 234 | 150 | 6172 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 5 |

Şekil 2.7. Covertype Dataset

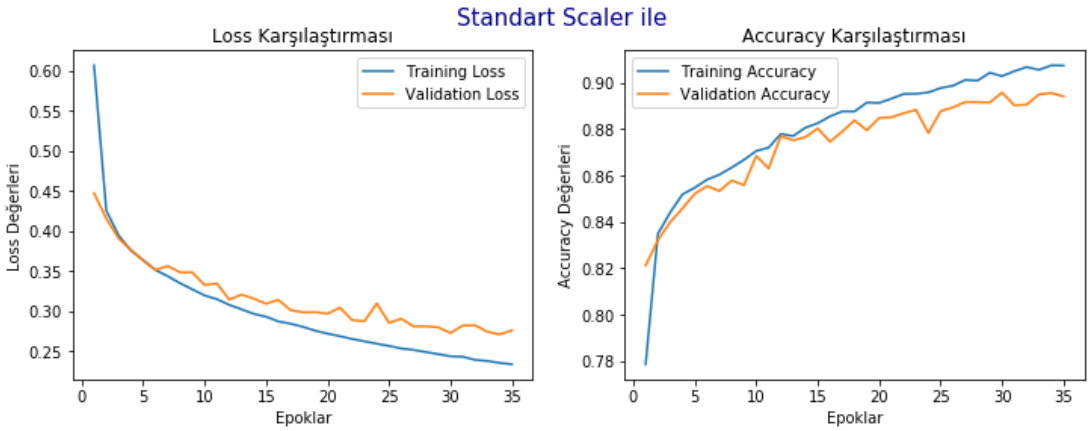
Kullandığım veriseti “Covertype Dataset”. Toplamda 54 adet girdisi(kategorik değişkenlere one-hot encoding yapılmış haliyle) ve bir çıktısı var. Çıktımız kategorik bir değişken yani bir sınıflandırma problemi ile karşı karşıyayız. Yukarıdaki şekilde de görüldüğü üzere girdilerin aralık değerleri birbirine çok yakın değil. Bu durumda önceki edindiğimiz bilgilerden ağımızın ölçeklendirme yapılmadan eğitildiğinde ölçekleme yapılarak eğitilmiş haline göre daha kötü başarım vermesini bekliyoruz. Ağımızı herhangi bir ölçekleme olmadan, Min-max normalizasyonu kullanarak, Standar Scaler kullanarak ve Robuts Scaler kullanarak eğittik. Aşağıda bu eğitimler sonucu oluşan loss ve accuracy grafikleri gözükmektedir.



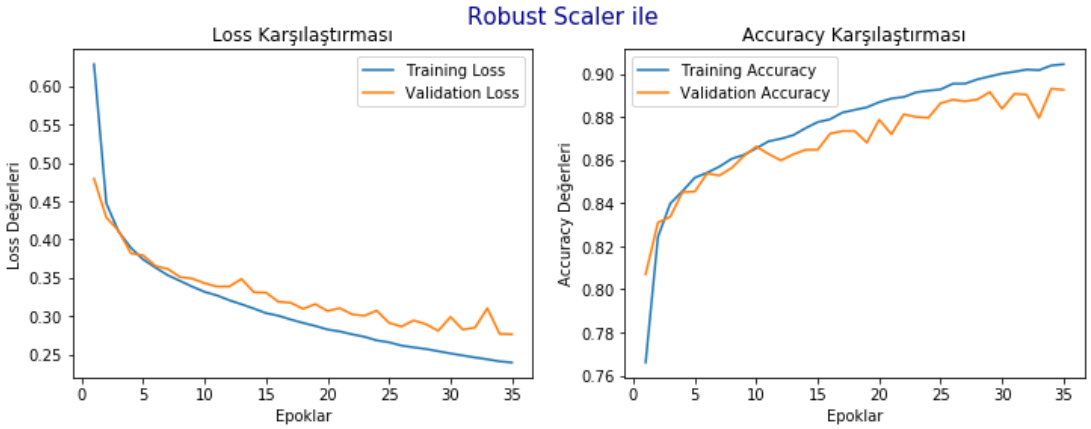

Şekil 2.8. Ölçekleme olmadan




Şekil 2.9. Min-max normalizasyonu yapıldığında




Şekil 2.10. Standart scaler yapıldığında




Şekil 2.11. Robust scaler yapıldığında

Grafikleri yorumlayacak olursak beklediğimiz üzere değişkenler üzerinde hiçbir ölçekleme kullanılmadan eğitilen model en düşük başarımı vermiştir. Normalizasyon yöntemi olan Min-max normalizasyonu, standartizasyon yöntemlerinden daha kötü başarım vermiş, standartizasyon yöntemleri olan Standart Scaler ve Robust Scaler ise birbirine çok yakın başarımlar vermiştir. Burada görmemiz gereken ölçekleme işleminin eğer değişkenlerin değer aralıkları birbirinden farklı ise başarımı iyi yönde etkileyeceğidir. Daha önce bahsedildiği üzere normalizasyon ve standartizasyon yöntemleri arasında seçim yapmak için kesin bir kural yoktur.

Bu alt başlık için son olarak hangi durumlarda bir yapay sinir ağı(sonraki bölümlerde yapay sinir ağını detaylıca inceleyeceğiz) modelinde ölçekleme işleminin yapılıp yapılmayacağını özetleyelim:

Tablo 2.2. Ölçekleme işlemleri ne zaman yapılır?

| Problem Tipi/Ölçekleme Gerekliliği | Girdilere ölçekleme yapılır | Çıktılara ölçekleme yapılır |
|---|---|---|
| **Tek çıktısı olan regresyon problemi** | Eğer girdilerin aralık değerleri birbirinden çok uzaksa | Son katmanda aktivasyon fonksiyonu kullanılmışsa. Örn, sigmoid için min-max(0-1) normalizasyonu yapılmalıdır. |
| **Birden fazla çıktısı olan regresyon problemi** | Eğer girdilerin aralık değerleri birbirinden çok uzaksa | Eğer çıktıların aralık değerleri birbirinden çok uzaksa, Son katmanda aktivasyon fonksiyonu kullanılmışsa |
| **Sınıflandırma problemleri** | Eğer girdilerin aralık değerleri birbirinden çok uzaksa | Hiçbir zaman |

### 2.2.3. Değişken Dönüşümü

Değişken dönüşümü genel olarak makine öğrenmesi algoritmalarının anlayamadağı kategorik verilerin sayısal olarak kodlanarak makine öğrenmesi algoritmalarının anlayabileceği bir biçime dönüştürülmesi işlemidir. Bu işlem çoğu zaman string türünde olan kategorik değişkenlerin 0-1ler veya bir tamsayı ile ifadelmesi işlemidir. Bunun dışında nadiren kullanılan, sürekli bir değişkeni kategorik değişkene dönüştürme işlemini de yapmak mümkündür. Bu başlık altında ele alacağımız değişken dönüşümleri daha önce bahsedilen ölçekleme yönteminde olduğu gibi verinin dağılımını korumaz. Bu bölümde bahsedeceğimiz değişken dönüşümleri yöntemlerini Iris verisetinde bulunan bitkinin türünü belirten bağımlı değişken üstünden örnekleyeceğiz.

| | Sepal Len | Sepal Width | Petal Len | Petal Width | Class |
|---|---|---|---|---|---|
| **0** | 5.1 | 3.5 | 1.4 | 0.2 | Iris-setosa |
| **1** | 4.9 | 3.0 | 1.4 | 0.2 | Iris-setosa |
| **2** | 4.7 | 3.2 | 1.3 | 0.2 | Iris-setosa |
| **3** | 4.6 | 3.1 | 1.5 | 0.2 | Iris-setosa |
| **4** | 5.0 | 3.6 | 1.4 | 0.2 | Iris-setosa |

Şekil 2.12. Iris dataset

```
iris["Class"].unique()
```

```
array(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'], dtype=object)
```

Şekil 2.13. Iris dataset tüm sınıflar

### 2.2.3.1. Label Encoding

Verilen kategorik değişkeni bir sayı değeri ile ifade edilecek şekilde kodlar. Genelde kodlamaya sıfırdan başlayarak her bir farklı sınıf için 1 arttırarak tüm sınıfları kodlarız. Fakat bunu özelleştirerek yapmamızda da sakınca yoktur.

| | Sepal Len | Sepal Width | Petal Len | Petal Width | Class | label_en |
|---|---|---|---|---|---|---|
| **116** | 6.5 | 3.0 | 5.5 | 1.8 | Iris-virginica | 2 |
| **15** | 5.7 | 4.4 | 1.5 | 0.4 | Iris-setosa | 0 |
| **56** | 6.3 | 3.3 | 4.7 | 1.6 | Iris-versicolor | 1 |
| **93** | 5.0 | 2.3 | 3.3 | 1.0 | Iris-versicolor | 1 |
| **74** | 6.4 | 2.9 | 4.3 | 1.3 | Iris-versicolor | 1 |

Şekil 2.14. Label encoding

Iris datasetinin class değişkenine label encoding işlemi uyguladığımızdaki hali yukarıda yeni eklenen "label_en" sütununda gösterilmiştir. Görüldüğü herbir sınıf 0dan başlanarak, 1er atacak şekilde sayı ile gösterilmiştir. Fakat burada büyük bir hata vardır. Makine öğrenme algoritmamız değeri artış gösteren label_en sütununu birbirleri arasında bir sıralama olan (örneğin, değer olarak büyük sayı atanan sınıfın daha değerli olduğunu düşünecektir) bir değişken olarak düşünecek ve buna göre çalışacaktır. Oysaki bizim class değişkenimizdeki tüm değerler(sınıflar) birbirleri arasında bir sıralama ilişkisi olmayan yani nominal değerlerdi. Nominal değişkenleri label encoding ile kodlamak yanlıştır, bu tip değişkenlerin kodlamasını ileriki kısımda anlatılacak olan one-hot encoding yöntemi ile yapacağız. Fakat birbirleri arasında bir sıralama yapılabilen kategorik değişkenleri yani ordinal değişkenleri label encoding ile kodlamamızda bir sakınca yoktur ve olması gereken de budur.

### 2.2.3.2. One-Hot Encoding

Nominal değişkenleri makine öğrenmesi algoritmamıza vermeden önce genelde one-hot encoding dönüşümü uygularız. One-hot encoding sonucu elimizdeki değişkenin sınıf sayısı kadar sütun verisetimize eklenecektir. Örneğin Iris datasetinde class sütununda 3 farklı sınıfımız olduğu için one-hot encoding sonucu 3 farklı değişken oluşacaktır. Herbir sınıf için yapılan kodlamada bu 3 sütundan sadece birinde 1 değeri olacak diğerlerinde 0 değeri bulunacaktır. Bu 1 değeri bulunan sütun ise her farklı sınıf içinde farklı yerde olacaktır. Bu şekilde her sınıfı farklı şekillerde kodlamak mümkün olur. Aşağıda Iris verisetindeki "Class" değişkeni için one-hot encoding yapılması ile oluşan yeni tablo gözükmektedir.

| | Sepal Len | Sepal Width | Petal Len | Petal Width | Class_Iris-setosa | Class_Iris-versicolor | Class_Iris-virginica | class |
|---|---|---|---|---|---|---|---|---|
| **140** | 6.7 | 3.1 | 5.6 | 2.4 | 0 | 0 | 1 | Iris-virginica |
| **11** | 4.8 | 3.4 | 1.6 | 0.2 | 1 | 0 | 0 | Iris-setosa |
| **51** | 6.4 | 3.2 | 4.5 | 1.5 | 0 | 1 | 0 | Iris-versicolor |
| **114** | 5.8 | 2.8 | 5.1 | 2.4 | 0 | 0 | 1 | Iris-virginica |
| **21** | 5.1 | 3.7 | 1.5 | 0.4 | 1 | 0 | 0 | Iris-setosa |

Şekil 2.15. One-hot encoding

### 2.2.3.3. Tuzak(Dummy) Değişken

Dummy variable verisetinde bir bilgiyi ifade etmek için gereğinden fazla değişken bulunmasıdır. Bu durum makine öğrenmesi algoritmasının başarımını olumsuz yönde etkiler. Bu durumu bir örnek üzerinden açıklayalım. Elimizde bireyin cinsiyetini belirten bir değişkene sahip veriseti olsun. Biyolojik olarak bir kişinin cinsiyeti ya kadın ya erkektir. Verisetimizde de sadece bu değerler bulunmakta. Verisetimiz görsel olarak aşağıdaki gibidir.

| | yas | cinsiyet |
|---|---|---|
| **0** | 22 | E |
| **1** | 24 | K |
| **2** | 18 | E |
| 3 | 33 | E |
| **4** | 41 | K |

Şekil 2.16. Veriseti

Makine öğrenmesi algoritmaları(daha da genel konuşacak olursak bilgisayarlar) sadece sayılardan anladığı için buradaki cinsiyet değişkenini sayısal olarak kodlamamız gerekecektir. Cinsiyet değişkeni nominal değişken olduğu için burada one-hot encoding yapacağız. Cinsiyet değişkenine one-hot encoding işlemi uygulunması sonucu oluşan yeni verisetimiz aşağıdaki şekildedir.

| | yas | cinsiyet_E | cinsiyet_K |
|---|---|---|---|
| 0 | 22 | 1 | 0 |
| 1 | 24 | 0 | 1 |
| 2 | 18 | 1 | 0 |
| 3 | 33 | 1 | 0 |
| 4 | 41 | 0 | 1 |

Şekil 2.17. Dummy variable

Giriş kısmında yaptığımız tanımda bahsettiğimiz gereğinden fazla değişken burada "cinsiyet_E" veya "cinsiyet_K" değişkeni olacaktır. Bunu açıklayalım. Sıfırıncı indise ait satırın "cinsiyet_E" değerine baktığımızda bu kişinin erkek olduğunu görüyoruz. Çünkü bu hücrede 1 değeri mevcut. Şimdide 1.indisteki bireyin "cinsiyet_E" sütunundaki değere bakalım. 0 değeri mevcut yani bu kişi erkek değil. Verisetimizde cinsiyet olarak 2 değer tanımlıydı. Biz sadece "cinsiyet_E" satırındaki değere bakarak bireyin erkek olmadığını yani kadın olduğunu anlayabiliriz. Bu durumda verisetindeki diğer bir değişken olan "cinsiyet_K" değişkenini(yada "cinsiyet_E" değişkenini) verisetinde tutmak gereksiz olacaktır. Daha da önemlisi bu durumun makine öğrenme algoritmalarını olumsuz etkilediğinden bahsetmiştik.

Verisetimizde bir de "dogum_yeri" adında bir değişken olsaydı ve 20 farklı değere sahip olsaydı bu değişkene one-hot encoding dönüşümü uyguladığımızda 20 farklı değişken verisetimize eklenecekti. Bu durumda da bir bireyin 19 farklı şehirden birinde doğmadığını biliyorsak o kişinin son seçenek olan 20. şehirde doğduğunu anlayabilirdik. Bu durumda da 1 tane dummy değişkenimizin olduğunda söz edebiliriz. Fakat bu durum gözardı edilebilir bir durumdur ve genelde bu tip durumlarda bir düzenleme yapmayız. Bunu one-hot encoding yaparken kodlama sonucu kullanılacak ideal değişken sayısının değişkenin alabileceği farklı değerler

sayısının 1 eksiği olmalıdır şeklinde kurallaştırmak mümkündür. Fakat söylendiği üzere bu kurala sadece 2 farklı değer alan değişkenlerde uyarız.

#### 2.2.3.4. Sürekli değişkeni kategorik değişkene çevirme

Sürekli bir değişkeni kategorik değişkene çevirme işlemi sık yapılan bir işlem değildir. Bu işlemi Quantile(çeyreklik) değerlerine göre, kullanıcı tarafından belirlenen aralık değerlerine göre gibi farklı yöntemlerle gerçekleştirmek mümkündür.

#### 2.2.3.5. Biri/Birkaçı 1, Diğerleri 0 Dönüşümü

Birden fazla sınıfı bulunan bir kategorik değişkenin, bir veya birkaç sınıfının 1 kalanların 0 olarak kodlandığı değişken dönüşümüdür. Aynı değerle kodladığımız sınıfları bir nevi aynı kefeye koyarız. Aynı değerle kodladığımız sınıfların önem derecesi problemimiz için aynıysa, problemimiz için bu sınıfların farklılığı bir önem arzetmiyorsa kullanmamız mantıklı olacaktır.

### 2.2.4. Değişken(Variable) Seçimi

Verisetlerinde değişkenler kimi zaman onlarca, yüzlerce sayıda olabilir. Bunca değişken arasından problemimizin sonucuna etkisi olan/hatrı sayılır bir seviyede etkisi olan değişkenleri tasarımcının kendi deneyimleri, bilgileri ile seçmesi her zaman mümkün olmayacaktır. Değişken seçimi için kullanılan yöntemlerden bahsedilmeden önce bu yöntemler anlatılırken sıkça duyacağımız H0, H1 ve p-value terimlerinden bahsedelim.

**H0(null hypothesis) :** Farksızlık hipotezi, sıfır hipotezi, boş hipotez olarak da adlandırılır. H0 hipotezi bir konu hakknıda ortaya atılmış olunan bir hipotezdir diyebiliriz. Örneğin bir fabrikada üretilen tüm kurabiyeler 30 gramdır.

**H1(alternatif hipotez) :** H0 hipotezine karşı atılan hipotezdir. Örneğin: bu fabrikada üretilen kurabiyelerin hiçbiri 30 gram değildir veya bazıları 30 gram değildir gibi ifadeler.

**p-value(p-degeri) :** H0 hipotezini çürütmek isteyen H1 hipotezinin doğru olması için bir sınır değer belirler. Genelde 0.05 olarak alınır.

- P0 değeri arttıkça H1'in hatalı olma ihtimali artar.
- P1 değeri arttıkça H0'ın hatalı olma ihtimali artar.

P0 değeri: H0'ı çürütmek isteyen H1'in aşması gereken sınır değeri(p-value)
P1 değeri: H1'I çürütmek isteyen H0'ın aşması gereken sınır değeri(p-value)

Örneğin; 100 kurabiyeden 6 tanesi 30 gram değil ve p = 0.05 olsun. %6 > %5 olduğu için H0 hipotezi çürütülmüş, H1 hipotezi doğru olmuş olacaktır.

H0, H1 ve p-value terimlerini kısaca tanıdığımıza göre değişken seçim yöntemlerini inceleyebiliriz. 5 farklı değişken seçim tekniğinden bahsedeceğiz.

#### 2.2.4.1. Bütün değişkenleri dahil etmek

Değişkenlerin sistemi doğru bir şekilde etkilediğinden eminsek veya veriseti üzerinde keşif yapmak istiyorsak bu yöntemi kullanırız.

#### 2.2.4.2. Geri doğru eleme(backward elimination)

Değişken seçimi için kullanılan bir yöntemdir.
Adımları:

1. Signifance Level(SL) yani kritik p-value belirlenir. Daha önce bahsedildiği üzere genelde 0.05 alınır.
2. Bütün değişkenler kullanılarak bir model inşa edilir.
3. Tüm model için analiz yapılır ve p değerleri hesaplanır. En büyük p değerine sahip değişken için:
    a. P > SL ise adım 4'e gidilir.
    b. P <= SL ise adım 6'ya gidilir.
4. 3. Aşamada bulunan en büyük p değerine sahip değişken sistemden kaldırılır.
5. Makine öğrenmesi algoritması yeni verisetiyle tekrar çalıştırılır ve 3. adıma dönülür.

6. Makine öğrenmesi algoritması sonlandırılır.

#### 2.2.4.3. İleriye Doğru Seçim(Forward Selection)

En düşük p-value'ye sahip değişken bulunarak yeni bir veriseti oluşturulur. P <= SL olduğu sürece yeni verisetine değişkenler eklenmeye devam edilir.

#### 2.2.4.4. Çift yönlü eleme(bidirectioanal elimination)

Hem ileri hem de geriye doğru bir seçim vardır. Yani Backward ve Forward Elimination tekniklerinin ikisi de kullanılır. 2 adet SL değeri kullanılabilir. Bir sisteme eleman eklenirken hem ekleme hem de sistemde kalma için kontrol yapılır.

#### 2.2.4.5. Skor karşılaştırması

Değişken seçimi için kullanılabilecek bir kaba kuvvet metodur.
Adımları:

1. Bir başarı kriteri belirlenir. (herhangi bir kriter olabilir)
2. Bütün olası modeller inşa edilir. n tane değişkenden 2'şer tane alarak model inşa edeceksek C(n,2), tüm olası durumlardaki modelleri inşa edeceksek 2^n - 1 tane farklı model söz konusudur.
3. 1.adımda belirlenen kritere göre en iyi model seçilir ve işlem tamamlanır.

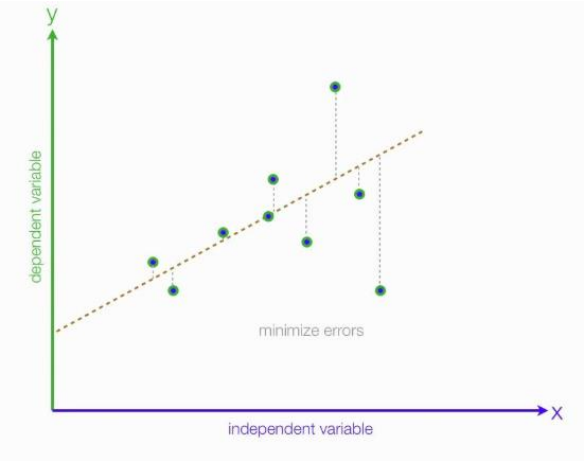
# BÖLÜM 3. MAKİNE ÖĞRENMESİ

Makine öğrenmesi, geleneksel programlama yöntemleri kullanılmadan, çeşitli algoritmalar ile öğrenme yeteneği kazandırılmış sistemler oluşturan, bir yapay zeka alanıdır.(makine öğrenmesi sinan oğuz). Geleneksel programlama yöntemleriyle geliştirilen bir durağan yapay zeka sisteminde, bulanık mantık gibi sistemlerde bir öğrenme söz konusu değildi. Bu tip yöntemlerde sisteme zeka özelliği kazandırabilmek için tasarımcının veriyi tamamen anlamış ve kurallaştırabiliyor olması gerekliydi. İşlenen verinin değişken ve satır sayısı arttıkça, verinin yapısı karmaşıklaştıkca bu yöntemleri uygulamak, bu yöntemlerle üst seviyede genelleştirme yapabilen sistemler geliştirmek zorlaşmaktadır. Bu noktada veriden öğrenebilen algoritmalar olan makine öğrenmesi algoritmaları sorunumuza çözüm olmaktadır. Makine öğrenmesi algoritmalarını kullanırken probleme özgü kurallar yazmayız, daha önceki bölümlerde bahsedilen veri ön işleme süreçlerini uyguladıktan sonra tek yapmamız gereken bu işlenmiş veriyi makine öğrenme algoritmasına vererek algoritmayı çalıştırmak olacaktır. Bu noktada bir veri bilimi projesinde içindeysek veriyle ilgili hiçbir bilgimiz olmasına gerek olmadığı, veriyi yorumlama gerekliliğimiz olmadığı gibi yanlış bir algı oluşmamalıdır. Veriyi anlama, yorumlama gibi işlemler gerçek dünyadaki bir veri bilimi projesinde kesinlikle yapılması gereken aşamalardır ve veri analizi başlığı altında toplanmışlardır.

Makine öğrenmesi algoritmalarının temelinde matematik ve istatistik yatmaktadır. Sonraki bölümlerde bahsedeceğimiz regresyon analizinin temelini istatistiğin temel konularından olan en küçük kareler yöntemi oluşturur. Olasılıksal bir makine öğrenme algoritması olan Naif Bayes Algoritmasının temelinde de yine istatistik(olasılık) alanı altında işlenen Bayes Teoremi yatmaktadır. Hatta makine öğrenmesinin alt dalı olan derin öğrenmede bahsedeceğimiz yapay sinir ağlarının temelini yine istatistiğin bir alt başlığı olan lojistik regresyon oluşturmaktadır.

## 3.1. Regresyon

Regresyon bir egri uydurma problemidir. Elimizdeki girdileri ve çıktıları bir düzlem üstünde hayal edersek regresyon sonucu ortaya çıkan doğru/eğrinin bize girdiler ve çıktı arasında en iyi ilişkiyi veren eğri/doğru olmasını bekleriz.




Şekil 3.1. Lineer regresyon problemi

Yukarıdaki şekilde tek girdisi(bağımsız değişkeni) ve tek çıktısı(bağımlı değişkeni) olan bir regresyon problemine ait grafik verilmiştir. Regresyon probleminde amaç şekilde kesikli çizgiler ile gösterilen doğrunun şekildeki yeşil noktalara uzaklıkları toplamı minimum olacak şekilde oluşturmaktır.

Bu eğri uydurma işlemini daha önce bahsedilen en küçük kareler yöntemiyle kapalı formların çözülmesiyle de yapabilirdik. Fakat bu yöntemi büyük verisetlerinde uygulamak sorun çıkaracaktır. Bağımsız değişkenlerin ağırlık değerlerini hesaplamak için yapılan matrisin tersi alınması işlemini büyük verisetlerinde gerçekleştirmek zorlaşmaktadır. Bazı durumlarda da tersi alınmak istenen matrisin tersi alınmıyor olabilir. Bu tip sorunlardan sıyrılmak amacıyla regresyon problemlerinin amacı olan doğru uydurma işlemini bir optimizasyon algoritması ile yaparız. Konuyu anlatırken makine öğrenmesinin en temel algoritmalarından olan ve diğer optimizasyon algoritmalarının da temel aldığı Gradient Descent(Düşen Eğim) Algoritmasını inceleyeceğiz. Bu algoritma ile yapılan eğri uydurma işleminde en küçük kareler yönteminden farklı olarak bir öğrenme süreci olduğu için(en küçük kareler yönteminde olduğu gibi doğrunun denklemini tek seferde analitik bir yöntemle bulmuyoruz) bu işlem bir makine öğrenme algoritmasıdır. Bu makine öğrenme

yöntemi olan regresyondan ve çeşitlerinden daha da bahsetmeden önce Gradient Descent Algoritmasını anlamaya çalışalım.

## Gradient Descent(Gradyan İnişi) Algoritması

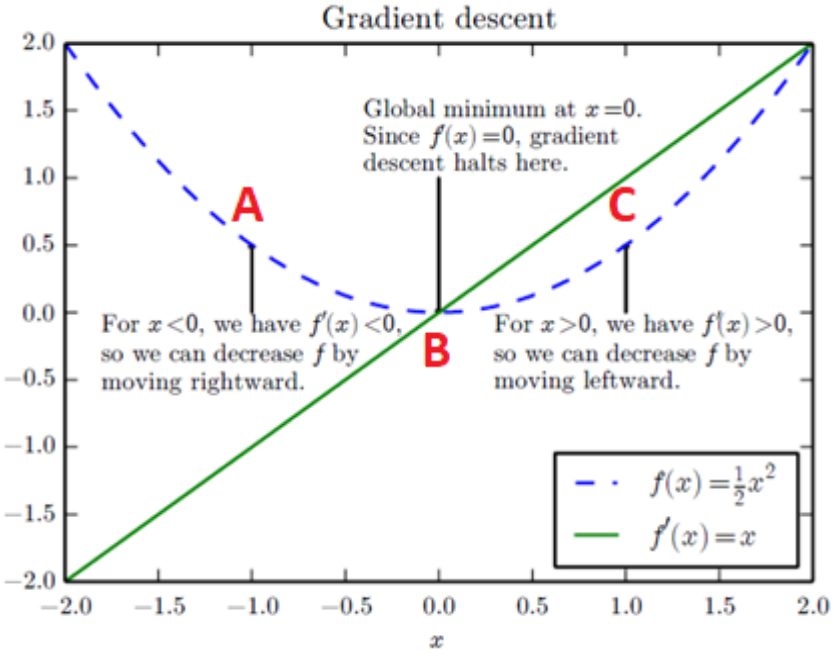
Bir optimizasyon(en iyileme) algoritmasıdır. Bir f(x) fonksiyonunun en küçük yada en büyük olduğu noktayı bulmayı hedefler. En büyütme işlemi sıkça kullanılan bir yöntem değildir, -f(x) fonksiyonunu en küçülterek gerçekleştirilir.

Gradient Descent en iyileme yaparken gradyan vektörünü kullanır. Gradyan vektörü bir f(x,y) fonksiyonunun x ve y değişkenlerine göre kısmi türevlerinin bileşkesi olarak ifade edilir(sinan oğuz makine öğrenmesi). Gradyan vektörünün tersi yönde sürekli giderek fonksiyonun en küçük çıktıyı verdiği noktayı bulmayı hedefleyen işleme gradyan inişi denmektedir. Bu işlem sonucu çoğu zaman en minimum(global minimum) nokta bulunamasa da kabul edilebilir değerlere ulaşıldığında işlem sonlandırılır. Gradient Descent algoritmasının en iyilemeye çalıştığı değer **loss değeridir**. Bu değer bir loss fonksiyonu(hata fonksiyonu, maliyet fonksiyonu) ile hesaplanır. Bir gözetimli öğrenme algoritması ile çalışıyorsak elimizde hedef değişkenimiz(çıktı değişkeni/bağımlı değişken) olacak ve bunun herbir örnek için bir değeri olacaktır. Makine öğrenme algoritmasına bu bağımlı değişkeni göstermeden bağımsız değişkeni vererek bu bağımsız değişkenle aynı/ona yakın bir değer vermesini isteriz. Makine öğrenmesinin bize verdiği bu değere y_tahmin, veri setimizdeki bağımsız değişkenlere karşılık gelen gerçek çıktı değerine de y_gerçek dersek, loss fonksiyonumuz y_gerçek ve y_tahmin arasındaki farkı göz önüne alarak bir loss değeri hesaplayacaktır. Bu fark hesaplaması genelde sadece iki değişkenin birbirinden çıkarılması işlemi olmayıp başka farklı işlemler de içerir. Bu işlemlerin farklılığına göre de farklı loss fonksiyonları oluşmaktadır. Genelde loss fonksiyonları sınıflandırma ve regresyon problemleri için ayrı olacak şekilde gruplandırılmış olsa da bazı loss fonksiyonları her iki problem türü için de kullanılabilmektedir. Bir loss fonksiyonu olan regresyon problemlerinde sıkça kullanılan **MSE**(Mean Squared Error) loss fonksiyonunu kısaca inceleyelim:

$$MSE = \left(\sum_{i=1}^{n}\left(y - h_0(x)\right)^2\right) / n \qquad \textbf{(3.1)}$$

Bahsettiğimiz y_gerçek ve y_tahmin değerleri burada y ve h0(x) olarak karşımıza çıkmaktadır. Formül 3.1'de görüldüğü üzere MSE fonksiyonu gerçek y değerinden, makine öğrenme algoritmasının verdiği tahmin değerini çıkararak bunun karesini alır. Buradaki toplama sembolünde bulunan n ifadesi ise bizim sistemimizde bulunan bağımlı değişken(çıktı) sayısıdır. Tüm çıktılar için gerçek ve tahmin değerleri arasındaki farkın karesi alınıp toplandıktan sonra çıkan sonuç çıktı sayısına bölünür.

Eğittiğimiz parametreler(bağımsız değişkenler) ve loss fonksiyonunu(bağımlı değişken) analitik olan düşündüğümüzde en az 2 boyutlu bir düzlem oluşturacaktır. Optimizasyon algoritmaları ile bu düzlemde oluşan fonksiyondaki minimum değer alan noktayı yani bağımlı değişkenin(loss değerinin) minimum olduğu noktayı bulmak isteriz. Bunu ayarlamak için de optimizasyon algoritması eğitilecek parametleri yani bağımsız değişkenleri sürekli güncelleyerek loss fonksiyonunu minimize etmeye çalışır. Bu güncelleme işlemininin neye göre yapıldığını bir şekil üzerinden anlatalım.



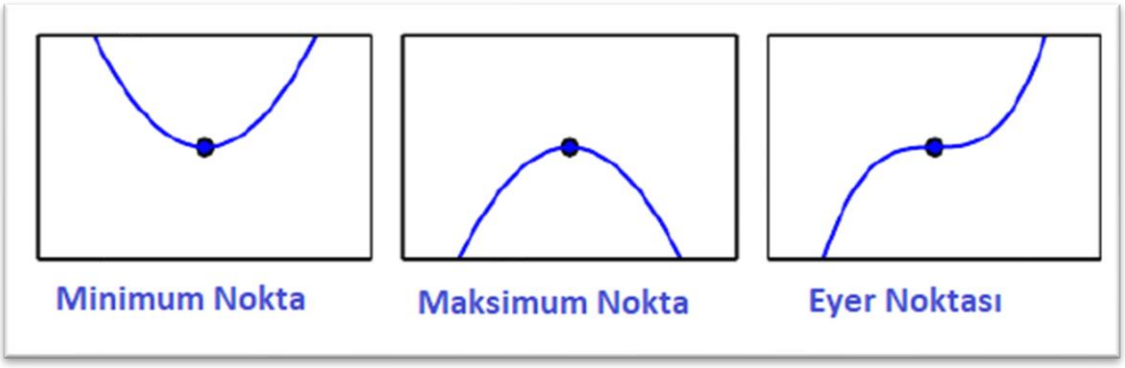

Şekil 3.2. Gradient descent

Gradient Descent algoritmasında sürekli gradyan vektörünün tersi yönde giderek minimum noktayı bulmaya çalıştığımızı söylemiştik. Bu işlemi Şekil 3.2 üzerindeki bazı noktalar üzerinde örnekleyecek olursak:

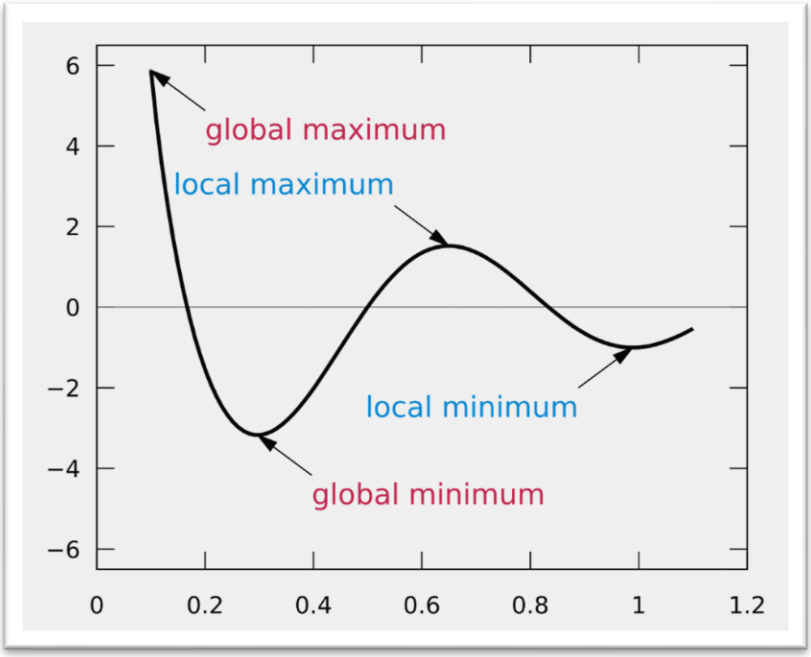
**A noktası için :** Türev negatiftir. Türevin tersi yönde(pozitif yönde) giderek yani sağ tarafa giderek f fonksiyonunun değerini küçültmemiz mümkün olacaktır.

**B noktası için :** Bir kritik noktadır. Kritik nokta çeşitlerinden global minimum noktaya karşılık gelmektedir. Bu noktada f'in türevi 0 olduğundan dolayı türevin tersi yönde gitmemiz de mümkün olmayacaktır yani bu noktada sağa ya da sola doğru bir hareket olmayacaktır. Eğer bu örnekteki gibi bu nokta bizim global minimum noktamız ise bu noktada algoritmamızın tamamlanması bir problem teşkil etmeyecektir hatta çoğu zaman gerçekleşmeyek olan bizim için en iyi durumdur. Kötü olan ise başka bir durumda bu noktanın yerel minimum nokta olabileceğedir. **Yerel minimum nokta** bir fonksiyonun değer aralığı içindeki en küçük nokta değil de sadece belirli bir kısmı için minimum noktası yani eğiminin 0 olduğu noktadır. Bu noktaya gelindiğinde algoritma bu noktanın yerel mi yoksa global minimum nokta mı olduğunu bilemeyecektir onun için orası sadece minimum noktadır. Algoritmamızın bu noktalarda takılıp modelimizi eğitmeye devam edememesi sorunun çözmek için optimizasyon algoritmaları ağırlık güncellemesi yaparken momentum değişkenini de kullanır. **Momentum değeri,** ağırlık güncellemesi yapılırken ağırlık değişim miktarına kullanılan optimizasyon algoritmasına göre direkt olarak veya bir başka değer ile çarpılarak eklenen 0-1 arası değer alan bir değişkendir. Bahsedilen bu minimum nokta problemini çözmek için kullanılır.

**C noktası için :** Bu noktada türev pozitiftir. Türevin tersi yönde yani negatif yönde giderek fonksiyonu en küçültmeye çalışırız.




Şekil 3.3. Kritik noktalar




Şekil 3.4. Global ve lokal noktalar

Bu noktaya kadar makine öğrenmesi için çok önemli bir konu olan Gradient Descent Algoritmasını daha sözel bir anlatımla işlemeye çalıştık. Bu noktadan sonra bu konuyu daha iyi anlamak ve arkasında yatan matematiksel işlemleri bilmek için Gradient Descent Algoritmasını daha matematiksel bir yaklaşımla inceleyeceğiz.

**Algoritma Aşamaları:**

1. Modelin o anki parametleri ve girdiler kullanılarak bir tahmin değeri hesaplanır. Bu tahmin değerine h(x) diyelim.
2. Kullanılan loss fonksiyonun eğitilen her parametre için kısmi türevini al:

$$\frac{\partial y}{\partial \theta_{j(t)}} J(\theta) \qquad \textbf{(3.2)}$$

3. Kısmi türev alımı sonucu oluşan bu ifadelerdeki bağımsız değişkenlere değerlerini koyarak her bir eğitilen parametre için hata miktarını hesapla.
4. Bu hata miktarlarını kullanarak ağırlıkları güncelle:

$$\theta_{j(t)} = \theta_{(j(t-1))} - \gamma \frac{\partial y}{\partial \theta_{j(t)}} J(\theta) \qquad \textbf{(3.3)}$$

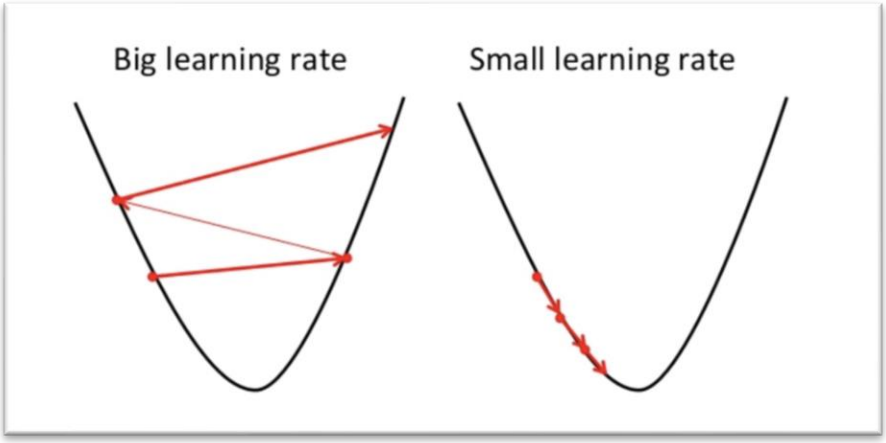
$\gamma$ = Öğrenme oranı, $\theta$ = Eğitilen parametreler, J = Loss fonksiyonu

Bu noktaya kadar bahsedilmeyen bazı önemli terimlerden bahsedelim.

**Öğrenme oranı(γ) :** Genel olarak optimazsyon algoritmalarında eğitilecek parametrelerin dolayısıyla modelin eğitim/yakınsama hızını belirler. Bu değer bahsetmekte olduğumuz algoritma olan Gradient Descent algoritmasında yeni ağırlık değerleri hesaplanırken kısmi türevler sonucu hesaplanan değişim miktarını oranlayarak küçültmektedir. Değer aralığı 0-1 arasında olan, 0'a çok daha yakın bir değer olarak seçilen bu değer modelimizin yakınsama hızını, başaramını etkileyen bir değerdir. Öğrenme oranın çok küçük seçilmesi modelin lokal minimum değerlere takılma ihtimalini yükseltecektir. Ayrıca modelin eğitim süresini de uzatacaktır. Öğrenme oranının gereğinden büyük seçilmesi ise modelin eğitim sırasında salınımlar yaparak optimal noktayı es geçme olasılığını arttıracak, belki de modelin kabul edilebilir bir lokal minimum noktasına veya global minimum noktaya hiç ulaşamamasına neden olacaktır. Optimal değeri modele, probleme göre değişecek bu hiperparametre için bazı değer atama yaklaşımları mevcuttur:

- Öğrenme oranı sabit bir değer olarak(örneğin 0.01) olarak belirlenebilir.
- Öğrenme oranı başlangıçta küçük bir değerle (örneğin 0.01 gibi) başlatılıp belirli bir iterasyon sayısından sonra daha da küçük değerlere(0.001 gibi) düşürülebilir.
- Değeri eğitim esnasında optimizasyon algoritması tarafından değiştirilebilir. Bu işlemin yapıldığı algoritmalar adaptif algoritmalar olarak adlandırılır.

Şekil 3.5'i inceleyecek olursak; sol tarafta yüksek learning rate kullanımı sonucu modelin her optimizasyon algoritmasının her çalıştığı iterasyon sonunda büyük adımlar atarak optimum noktayı es geçtiği, bu noktanın etrafında dolandığı görülmektedir. Sağ tarafta ise düşük değerde learning rate kullanılmış ve model salınımlara yer vermeden, her ne kadar küçük adımlarla dolayısıyla uzun sürede olsa da optimal noktaya doğru ilerlemektedir.




Şekil 3.5. Öğrenme oranının büyüklüğünün etkisi

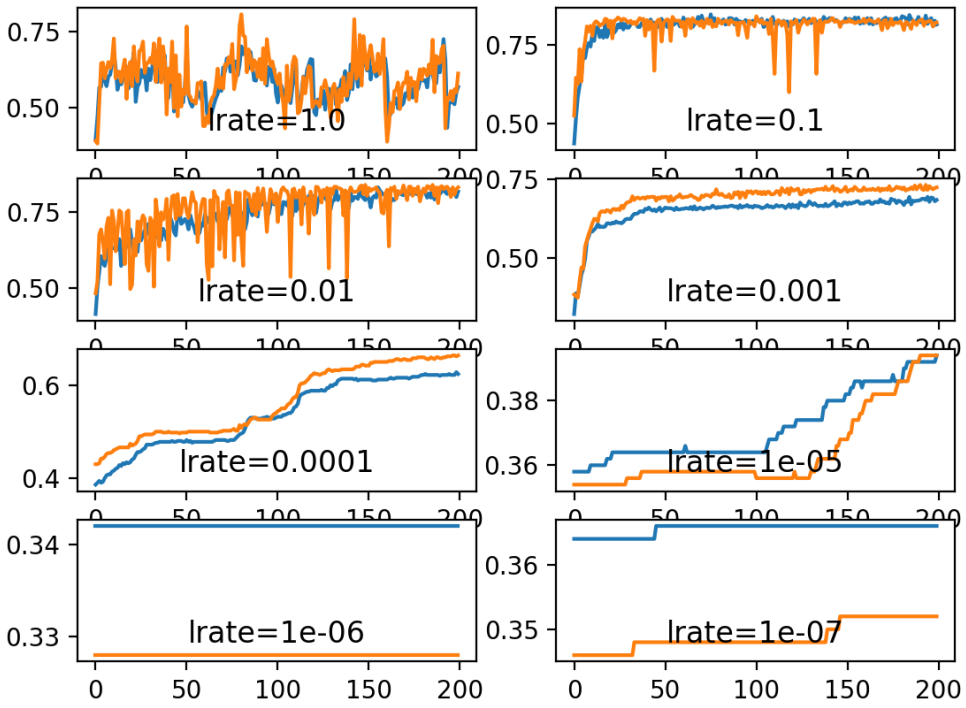
Öğrenme oranını bir de bir başka şekil olan Şekil 3.6 üzerinde inceleyelim. Şekil 3.6'i yorumlanacak olursak; çok yüksek learning rate değerinin(1.0) kullanıldığı durumda modelin loss grafiğinde oldukça fazla salınım olduğu görülmektedir. Model aşırı uydurma problemi göstermediği halde epoch sayısı arttıkça daha önceki epochlara göre daha düşük başarımlar verdiği durumlar bulunmaktadır. Bunun sebebi daha önce bahsedildiği üzere modelin attığı adımın gereğinden büyük olması sonucu optimum noktaları es geçmesi, bu noktanın etrafında dolaşması olarak açıklanabilir.

Öğrenme oranının 0.1, 0.01 ve 0.001 olduğu durumlar her ne kadar birbirine yakın başarımlar vermiş olsa da en idael durum bu şartlar altında learning ratein 0.001 olduğu durum olarak gözükmektedir. Bu değer için model salınımlara yer vermemiş daha stabil şekilde eğitimi gerçekleştirmiştir.

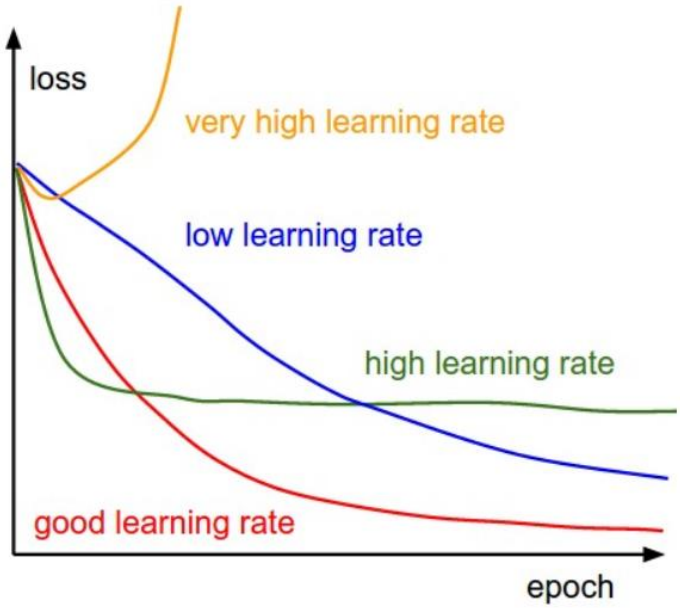
Learning rate değerini 0.0001 ve 1e-05 seçildiği durumlarda model başarım olarak diğer bir çok modelden düşük başarım vermiştir. Daha önce düşük öğrenme oranı seçildiğinde modelin eğitim aşamasının uzayacağından bahsetmiştik. Bu bahsedilen olay learning rate değerinin düşük seçildiği bu durumlarda gözükmektedir. Burada şunu belirtmek gerekir. Epoch sayısı arttıkça bu 2 modelin diğerlerinin ulaştığı maksimum başarıma ulaşması hatta daha iyi başarım vermesi mümkündür. Fakat öğrenme oranın düşük olması lokal minimumlara takılma olasılığını arttıracağı için lokal minimuma takılma sorununun bu modellerde yaşanma olasılığı yüksektir.

Learning rate değerinin 1e-06 seçildiği durumda modelin 200 epoch sonucunda eğitim ve doğrulama başaramında bir değişimin olmadığı yani eğitimin gerçekleşmediği

görülmektedir. Bu noktada modelin lokal minimum noktaya takılı kaldığı yorumunu yapabiliriz.

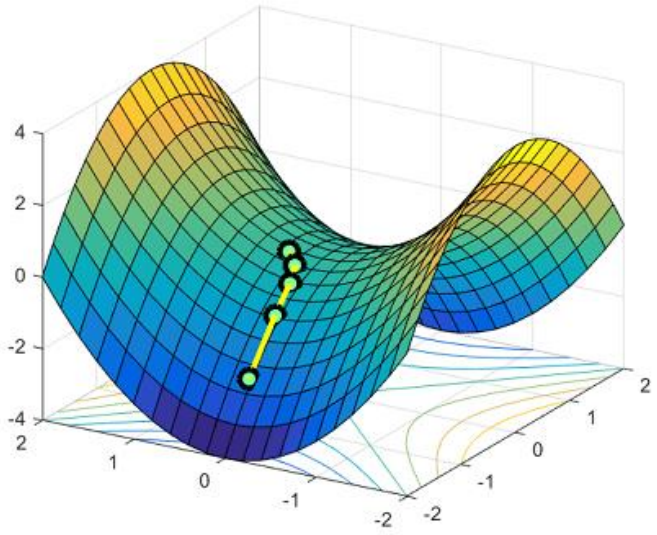



Şekil 3.6. Farklı öğrenme oranlarının modele etkisi




Şekil 3.7. İdeal öğrenme oranı

**Eyer Noktası :** Kritik noktaların listelendiği Şekil 3.3'de görebileceğimiz bu nokta modelimizin eğitim hızına negatif yönde etki yapmaktadır. Şekilden de görüleceği üzere bu nokta yakınlarında eğim çok düşük olduğu için algoritmamızın yapacağı adım büyüklüğü de çok küçük olacaktır. Eyer noktasında diğer kritik noktalardan olan minimum ve maksimum noktalarda olduğu gibi bir takılma söz konusu değildir, sadece modelin eğitimini yavaşlatır.[5] Algoritmada daha önce ilen momentum değerinin kullanılması bu sorunun etkisini azaltmaktadır.




Şekil 3.8. Eyer noktası(saddle point)

Bu noktaya kadar Gradient Descent Algoritmasını anlamaya çalıştık. Gradient Descent algoritması çalışmadan önce eğitilecek model elindeki verisetinde bulunan veriler ile bir çıktı değeri tahmin ediyordu. Bu tahmin değeri ile loss değerleri hesaplanıyor ve gradient descent algoritması bu loss değerlerine göre ağırlıkları güncelliyordu. Gradeint Descent algoritmasının her adımda verisetinin ne kadarlık kısmını ele alarak çalıştığı Gradient Descent algoritmasının farklı varyasyonlarını oluşturmaktadır. Bu 3 varyasyonu alt başlıklar halinde inceleyeceğiz.

**Batch Gradient Descent(Yığın Eğim İniş Algoritması)**

Algoritmanın her iterasyonunda verisetindeki tüm veriler için tahmin değerleri, bu tahmin değerlerinin her biri için 1 loss değeri hesaplanır. Bu loss değerlerinin ortalamasıyla ağırlıklar güncellenir.

$$w = w - (a\nabla_w J(x, y: w)) \setminus n \quad \textbf{(3.4)}$$

n : Verisetindeki Gözlem Sayısı

Batch Gradient Descent algoritmasında yapılan güncelleme sayısı sınırlı sayıdadır. Yapılan güncelleme verisetindeki tüm veriler göz önüne alınarak yapıldığı için genellemesi daha iyi olan bir model, dalgalı olmayan bir eğitim grafiği verir. Çok büyük verisetlerinde tüm satırlar için ileri besleme yapıp hata değerlerinin hesaplanması uzun sürebilir. Bu Batch Gradient Descent algoritmasının bir dezavantajıdır.

**Stochastic Gradient Descent(Rastgele Eğim İniş Algoritması)**

Algoritma her çalıştığında verisetinden rastgele olarak seçtiği bir örnek üzerinden işlem yapar. Yani rastgele olarak seçilen bir örnek için tahmin değeri/değerleri hesaplanır, bu tek örnek için loss fonksiyonu hesaplanır ve ardından gradient descent algoritması çalışır.

$$w = w - a\nabla_w J(x^i, y^i: w) \quad \textbf{(3.5)}$$

Algoritma her çalıştığında sadece elindeki tek bir veriye ait bilgileri öğrenmeye çalıştığı için her iterasyonda farklı bilgiler öğrenilecek bunun sonucunda da loss grafiğimiz dalgalı olacak, eğitim ve doğrulama başarım metrikleri arasındaki fark(varyans) yüksek olacaktır. Stokastik Gradient Descent algoritmasında güncelleme sayısı bir epok için verisetindeki satır sayısı kadar olacaktır. Bu sayı batch gradient descent ile karşılaştırıldığında çok yüksektir. Bu güncelleme sayısının fazlalığı büyük verisetlerinde modelin daha az epok sayısıyla yakınsamasını sağlayabilir. Fakat algoritmanın çalışma sıklığı çok fazla olduğu için eğitim sırasında hesaplanan parametrelerin hesaplanmasında geçen vakit eğitimin yavaşlamasına da sebep olacaktır. Bu durumlar altında hangi algoritmanın daha az sürede daha iyi skoru

vereceği hakkında net bir şey söyleyemeyeceğim. Fakat batch ve stokastik gradient descent algoritmalarının arasında bir algoritma olan mini-batch algoritmasını kullanarak bu ikilemi çoğu zaman düşünmemize gerek kalmayacağını söylebilirim.

**Mini-batch Gradient Descent(Mini-yığın Düşen Eğim Algoritması)**

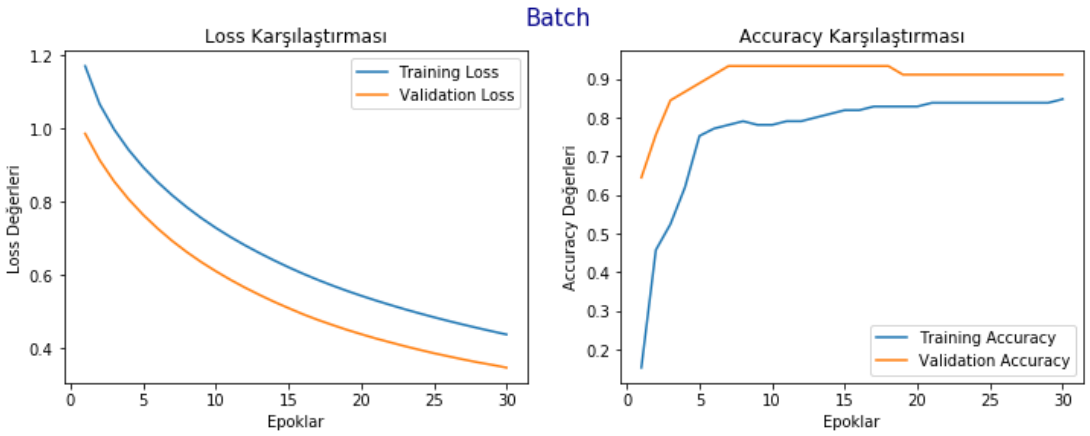
Algoritma her defasında eğitim verisetinden seçilen bir miktar veri için çalışır. Oluşan eğitim grafiğinin düzensizliği stokastik gradient descent algoritmasına göre daha az, batch gradient descent algoritmasına göre daha fazla olacaktır.

$$w = w - (a\nabla_w J(x, y; w)) \setminus x \tag{3.6}$$

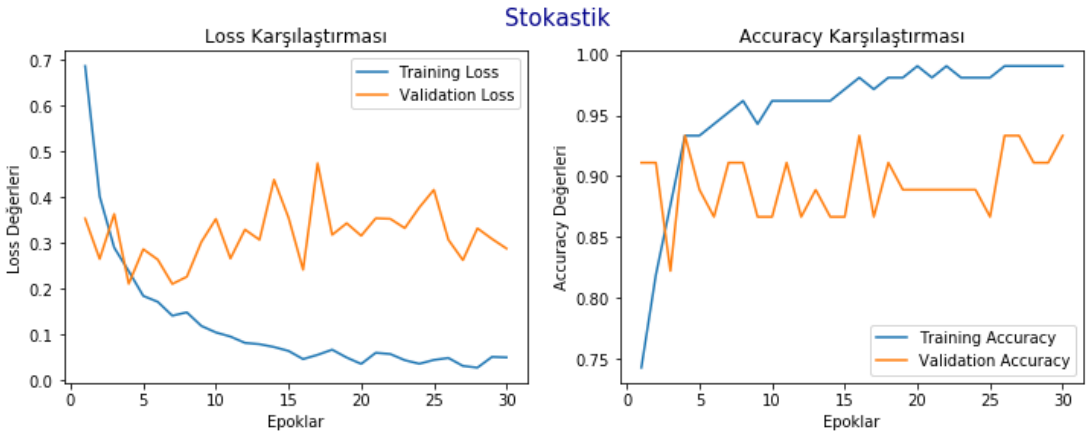
x : Seçilen Gözlem Sayısı

Denklem xx'den görüldüğü üzere mini-batch gradient descent algoritmasında verisetinden rastgele olarak seçilen x gözlem için hesaplanan loss değerlerinin ortalaması alınarak ağırlıklarda güncelleme yapılmaktadır.

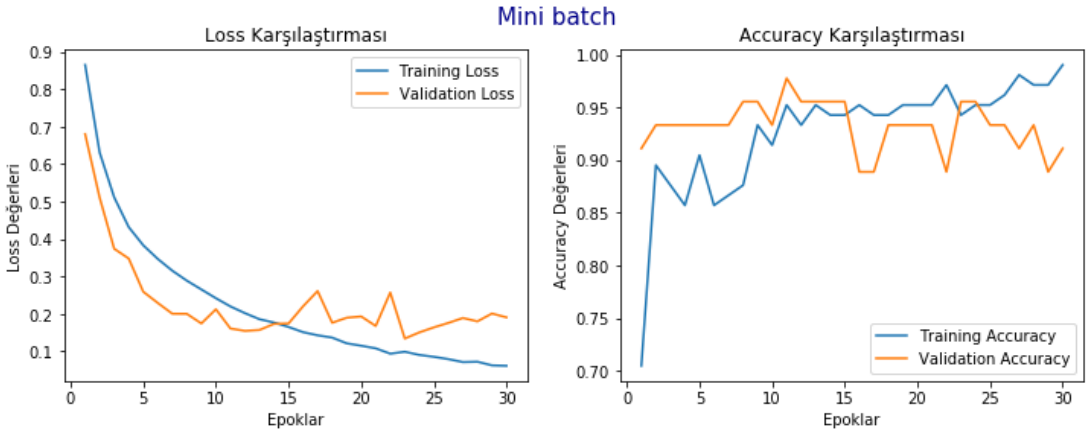
Şekillerde bu 3 algoritmanın eğitim, doğrulama grafiklerinin ve karşılaştırılmalarını görebilirsiniz.



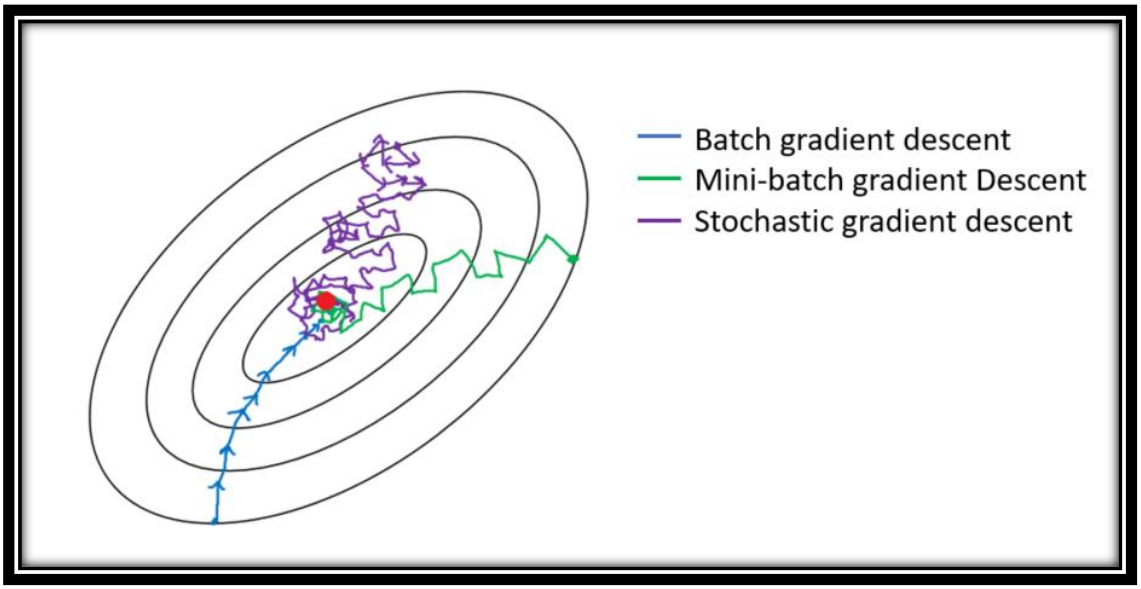

Şekil 3.9. Batch gradient descent grafiği




Şekil 3.10. Stokastik gradient descent grafiği




Şekil 3.11. Mini-batch gradient descent grafiği

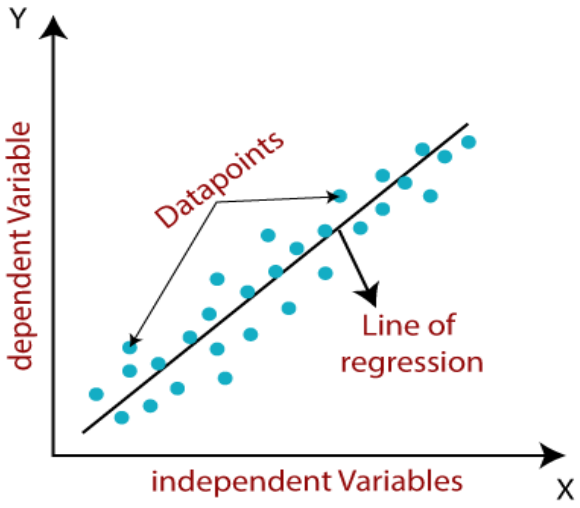



Şekil 3.12. Farklı gradient descent yöntemlerinin karşılaştırılması

### 3.1.1. Lineer Regresyon

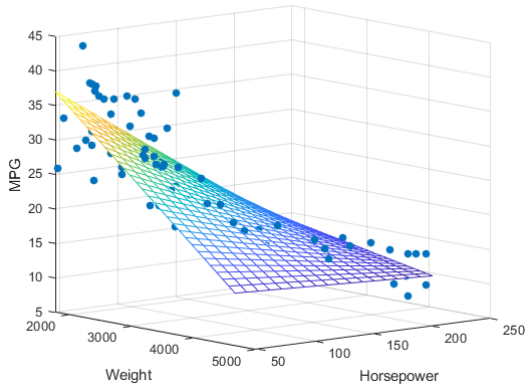
Regresyon yapılacak modeldeki tüm bağımsız değişkenlerin üssü 0 veya 1 olduğu regresyon çeşididir. Modelde kullanılacak veri düzlemde doğrusal bir şekilde dağılmışsa bu regresyon çeşidini kullanmak en ideal senaryo olacaktır.

$$y = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n \quad \textbf{(3.7)}$$




Şekil 3.13. Lineer regresyon

Modeldeki bağımsız değişken sayısı 1 ise **Simple Lineer Regresyon**, 1'den fazla bağımsız değişken varsa **Multi Lineer Regresyon** olarak adlandırılır. Bir bağımsız değişken ve bir çıktıya sahip bir sistemde eğiticeğimiz model y = ax + b şeklinde olup bir doğru olacaktır(bkz Şekil 3.13). Sistemimizde 2 tane bağımsız değişken, 1 tane de bağımlı değişken varsa eğiteceğimiz model y = a + bx1 + cx2 şeklinde olup bir düzlem belirtecektir(bkz Şekil 3.14).




Şekil 3.14. Lineer regresyon

Lineer regresyon çeşitlerinde(aslında tüm doğru uydurma çeşitlerinde) x'lerin katsayıları eğimleri verirken, denklemdeki sabit sayı ise y ekseninde(yani bağımlı değişkenin ifade edildiği eksen) öteleme miktarını vermektedir.

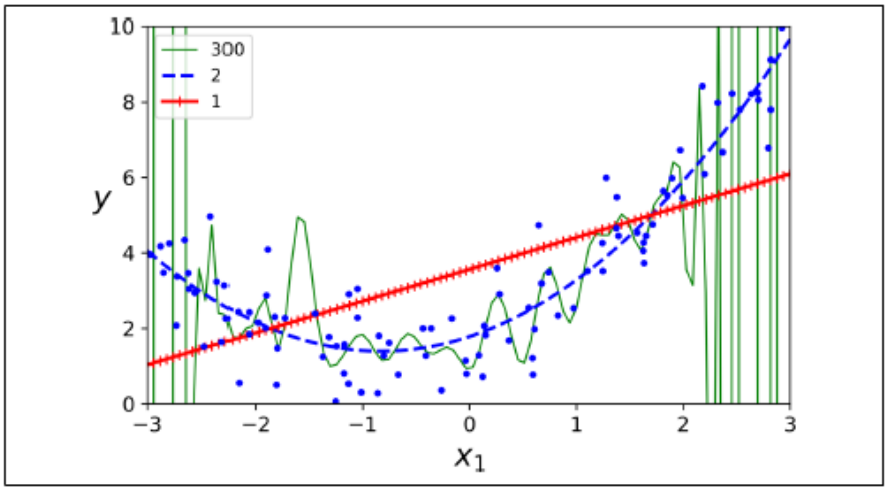
### 3.1.2. Polinomsal Regresyon

Önceki bölümde verimiz düzlemde doğrusal olarak dağılmış bir veriyi nasıl modelleyebileceğimizi gördük. Ne yazık ki gerçek hayatta kullanacağımız veriler genelde düzlemde doğrusal olarak dağılmazlar. Bu tip verileri lineer bir denklem ile ifade etmek başarımı çok düşük, yetersiz modeller oluşmasına sebep olacaktır. Bu noktada bu tip verileri bir lineer denklem ile ifade etmek yerine polinom denklemi ile ifade etmek başarımı yükseltebilir. Lise matematiğinden bildiğimiz üzere polinomlar, değişkenlerinin üssü doğal sayı olan ve birbirleriyle toplama ilişkisinde olan bağımlı değişkenlerin oluşturduğu denklemlerdi.

$$y = w_0 + w_1 x_1 + w_2 x_2^2 + \cdots + w_n x_n^n \quad \textbf{(3.8)}$$

Denklemden görüldüğü üzere polinom regresyonda eğittimiz model lineer olmak zorunda değildir. Eğitilen bir polinom olabilir. Fakat bu işlemi makine öğrenmesi algoritması tarafından inceleyecek olursak durum farklıdır. Eğitilen parametrelerimizin üssel değeri her zaman 1'dir. Polinomsal bir yapıda değillerdir. Dolayısıyla makine öğrenmesi algoritması tarafından bakılırsa polinomsal regresyon da lineer bir modeldir.

**Polinomun derecesi :** Bir polinom denklemindeki bağımsız değişkenlerin sahip olduğu en yüksek üs değeridir. Bu değerin polinom regresyonu yaparken iyi ayarlanması gerekmektedir.




Şekil 3.15. Farklı derecelerdeki polinom regresyonları

Şekil 3.15'deki mavi noktalarla işaretlenen verilerimizin düzlemde doğrusal olarak dağılmadığı gözükmektedir. Böyle bir dağılıma sahip veriyi bir doğruyla (kırmızı çizgi) ifade etmek hata değeri yüksek bir model ortaya çıkmasına sebep olacaktır.

Aynı model derecesi 300 olan bir polinom modeli ile(yeşil çizgi) ifade edildiğinde grafiğe bakarak gayet iyi bir model oluşturulduğu yanılgısına varılabilir. Oluşturulan bu model her ne kadar eğitim verisetinde tanımlı değerler için iyi sonuçlar verse de modelin görmediği(eğitilmediği) başka veriler ile çalıştırıldığında anlamsız değerler verecektir. Çünkü model, verisetini en iyi genelleyecek şekilde oluşturulmamış aksine genellemeden çok uzak bir modeldir. Bu durum makine öğrenmesinde karşımıza çıkan ana problemlerden olup **aşırı öğrenme(overfitting)** problemi olarak karşımıza çıkmaktadır. Klasik bir örnekle bu durum sınava sadece çıkmış sorulara çalışarak(daha doğrusu onları ezberleyerek) giren ezberci bir öğrenciye benzetilebilir. Bu öğrenci bu çıkmış sorular üstünde gayet başarılı olsa da genel bir çalışma yapmadığı için sınavda daha önce görmediği sorular çıktığında bunları doğru bir şekilde yorumlayıp doğru çözüme ulaşamayacaktır.

Model derecesi 2 olan bir polinomla ifade edildiğinde(mavi çizgi) genelleştirme kabiliyeti ve başarımı yüksek bir model ortaya çıktığı gözükmektedir.

Bu örnekten çıkaracağımız sonuç şudur: Polinom regresyonu yaparken polinomun derecesini gereğinden düşük seçersek yetersiz, başarımı düşük bir model; polinomun derecesini gereğinden yüksek seçersek eğitim verisetinde başarımı yüksek ama test

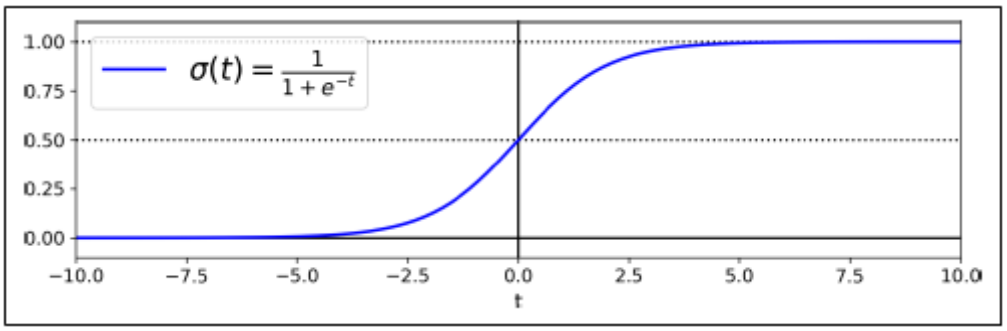
verisetinde genellemeden yoksun bir model elde edeceğimiz bu yüzden polinom regresyonu ile çalışırken polinom derecesini iyi ayarlamamız gerektiğidir.

### 3.1.3. Lojistik Regresyon

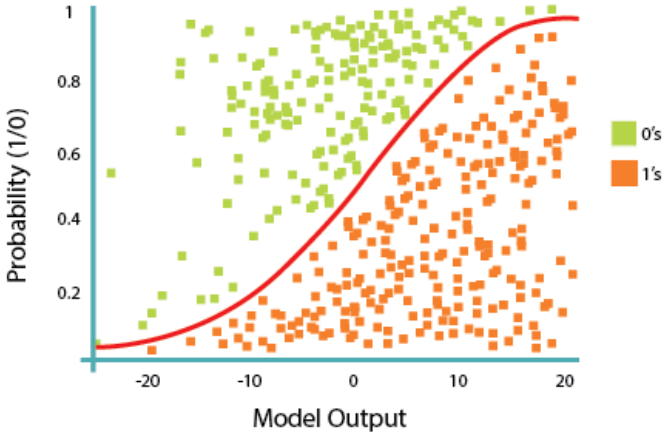
Lojistik regresyon en basit şekliyle açıklanacak olursa düzlemde bulunan ve farklı sınıflardaki 2 grup arasındaki ayrımı yapacak en uygun sigmoid fonksiyonunu(diğer adıyla lojistik fonksiyonu) bulmayı hedefleyen bir makine öğrenme algoritmasıdır. Bir sınıflandırma algoritması olup, yapay sinir ağlarının temelini oluşturmaktadır. Lojistik regresyondaki amaç her ne kadar modele en uygun sigmoid fonksiyonunu bulmak olsa da eğittiğimiz parametreler yine lineer regresyonda olduğu gibi lineer bir denklemin parametreledir. Bundan dolayı lojistik regresyon lineer bir sınıflandırma algoritmasıdır.

$$z = 1 \,/\, (1 + e^{-y}) \tag{3.9}$$

$$y = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n \tag{3.10}$$




Şekil 3.16. Sigmoid fonksiyonu




Şekil 3.17. Lojistik regresyon

Şekil 3.16'da logistic(sigmoid) fonksiyonunu görmekteyiz. Sigmoid fonksiyonuna verilen değer 0'dan küçükse 0.5'den küçük bir değer, büyük ise 0.5'den büyük bir değer vermektedir. Fonksiyonun çıktılarının değer aralığının değişiminin böyle olması onu sınıflandırma yapabilmesine elverişli hale getirmektedir. Fonksiyona 2 sınıfı ayırt etmesi için bir değer verdiğimizde çıktı değeri 0.5 altındakileri bir gruba, 0.5 üstündekileri ise bir başka gruba dahil edebiliriz(Fonksiyon çıktısı 0.5'e eşit olduğunda yapılacak olan işlem ise tasarımcı tarafından belirlenir). Logistic regresyon da tam olarak bunu yapmaktadır.

**Lojistik Regresyonda Kullanılan Loss Fonksiyonu: Log Loss(Binary CrossEntropy)**

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\log(h_\theta(x^{(i)})) + (1-y^{(i)})\log(1-h_\theta(x^{(i)}))]$$

Şekil 3.18. Binary cross entropy

y = Gerçek Çıktı, h(x) = Hesaplanan Çıktı, x = Doğru denkleminin verdiği çıktı, m = Çıktı Sayısı

Log Loss fonksiyonun nasıl hesaplandığı Şekil 3.18'de görülmektedir. Neden böyle bir loss fonksiyonu kullandığımızı inceleyelim.

- y = 1 iken ve h(x) değeri 0'dan 1'e doğru giderken hatanın sıfıra doğru gitmesini isteriz. Yani gerçek y değerimiz 1, bu durumda hesaplanan çıktı değeri 0'dan 1'e doğru yaklaştıkça hata değerimizin de 0'a doğru yaklaşması gerekmektedir.
- y = 0 iken ve h(x) değeri 1 den 0'a doğru giderken hata değerimizin sıfıra yaklaşmasını isteriz. Yani gerçek y değerimiz 0, bu durumda hesaplanan çıktı değeri 1'den 0'a doğru yaklaştıkça yani gerçek y değerine doğru yaklaştıkça hata değerimizin de 0'a yaklaşması gerekiyor

Bu şartları sağlayan bir loss fonksiyonunu logaritma fonksiyonu ile yazmak uygun olduğu için loss fonksiyonu da bu şekilde elde edilmiştir.

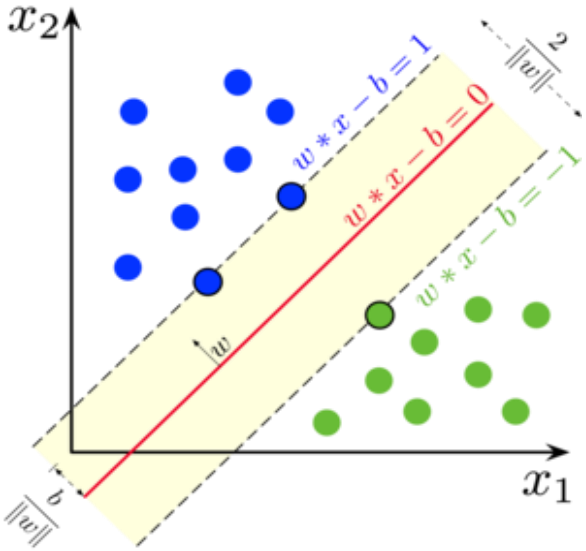
Bu noktaya kadar anlatılan bilgiler ile bağımlı değişkenin değeri sadece 2 farklı değer alan(binary classification) modelleri sınıflandırmak mümkündür. Lojistik regresyon ile 2'den daha fazla sayıda sınıfa sahip modelleri sınıflandırmak için OneVsAll gibi multiclass classification yöntemleri kullanılır.

### 3.1.4. Support Vector Machine

Sınıflandırma (Support Vector Classifier) ve regresyon(Support Vector Regression) problemlerini çözmek için 2 farklı çeşiti bulunmaktadır. Biz Support Vector Machine konusunu Support Vector Classifier üzerinden anlatacak, konun sonunda bir miktar da Support Vector Regression'dan bahsedeceğiz.

#### 3.1.4.1. Support vector classifier

Support vector classifierin en basit halinde 2 farklı sınıf değerine sahip çıktı(etiket/bağımlı değişken) içeren bir verisetini bir doğru/düzlem ile ayırmaya çalışırız. Bu doğru/düzlem verisetinin ifade edildiği düzlemi 2 farklı parçaya bölecek, bu parçaların farklı tarafları farklı bir sınıfı temsil edecektir. SVC'deki amaç sadece düzlemi 2 parçaya bölen bu doğrusal denklemi bulmak değildir. Ayrıca bu doğrunun her iki tarafında bulunan ve eşit uzunlukta olan 2 tane vektörü de eğitmeye çalışırız. Bu vektörler margin vektörü veya destek vektörü olarak adlandırılırlar.




Şekil 3.19. Support vector classifier

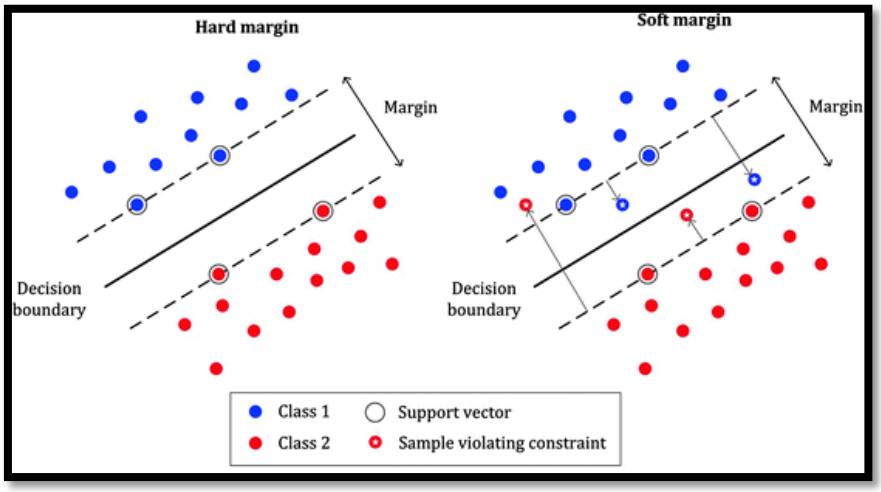
Şekil 3.19'daki 2 boyutlu düzlemde mavi ve yeşil noktalar 2 farklı sınıfı temsil etmektedir. SVC'de amaçlarımız:

1. Bu 2 sınıfı en iyi ayıracak bir doğruyu(kırmızı çizgi) eğitmektir.
2. Eğitilen bu doğrunun her iki tarafında bulunan ve ona eşit uzaklıkta olan destek vektörlerinin ana doğruya olan uzaklığını maksimum tutacak şekilde eğitmektir. Ana doğru ve destek vektörleri arasında örneklerinin bulunup bulunmayacağı hard margin veya soft margin yöntemlerinden hangisinin kullanıldığına bağlıdır.

**Hard margin :** Eğitilen ana vektör ile destek vektörleri arasında veri örneklerinin olmasına izin verilmez. Bu yöntem eğitilecek modeli bazı durumlarda çok fazla kısıtlayabildiği için genelleştirmesi düşük modellerin oluşmasına sebep olabilir.

**Soft margin :** Eğitilen ana vektör ile destek vektörleri arasında veri örneklerinin olmasına izin verilir. Örneğin ana doğrunun üst tarafı A sınıfına ait örnekleri, alt tarafı B sınıfına ait örnekleri temsil etsin. Bu ana doğru ve üst taraftaki destek vektörü arasında B sınıfına ait bir gözlemin bulunmasına izin verilmektedir(Şekil 3.20 sağ tarf). Bu durum bazı örneklerin eğitim veriseti için yanlış sınıflandırılmasına neden olacaktır. Fakat bu yanlış sınıflandırılan verilerin aykırı, gürültülü verilerin olma durumu göz önüne alınırsa modelimizin genelleme kabiliyetinde hard marginde olduğu kadar düşüş olmayacaktır. Aksine modelimiz çok daha iyi genelleme yapacaktır. Fakat yine de ana vektör ve destek vektörleri arasında yanlış sınıflandırılan

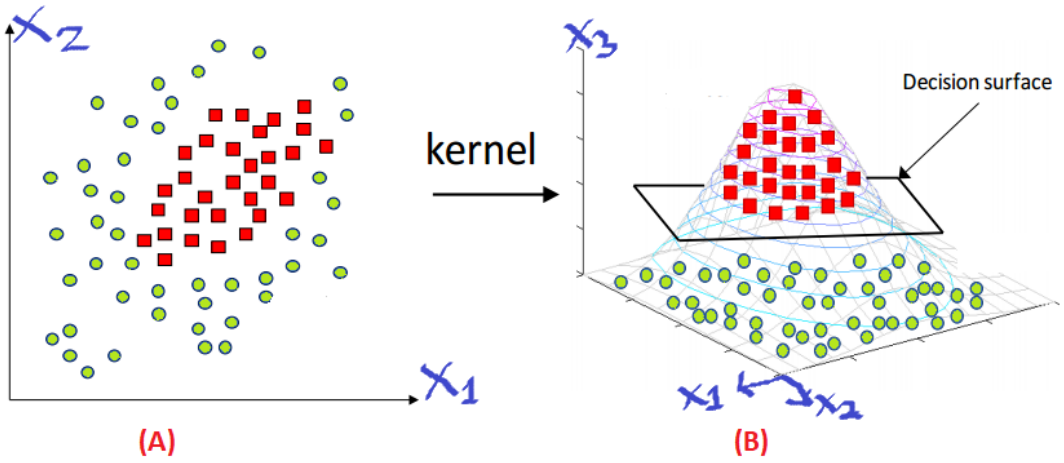
gözlemlerin fazla sayıda olması istediğimiz bir durum değildir. Bu yüzden hard margin yönteminde ayrıca eğitimi modelimizi bu gözlem sayısını minimum tutacak şekilde gerçekleştiririz.




Şekil 3.20. Hard margin ve soft margin

**Doğrusal olarak dağılmayan verilerle çalışmak**

Şuana kadar SVM konusu altında anlatılanlarla lineer olarak dağılım gösteren verileri başarılı bir şekilde modellememiz mümkün olacaktır. Daha önce bahsedildiği üzere gerçek hayatta işleyeceğimiz veriler genelde lineer olarak dağılmazlar. Şekil 3.21 A'daki veri dağılımı lineer olmayan bu dağılıma örnek olarak incelenebilir. Bu örneği inceleyelim. x1 ve x2 bağımsız değişkenlerimiz olarak karşımıza çıkmakta. Çıktımız 2 farklı sınıftan oluşuyor; burada kırmızı ve yeşil noktalar olarak işaretlenmişler. Sadece SVC'de şuana kadar gördüğümüz lineer denklem eğitme işlemini yaparak bu veri kümesini başarılı bir şekilde ayıran doğruyu oluşturmamız mümkün olmayacaktır. Bu verisetini lineer bir denklem ile ayrılabilecek bir şekel dönüştürmek için elimizdeki veriye yeni bir boyut ekleyerek verimizin dağılmını değiştirebiliriz. Bu işlem **yüksek boyutlu uzaya eşleme** olarak adlandırılır. Yüksek boyuta eşleme işleminin matematiksal tanımından bu tezde bahsedemeyeceğim. Fakat şunu söylemem gerekir ki yüksek boyuta eşleme işlemini gerçekleştirmek maliyetli bir işlemdir. Bu yüzden yüksek boyuta eşleme işleminin yaptığı işi yapabilen maliyeti daha az, kestirme bir yöntem olan(adını da bu özelliklerinden almıştır) **Kernel Hilesi(kernel trick)** 'ni kullanırız.




Şekil 3.21. Kernel trick

Şekil 3.21 B'de kernel trick kullanılarak veriye yeni bir boyut eklenerek verinin dağılımı değiştirilmiştir. Yeni dağılımda verimizi lineer bir denklem ile(bir düzlemle) ayırmak mümkün hale gelmiştir. Şekil 3.21 B'de işaret edilen "Decision surface" bu düzlemi ifade etmektedir. Kernel trick uygulanmış bir verisetinde artık problemimiz verisetini en iyi şekilde ayıracak bu doğrusal denklemi(düzlemi) eğitmek olacaktır. Dolayısıyla Destek Vektör Makinelerinde eğitilen denklem lineer bir denklemdir.

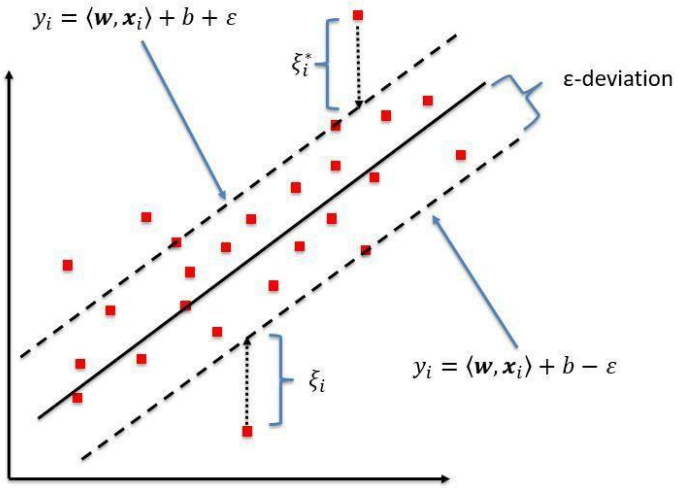
| Kernel | Equation |
|---|---|
| Linear | $K(x, y) = x \cdot y$ |
| Sigmoid | $K(x, y) = \tanh(ax.y + b)$ |
| Polynomial | $K(x, y) = (1 + x \cdot y)^d$ |
| KMOD | $K(x, y) = a\left[\exp\left(\frac{\gamma}{\|\|x-y\|\|^2+\sigma^2}\right) - 1\right]$ |
| RBF | $K(x, y) = \exp(-a\|\|x - y\|\|^2)$ |
| Exponential RBF | $K(x, y) = \exp(-a\|\|x - y\|\|)$ |

Şekil 3.22. Farklı kernel fonksiyonları

### 3.1.4.2. Support vector regression

Support vector regression algoritmasında, SVC'de olduğu gibi bir ana vektör ve ona eşit uzaklıklarda 2 tane destek vektörü bulunmaktadır. Fakat SVC'den farklı olarak SVR'de destek vektörleri arasındaki gözlem sayısının maksimum sayıda olmasını

isteriz. Ayrıca bu destek vektörlerinin ana doğruya(vektöre) uzaklığı da minimum olmalıdır. SVR'de cost fonksiyonu da bu 2 şart altında oluşturulmuş ve bunun sonucunda eğitimin bu şartlar altında gerçekleşmesi sağlanmıştır.



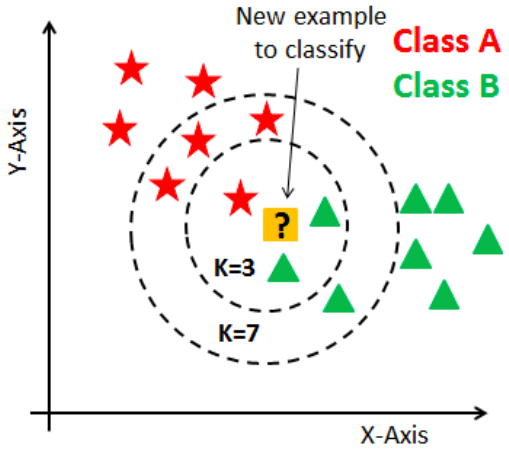

Şekil 3.23. Support vector regressor

Bu noktaya kadar Support Vector Machine konusu matematiksel detaya girmeden anlatılmaya çalışılmıştır. Destek vektör makineleri üzerinde çok fazla çalışılmış, araştırmalar yapılmış, tezler yapılmış, matematiksel arka planınında işlemler daha önce bahsettiğimiz makine öğrenme algoritmalarına göre daha ağır olan bir makine öğrenme algoritmasıdır. Bu yüzden bir yükün, sorumluluğun altına girmemek için Destek Vektör Makinelerinin mantığını ve terimleri anladığım ve çeşitli kaynaklardan onayladığım kısmını anlatmaya çalıştım. Bu konu hakkında daha fazla araştırılması, okunması gerekir. Son olarak SVM'lerin günümüzde hala sıkça tercih edilen, yüksek başarım veren algoritmalar olduğunu belirterek bu alt başlığımızı bitirelim.

### 3.1.5. KNN(K nearest neighbors/En yakın komşular) Algoritması

Hem sınıflandırma hem de regresyon problemleri için farklı 2 varyasyonu bulunan bir makine öğrenme algoritmasıdır. KNN algoritması "Bana arkadaşını söyle, sana kim olduğunu söyleyeyim"(veribilimi okulundan biri bu benzetmeyi yapmış) mantığı ile çalışan bir algoritmadır. KNN algoritması ile bağımsız değişkenler ile bağımlı değişkeninin alacağı değer hesaplanmak istendiğinde verilen bağımsız değişkenleri eğitim verisetindeki tüm(aslında bu durum kullanılan yönteme göre de değişir, her

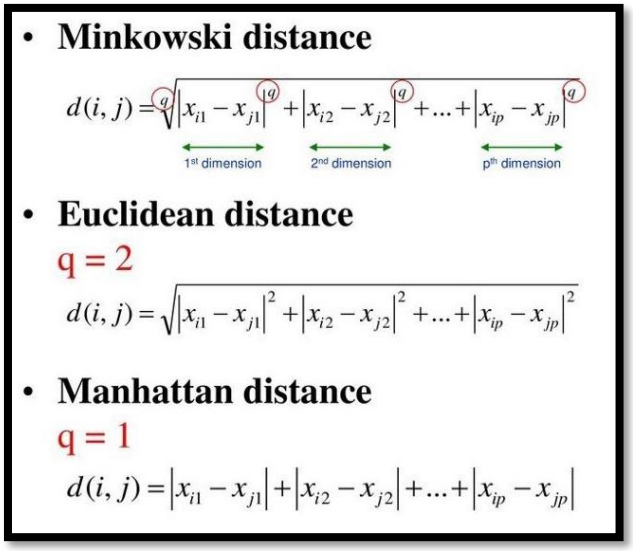
zaman tüm veriseti taranmak zorunda değildir) veriler ile uzakları hesaplanır. Yani burada bir bruteforce mantığı vardır. Hesaplanan bu değerler arasında en küçük K adet değer incelenerek bağımlı değişkenin değeri hesaplanacaktır. Bu K değeri algoritma çalışmadan önce belirlenen bir değerdir. Hesaplanan uzaklıkların en az değere sahip ilk K adedinden(yani en yakın K komşusu, algoritma adını buradan almaktadır), eğer regresyon yapılıyorsa; bu K adet hesaplanan bağımlı değişkenin ortalaması, sınıflandırma yapılıyorsa; bu K adet hesaplanan sınıf içinde en fazla tekrar eden sınıf algoritmanın vereceği tahmin değeri olacaktır. Sınıflandırma probleminde elimizdeki sınıfların tekrar sayıları aynıysa bu sefer sınıflara olan uzaklıklara bakılır ve uzaklıklar toplamı en küçük olan sınıf algoritmanın sunduğu tahmin değeri olur.




Şekil 3.24. KNN algoritması

KNN'in çalışma adımlarını bir de algoritmik olarak listeleyelim:

1. K değerini belirle, uzaklık metriğini ve hesaplama algoritmanı belirle.
2. Tahmin etmek istediğin değerin verisetindeki tüm(?) değerlere olan uzaklıklarını hesapla.
3. Bu uzaklıklar arasından en küçük olan K adet değeri incele. Sınıflandırma yapıyorsan en çok tekrar eden sınıf, regresyon yapıyorsan K adet değerin ortalaması senin tahmin değerin olacaktır.




Şekil 3.25. Uzaklık metrikleri

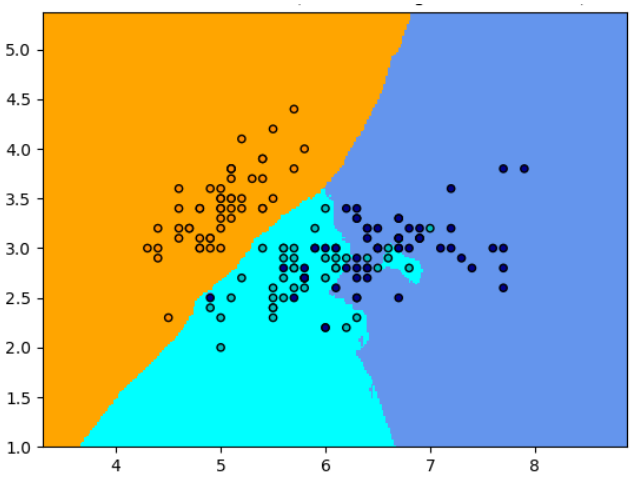
Şekil 3.25'de KNN algoritmasında kullanılabilecek bazı uzaklık metrikleri listelenmiştir. Şekildeki Öklid ve Manhattan uzaklıkları aslında Minkowski Uzaklığı'nın varyasyonlarıdır. Şekilde verilen Manhattan Uzaklığı formülündeki q değeri 1 olarak seçilse Manhattan, 2 olarak seçilirse Öklid Uzaklığı elde edilecektir. KNN algoritmasında uzaklık metrikleri problemden probleme farklı başarımlar vermekte olup tasarımcı tarfından optimal değerinin bulunması gereken bir parametredir.

KNN algoritması **Lazy Learning** veya **Eager Learning** yaklaşımları kullanılarak 2 farklı şekilde uygulanabilir.

**Lazy Learning :** Bir öğrenme süreci söz konusu değildir. Verilen bağımsız parametrelerden bir bağımlı değişken hesaplanmak istendiğinde bu örneğin eğitim verisetindeki verilerin kullanılan yönteme göre tümü veya bir kısmına olan uzaklıkları hesaplanır. Bu hesaplama sonuçlarına göre de tahmin değeri dinamik bir şekilde hesaplanır.

**Eager Learning :** Öğrenme süreci söz konusudur. Eğitim verisetindeki verilerin birbirine olan uzaklıklarına göre bölgeler oluşturulur. Daha sonra model çalıştırılmak istenildiğinde girilen değerlere karşılık gelen bölge içindeki değer algoritmanın vereceği tahmin değeri olacaktır. Şekil 3.26'da bağımlı değişkeni 3 farklı sınıf

değerine sahip bir verisetine(Iris Dataset) eager learning yöntemi ile KNN algoritması uygulanması sonucu oluşan bölgeleri görebilirsiniz.




Şekil 3.26. KNN algoritmasının oluşturduğu bölgeler

### 3.1.6. Naif Bayes Algoritması

Bayes teorimini temel alan bir olasılık bir sınıflandıcıdır(classifier). Naif Bayes Algoritmasına giriş yapmadan önce Bayes Teorimini tanıyalım.

**Bayes Teoremi**

Bayes Teoremi bir olasılık değerini, değerini bildiğimiz diğer olasılık değerlerini kullanarak hesaplama fikridir. Bunu yaparken koşullu olasılığı kullanır. Koşullu olasılık Koşullu olasılığın gösterimi ve neye eşit olduğu Formül 3.11'de görülmektedir.

$$P(A|B) = \frac{P(A \cap B)}{P(B)} \quad \textbf{(3.11)}$$

Koşullu olasılık P(A|B) şeklinde gösterilip bu ifade; B olayının gerçekleştiği durumda A olayının gerçekleşme olasılığını vermektedir. Bu değerde denklemde gösterildiği üzere A ve B olayının aynı gerçekleşme olasılığının, B olayının tek başına gerçekleşme olasılığına bölümüne eşittir.

Bayes teoreminin denklemi ise Formül 3.12'de verilmiştir. Bu formülü inceleyecek olursak B olayının gerçekleştiği durumda A olayının gerçekleşme olasılığı; A olayının

gerçekleşme olasılığı, A olayının gerçekleştiği durumda B olayının gerçekleşme olasılığı ve B olayının gerçekleşme olasılığı kullanılarak hesaplanmıştır.

$$P(A|B) = \frac{P(A)\cdot P(B|A)}{P(B)} \quad \textbf{(3.12)}$$

Formül 3.12'nin çıkarılışı daha önceki bahsettiğimiz koşullu olasılık eşitliğine dayanmaktadır. Formül 3.11'e dayanarak:

$$P(B|A) = \frac{P(B\cap A)}{P(A)} \quad \textbf{(3.13)}$$

denklemini yazabiliriz. Buradaki $P(B \cap A)$ ifadesiyle $P(A \cap B)$ ifadesi birbirine eşittir. $P(B \cap A)$ yerine $P(A \cap B)$ yazarsak ve denklemin her iki tarafını rasyonel ifadelerden kurtarırsak:

$$P(A \cap B) = P(B|A).P(A) \quad \textbf{(3.14)}$$

eşitliğini yazmamız mümkündür. Koşullu olasılık denklemi olan 2.2'deki $P(A \cap B)$ değerinin yerine formül 3.11'deki eşitini yazarsak:

$$P(A|B) = \frac{P(B|A)\cdot P(A)}{P(B)} \quad \textbf{(3.15)}$$

şekline dönüşecektir. Bu da formül 3.12'de verilen Bayes Teoremine eşit olacaktır.

**Naif Bayes algoritmasının özellikleri**

- Bayes teoriminden faydalanarak sınıflandırma yapan bir makine öğrenme algoritmasıdır.
- Veriseti dengesiz dağılmışsa(örneğin bağımlı değişkendeki A sınıfına ait örnek sayısı 10, B sınıfına ait örnek sayısı 100) bu algoritmayı kullanmak mantıklı olacaktır.
- Eager ve lazy learning yapabilir.

Naif bayes farklı varyasyonları da bulunmaktadır:

**Naif Bayes algoritmasını çeşitleri(sklearn kütüphanesinde hazır olarak bulunan)**

1. **Gaussian Naive Bayes:** Tahmin edilen veri sürekli bir değer ise kullanılır.
2. **Multinomial:** Bağımlı değişken nominal ise kullanılır.
3. **Complement:** Metin sınıflandırma işlemlerinde iyi başarım vermektedir.
4. **Bernouilli:** İkili sınıflandırma yapmak için kullanılır(nominal değerler için).
5. **Categorical:** Kategorik verilerde kullanılır. Multinominal Naif Bayes'ten farklı olarak ordinal veriler için de kullanılabilir.

Bu bölüme kadar temel makine öğrenmesi algoritmalarından bazıları anlatıldı. Bunlar dışında kullanılan temel makine öğrenme algoritmalarından bazıları:

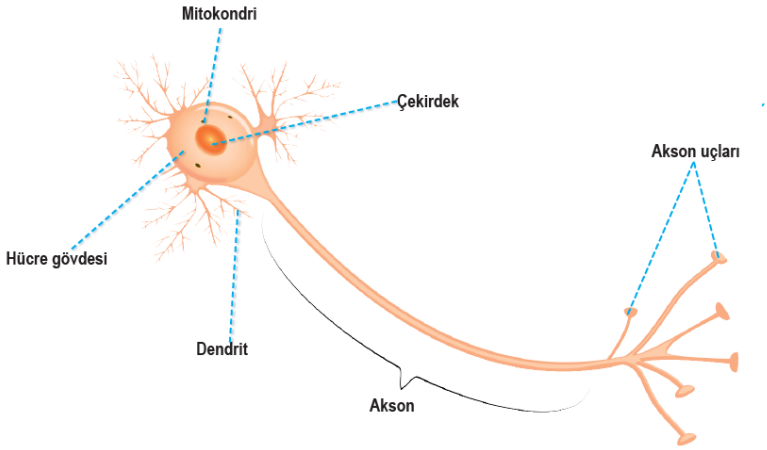
- Ağaç tabanlı algoritmalar: decision trees, bagging trees, random forest, extremly randomized trees
- GBM(Gradient Boosting Machines): güncel olarak kullanılan ve en yüksek başarım veren algoritmalardandır. Genelde ağaçlar üstüne kuruludurlar.
- Diğer regresyon algoritmaları: Ridge Regression, Lasso Regression, Elastic Net Regression
- Gözetimsiz öğrenme algoritmalarından olan: K-Means ve Hiyerarşik Bölütleme algoritması

olarak listelenebilir. Bu tezde daha fazla temel makine öğrenme algoritmaları üzerinde durulmayacak ve buradan sonra tasarlayacağım yazılımda kullanacağım Deep Learning(Derin Öğrenme) teknolojisi anlatılacaktır.

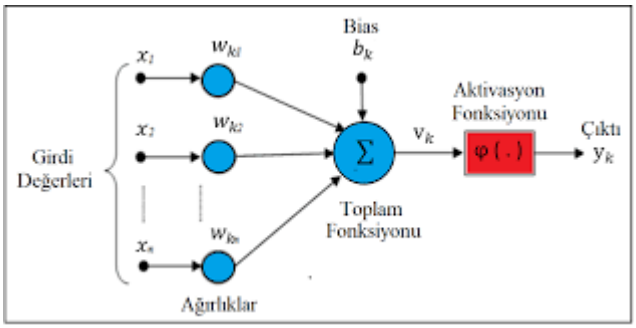
# BÖLÜM 4. DERİN ÖĞRENE

Derin öğrenme makine öğrenmesinin bir alt dalıdır. Bildiğimiz üzere veri ve parametre sayısı arttıkça zayıf yapay zeka teknikleri ile bir makineye zeka niteliğinin kazandırılmasının imkansızlaşmaktadır. Makine öğrenmesi bu soruna bir süre çözüm olmuştur. Fakat veri ve parametre sayısı daha da arttığında makine öğrenmesi ile makineye zeka niteliğinin kazandırılması da zorlaşmaktadır. Makine öğrenmesi modelleri genelde sığ modeller olduğu için büyük ve karmaşık verileri modellemede sorunlar yaşayabilmektedir. Bu noktada derin öğrenme bu soruna çözüm olmaktadır. Sırasıyla yapay sinir ağları, öğrenme, derin öğrenmeden bahsedeceğim. Son olarak yapay sinir ağları ile derin öğrenmenin ilişkisinden/farklarından bahsedeceğim.

1890 yılında William James nöral öğrenme sürecini tanımlamıştır. Bu sürecin matematiksel model olarak tanımlanması ve üzerinde denemeler yapılması ancak 1943 yılından sonradır. Yapay sinir ağı insanın öğrenme mekanizmasını matematiksel olarak modellenmesi amaçlanarak tasarlanmıştır. İnsanda bu olay nöronlar vasıtası ile gerçekleşir dolayısıyla burada modellenen yapı nöronlardan oluşmaktadır. Nöronlar dentritler, sinaps, hücre çekirdeği ve aksondan oluşur. Nöronların matematiksel olarak modellenmiş halinde dentritleri veri girişinin alındığı elemanlar olarak nitelendirebiliriz. *Aslında yapay sinir ağlarındaki her birimin yaptığı(fully connected layerlardaki) bir lojistik regresyon işlemidir diyebiliriz. Lojistik regresyon birimlerinin birden fazla sayıda birbirine bağlı şekilde kullanılmasıyla yapay sinir ağları oluşur dememiz yanlış olmayacaktır. Yapay sinir ağlarının insanın öğrenme sürecine benzetilmesi ise insan beynindeki nöronların birbirine bağlı bir şekilde çalışması, girişleri çıkışları olması gibi benzerlikler olabilir.*




Şekil 4.1. Nöronun Biyolojik Yapısı




Şekil 4.2. Yapay Nöronun Yapısı

Sinapslar ise her bir veri giriş ucu yani dentrit için ağırlık değeridir. Dentritlerden gelen veri bu ağırlık değeri ile çarpılır. Bulunan bu değerler toplama işlevine verilir.

| | |
|---|---|
| Toplam<br>$\text{Net} = \sum_{i=1}^{N} X_i * W_i$ | Ağırlık değerleri girdiler ile çarpılır ve bulunan değerler birbirleriyle toplanarak Net girdi hesaplanır. |
| Çarpım<br>$\text{Net} = \prod_{i=1}^{N} X_i * W_i$ | Ağırlık değerleri girdiler ile çarpılır ve daha sonra bulunan değerler birbirleriyle çarpılarak Net Girdi Hesaplanır. |
| Maksimum<br>$\text{Net} = \text{Max}(X_i * W_i)$ | n adet girdi içinden ağırlıklar girdilerle çarpıldıktan sonra içlerinden en büyüğü Net girdi olarak kabul edilir. |
| Minimum<br>$\text{Net} = \text{Min}(X_i * W_i)$ | n adet girdi içinden ağırlıklar girdilerle çarpıldıktan sonra içlerinden en küçüğü Net girdi olarak kabul edilir. |
| Çoğunluk<br>$\text{Net} = \sum_{i=1}^{N} Sgn(X_i * W_i)$ | n adet girdi içinden girdilerle ağırlıklar çarpıldıktan sonra pozitif ile negatif olanların sayısı bulunur. Büyük olan sayı hücrenin net girdisi olarak kabul edilir. |
| Kumilatif Toplam<br>$\text{Net} = \text{Net(eski)} + \sum_{i=1}^{N} X_i * W_i$ | Hücreye gelen bilgiler ağırlıklı olarak toplanır. Daha önce hücreye gelen bilgilere yeni hesaplanan girdi değerleri eklenerek hücrenin net girdisi hesaplanır. |

Şekil 4.3. Bazı Toplama İşlevleri

Toplama işlevinde kullanılan işleve göre farklı işlemler yapılır. Gelen tüm değerler toplanabilir, verilerin maksimum veya minimum değeri alınabilir, çoğunluk ve normalleştirme algoritmaları kullanılabilir. Fakat genelde toplama işlevinde toplama işlemi yapılır. Toplama işleminden çıkan sonuç aktivasyon fonksiyonuna verilir. Aktivasyon fonksiyonunu nöron çekirdeğinin matematiksel karşılığı olarak düşünebiliriz. Farklı aktivasyon fonksiyonu çeşitleri bulunmaktadır. Aktivasyon fonksiyonuna gelen değer fonksiyona girdi olur, fonksiyondan çıkan değer ise bizim nöronumuzun çıktısıdır. Bu çıktı aksonlar aracılığıyla diğer nöronlara iletilir.

Şuana kadar bahsettiğimiz yapay sinir ağları girdi ve çıktı katmanlarından oluşan 2 katmanlı bir yapıya sahipti. Bu yapı ancak basit ve lineer problemlere çözüm getirebiliyordu. Kullanımı çok yaygın değildi. Dönemin şartlarında yeni modeller oluşturulsaydı bile bu modelleri eğitebilecek donanım gücü ve veri yeterli değildi. Gelişen teknolojinin sonucu olarak veri sayısının ve donanım gücünün artması ile birlikte yeni ve daha çok katmanlı yapay sinir ağı modelleri geliştirilmesinin yolu açıldı. Sadece girdi ve çıktı(veya bunlar dışında birkaç ara katman) katmanlarından oluşan modellere ara katmanlar eklenerek derin sinir ağları(DNN) oluşturuldu. Derin sinir ağlarının ve diğer derin öğrenme modellerinin kullanılarak makine öğrenmesinin gerçekleştirilmesine Derin Öğrenme denir. Derin öğrenme finans, hesaplamalı biyoloji, enerji üretimi, üretim sektörü, doğal dil işleme, güvenlik, sağlık gibi alanlarda kullanılmaktadır.

Bir derin öğrenme modelinde farklı türde katmanlar bulunabilir. Bazı katman türleri aşağıda listelenmiştir.

- Fully Connected Layers
- Convolutional Layers
- Pooling Layers
- Recurrent Layers
- Normalization Layeres

Fully Connected Layerların katmanlarında bulunan nöronların her birinden sonraki katmandaki nöronlara bağlantı vardır. Kullanım alanları çok çeşitlidir. Hemen hemen

her yapay sinir ağı modelinde özellikle son katmanlarda kullanılır. Konvolüsyonel katmanlarda sinir ağları bulunmaz, bu katmanlarda resim matrisleri üzerinde bir veya birden sayıda filtre ile konvolüsyon işlemi yapılır. Bu katman türü ve birazdan bahsedeceğim pooling katmanları bilgisayarlı görüde çokça kullanıldığı için daha sonra detaylıca açıklamaya çalışacağım.

Havuzlama katmanlarında(pooling layers) önceki katmandan girdi olarak gelen verilere filtreleme uygulanarak işlenecek verinin sayısının düşürülmesi amaçlanmaktadır. Recurrent Layers'lar ise daha çok daha çok doğal dil işleme, konuşma tanıma, diller arası çeviri gibi alanlarda kullanılır. Normalizasyon katmanlarında ise veriler normalize edilir. Bazı ağlarda bu işlem isabet oranını arttırabilir.

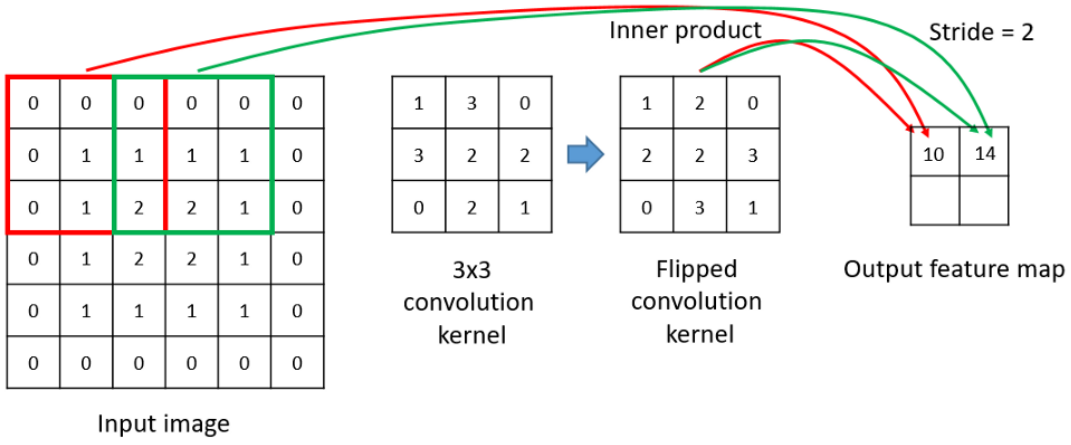
### 4.1. Konvolüsyonel Sinir Ağları(CNN)


Şekil 4.4. Yatay şekil kullanım örneği

Klasik yapay sinir ağlarında görüntüden bir bilgi çıkarılmak istendiğinde resim tek katmana indirgenerek tek boyutlu bir matris haline(vektör) getirilerek yapay sinir ağının nöronlarına direkt olarak verilir. Bunun sonucu resmin 2 boyutlu yapısı bozulmuş olur ve veri kaybı yaşanır. Bu probleme Yann LeCun isimli mühendis çözüm bulmuştur. Yann LeCun konvolüsyonel katmanları tanımlamıştır.

Konvolüsyon işlemini, konvolüsyonel katmanlarda gerçekleştiririz. Daha önce anlattığımız üzere bilgisayar resimi Şekil 4.4'de gösterildiği gibi matris formunda tutuyordu. Kullanılan renk uzayına bağlı olarak katman sayısı kadar da matrisimiz

vardı. Klasik yapay sinir ağı modellerinde bu verileri tek boyutlu bir vektöre indirgediğimizde ciddi veri kayıpları oluşuyordu. Konvolüsyon işlemini yaparken resmi olduğu biçimde alırız. Herbir katmanda belirlediğimiz boyutta ve sayıda olan filtreler ile konvolüsyon işlemini uygularız. Konvolüsyon işlemine giriş matrisimizin ilk katmanının sağ üst köşesinden başlanır. Belirlediğimiz filtrenin elemanları ile giriş matrisimizin filtre işlemine tabi tuttuğumuz elemanları tek tek çarpılır. Çıkan sonuç çıktı matrisine yazılır.




Şekil 4.5. Resimlere uyguladığımız konvolüsyon işlemi

Daha sonra filtre belirlenen adım sayısı yani Stride kadar sağ kaydırılır. Şekil 4.5'de stride sayısı 2 olarak belirlendiği için filtre 2 adım sağ kaydırılmıştır. Bir sonraki adımda da sağa kayması gerekecektir fakat giriş matrisinin boyutları bunun için yetersizdir. O satır için daha fazla filtreleme işlemi yapılamayacağı görüldüğünde belirlenen stride sayısı kadar satır atlanır ve bir sonraki adıma geçilerek filtreleme işlemine buradan devam edilir. Şekil 4.5'de stride sayısı 2 olduğu için 2 satır atlanacak. Giriş matrisinin uzunluğu, filtre matrisinin uzunluğu ve stride sayısı parametrelerine bağlı olarak çıkış matrisinin hesaplanmasına yarayan formül aşağıdadır:

$$U = (N - F) / \text{Stride} + 1 \qquad \textbf{(4.1)}$$

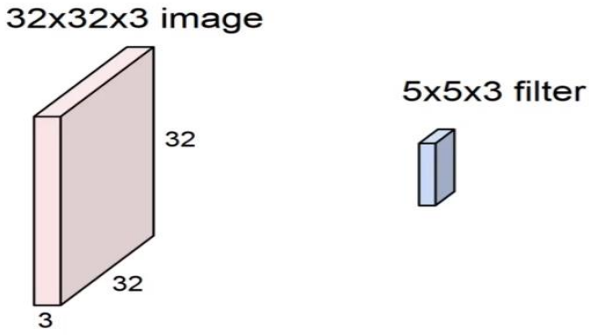
U burada çıkış matrisinin boyutu, N giriş matrisinin boyutu, F uygulanacak filtre matrisinin uzunluğudur. Şekil 4.5'deki örnekte padding ekleme işlemi yapılmıştır. Farklı padding ekleme yöntemleri vardır. Bu örnekte giriş matrisinin tümünü çevreleyecek şekilde 0 değerleri eklenerek 6x6 boyutunda bir matris elde edilmiştir.

Asıl giriş matrisi göbekte bulunan sıfırlardan farklı 4x4 boyutunda bir matristir. Bu padding yöntemi için eklenecek padding uzunluğunu (F-1)/2 formülü ile buluruz. Gerçekten sağlamasını yaparsak (3-1)/2 den 1 değerini bulacağız. Şekil 4.5'de de 1 birim uzunluğunda bir çevreleme yapılmıştır.

Konvolüsyon işlemi yaparken uygulayacağımız filtrenin orta elemanınn filtre uygulanacak öznitelik haritasının başlangıç noktası üzerine gelmesini isteriz. Bunun yanında öznitelik haritamızın boyutunun azalmamasını isteyebiliriz. Bu gibi sebeplerden ötürü padding işlemini uygularız. Bu örnek için giriş matrisimizin orjinali 4x4 boyutunda, filtremiz de 3x3 boyutunda, adım sayımız ise 2'dir. Biz 4x4'lük giriş matrisimizin sağ üst köşesinden başlayarak 3x3 filtre matrisimiz ilk konvolüsyon işlemini yaptıktan sonra stride sayımız 2 olduğu için 2 adım sağa kaymamız gerekecek. Fakat matrisi 2 adım sağa kaydırırsak filtre matrisimizin son sütunu dışarıda kalacak yani boyut uyuşmazlığı olacak. Bu yüzden bu aşamayı es geçip stride sayısı kadar satır atlayıp yine en sağ köşeden konvolüsyon işlemine devam etmemiz gerekecek. Aynı sorunu da burada satırlar için yaşayacağız. Bu sorunun oluşup oluşmayacağını Formül 4.1'de gerekli değerleri koyarak da anlayabiliriz. Bizim Şekil 4.5'deki değerlerimiz için sonucumuz tamsayı çıkmayacaktır, bu bize bir sorun olduğunun habercisidir. Ekleyeceğimiz paddingin uzunluğunu (F-1)/2 formülü ile tanımlamıştık. Daha önce verdiğimiz formül herhangi bir padding işlemi gerçekleşmemiş ve gerek yoksa kullanılabilir. Daha genel olarak çıkış matrisinin uzunluğunun hesaplanmasını sağlayan formül aşağıdadır:

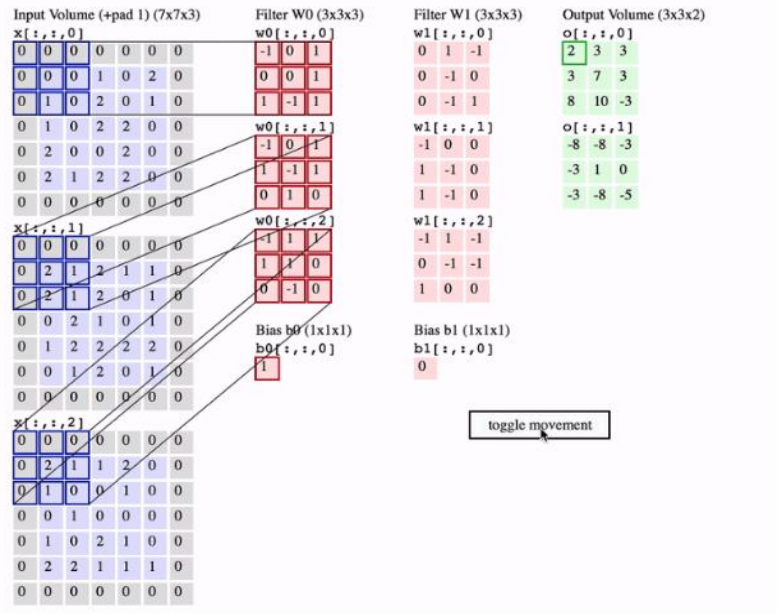
$$C = 1 + \frac{(N+2P-F)}{stride} \qquad P = \frac{(S).(N-1)+(F-N)}{2} \qquad \textbf{(4.2)}$$

Konvolüsyon işleminde girdi olarak alınan resmin ve uygulunacak filtreni katman sayısı aynı olmalıdır. Örnek olarak RGB renk uzayında olan bir resim girdimiz olsun. 3 katmanlı bir matris olacaktır. Uygulayacağımız filtrenin de katman sayısı 3 olmalıdır. Resmin her katmanına filtrenin eşleşen katmanı ile konvulüsyon işlemi yapılır.




Şekil 4.6. Resim ve filtre matrisi örneği

Şekil 4.6'da 32x32 boyutunda 3 katmanlı bir resmimiz vardır. Uygulunan filtre de 5x5 boyutunda ve 3 katmanlıdır. Katman sayılarının aynı olmasının şart olduğundan ve uygulanacak filtrenin boyutunu bizim belirlediğimizi söylemiştim. Buradaki filtre 3x3 boyutunda da olabilirdi. Uygulanacak filtre sayısını da biz belirleriz. 32x32x3'lük giriş matrisimiz, 5x5x3'lik filtremiz olsun, uygulayacağımız filtre sayısı 6, stride sayımız da 1 olsun. Tüm filtreleme işlemleri yapıldıktan sonra çıkış matrisimiz 28x28x6 boyutunda olacaktır. Buradaki 28 değeri Formül 4.1'den hesaplanabilir. Çıkış matrisinin katman sayısının 6 olmasının nedeni ise uyguladığımız filtre sayısının 6 olmasıdır. Yani kaç tane filtre uygulursak çıkış matrisimizin katman sayısı da o olur.




Şekil 4.7. 3 katmanlı bir resme filtre uygulanması

Şekil 4.7 üzerinden yapılan konvolüsyon işlemimi daha detaylıca anlatalım. Elimizde 3 katmanlı bir resim var, her katmana ait değerler 1 birim padding eklenerek alt alta sıralanmış.Onun hemen yanındaki 2 sütunda uygulayacağımız 3 katmanlı filtreler ve bias değerleri var. Son olarak da en sağda çıkış matrislerimiz var. İlk filtremizi resmimize eşleşen katmanlara göre uyguluyoruz. Giriş matrisin en sağından başlayarak konvolüsyon işlemi yapıyoruz. Her adım için her katmanda konvolüsyon işlemi sonucu bulduğumuz değerleri topluyoruz ve buna bias değeri ekliyoruz. Çıkan sonuç bizim çıktı matrisimizin ilk elemanı oluyor. Bu şekilde tüm resmi tarayarak çıktı matrisini elde ediyoruz. Aynı işlemleri diğer filtre ile yapınca da diğer çıkış matrisimi elde etmiş oluyoruz. Çıktı matrisimiz 3x3x2 boyutunda oldu. Çıkış matrisimizin boyutu azaldı, eğer bunun olmamasını istiyorsak işlemlerden önce daha fazla padding ekleyebiliriz.

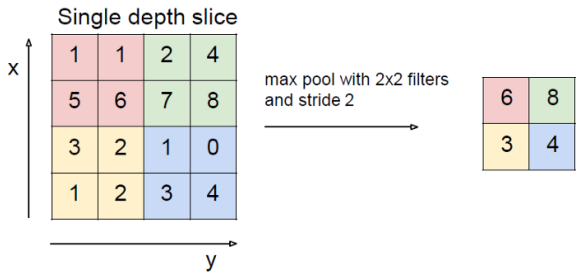
### 4.1.1. Konvolüsyonel sinir ağlarının(CNN) yapısı

CNN'ler genel olarak art arta bağlanmış konvolüsyon katmanlarından oluşmaktadır. CNN'lerin ilk katmanlarında probleme özgü olmayan genel bilgiler öğrenilir. Örneğin bir resim sınıflandırma yapan CNN ağının ilk katmanlarındaki filtrelerin kenar bulan, renklere odaklanan filtreler olması beklenir. CNN ağının son katmanlarındaki filtreler ise daha çok probleme özgü özellikleri öğrenir. Örneğin bir kedinin gözünü, kulağını tespit eden bir filtre probleme özgü olacaktır, her verisetinde bu sınıfın olması ve bu filtrenin iş yapması beklenemez.

Son konvolüsyon katmanından sonra elimizde genelde giriş resminden daha küçük boyutlu ve işlenmiş bir öznitelik haritası olur. Bu öznitelik haritası genelde 3 boyutlu olacaktır ve bir tam bağlı katmana bağlanması gerekir. Bu veri vektör halinde olmadığı için direkt olarak konvolüsyonel ağa bağlanamaz. Bu veri Flatten katmanı eklenerek veya Global Pooling işlemi uygulanarak düzleştirildikten(ayrıca indirgenir) sonra tam bağlı katmana bağlanır. Tam bağlı katman ise bu düzleştirilmiş/indirgenmiş öznitelik haritasındaki değerlere göre bir sonuç verir.

### 4.1.2. Pooling İşlemi

Bu işlem pooling katmanlarında yapılır. Bu katmanda öğrenme yoktur. Amacı giriş matrisinin katman sayısını sabit tutarak parametre sayısını yani boyutunu azaltmaktır. Farklı şekillerde yapılabilir. En genel kullanımı maksimum ortaklama yapmaktır. Hesaplama karmaşıklığını azaltır.




Şekil 4.8. Maksimum ortaklama

Şekil 4.8 üzerinden anlatacak olursak, 4x4 boyutunda olan giriş matrisimize 2x2 boyutunda bir boş matris ile pooling uygulanmıştır. Giriş matrisi 2x2 boyutunda küçük matrislere bölünmüştür. Bu 2x2 matrisin elemanları içinde en büyük olan eleman çıkış matrisine yazılır. Bu örnekte boyut yarıya düşmüştür. Pooling katmanları genelde gerekli görüldüğü yerde konvolüsyon katmanından sonra konulur.

Kanal sayısı sabitken verideki parametre sayısını düşürmek için adım sayılı(stride>1) konvolüsyon işlemi de yapılabilir. Fakat en ideal senaryo stride olmadan yapılan konvolüsyon ve max pooling işemidir. Max poolingin diğer pooling çeşitlerinden daha iyi başarım vermesinin sebebi her defasında her değerli bilgileri seçmesidir.[6] Çıktıya etkisi olmayan veya çok az olan bilgilerden sistemi arındırır.

### 4.1.3. Dropout(İletim Sönümü)

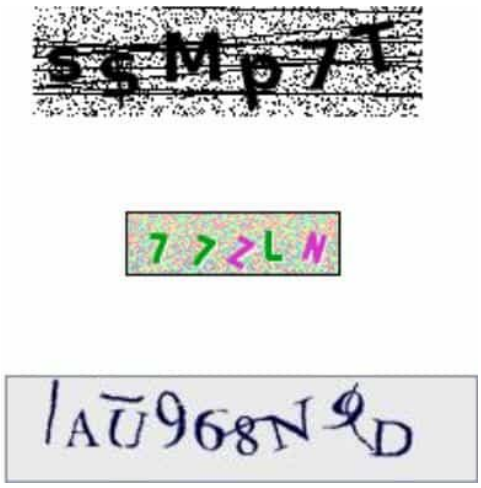
Bir regularizasyon tekniğidir. Aşırı uydurmayı engeller. Bu yöntem bir katmana uygulandığı eğitim sırasında yöntemin uygulandığı katmanın çıktılarının bir kısmı 0 değerini alır yani bir nevi bu nöronları iptal etmiş oluruz. İptal edilen bu nöronlar tasarımcı tarafından belirlenen bir orandadır. Genelde 0.2-0.5 arası seçilir. Hangilerinin 0 yapılacağı ise bir eşik değeri belirlenerek, rastgele olarak veya başka yöntemlerle belirlenir. Eğitim sırasında bazı nöronları iptal ettikten sonra ya test zamanında çıktıları belirlenen dropout oranında küçültürüz yada yine eğitim sırasından iptal etmeden sonra kalan çıktıları dropout oranında büyütürüz.[7] Genelde 2.yöntem

tercih edilir. Keras gibi bir çok kütüphanede dropout uygulamak istediğimizde, dropout uygulamak istediğimiz katmandan sonra bir dropout katmanı ekleriz. Burada sadece dropout oranını vermemiz yeterlidir, diğer işlemler otomatik olarak yapılır.

# BÖLÜM 5. PROBLEMİN TANIMI VE UYGULAMA

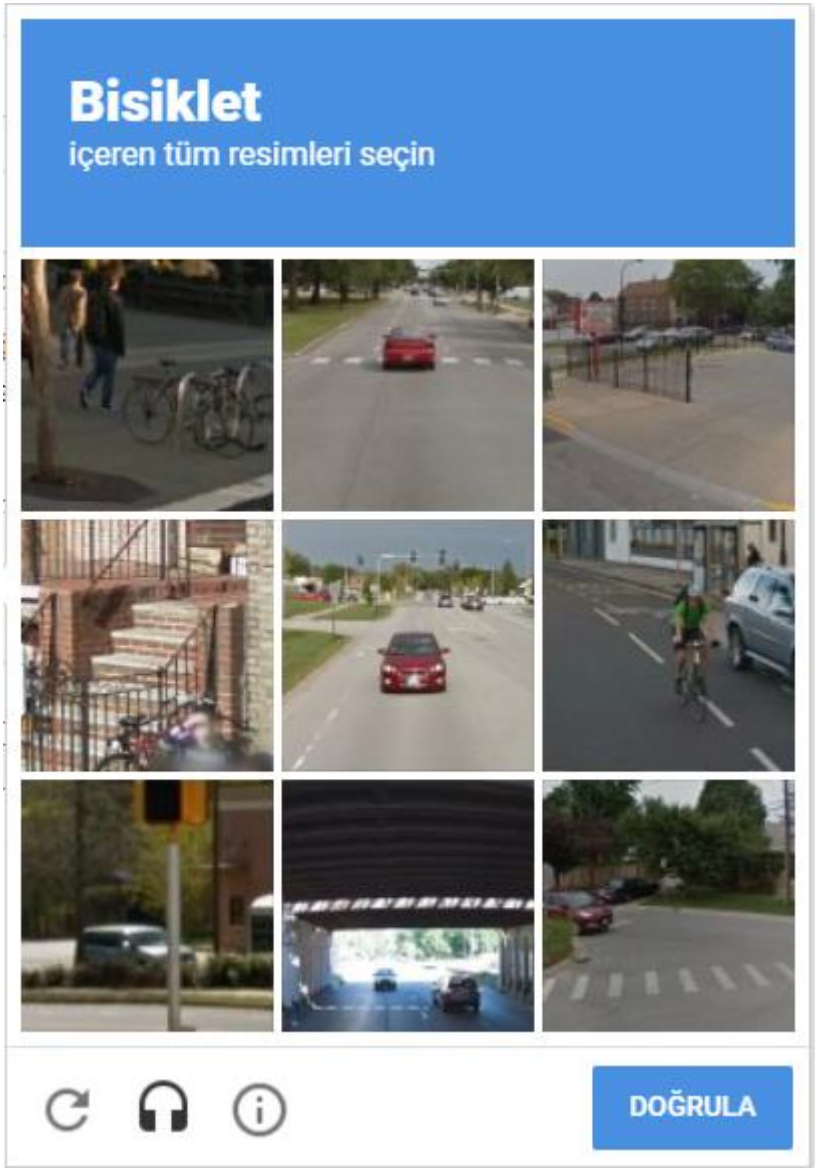
Problem en basit tabiriyle bir CAPTCHA(Completely Automated Public Turing test to tell Computers and Humans Apart/İnsan ve Bilgisayar Ayrımı Amaçlı Tam Otomatik Genel Turing Testi) çözen yapay zeka destekli bir sistem tasarlamaktır. CAPTCHA spam ve şifre çözme koruması sağlanmasına yardımcı olur. Bunun için sizden basit bir testi yanıtlamanızı isteyerek şifre korumalı bir hesaba girmeye çalışan bir bilgisayar değil insan olduğunuzu kanıtlamanızı sağlar. Yapacağım projede HCAPTCHA servisini elimine eden bir Yapay Zeka Projesi geliştirilmesi amaçlanmaktadır. Projede en kritik nokta bu CAPTCHA servisini çözen model ve onun hangi algoritmik adımlarla kullanılacağıdır.

CAPTCHA testleri, deforme edilmiş bir resim şeklinde görünen rastgele oluşturulmuş harf ve/veya rakam dizisi ile bir metin kutusu olmak üzere iki temel parçadan oluşur. Bir testi geçmek ve insan olduğunuzu kanıtlamak için resimde gördüğünüz karakterleri metin kutusuna girmeniz yeterlidir. Günümüzde bir çok firma resimdeki sayıları ve resimleri doğru bir şekilde girerek insan-bilgisayar ayrımını yapan CAPTCHA sistemlerini(bkz. Şekil 5.1) kullanmaktan vazgeçmiştir. Bu gibi CAPTCHA'ları az zamanda, başarımı yüksek bir şekilde çözen sistemler tasarlamak günümüzde artık zor değildir.



Şekil 5.1. Kullanımı artık eskisi kadar yaygın olmayan CAPTCHA çeşitleri

Bunlar yerine günümüzde yaygın olarak verilen fotoğraf veya fotoğraflardaki istenilen nesneyi seçmemizin istendiği, nesnenin bulunduğu sınırları çizmemizin/seçmemizin istendiği sistemler yaygın olarak kullanılmaktadır. Bunlardan en yaygın olanı Google ReCAPTCHA sonrasında ise son zamanlarda kullanımı oldukça yaygınlaşan ve yaygınlaşmaya devam eden HCAPTCHA servisidir.



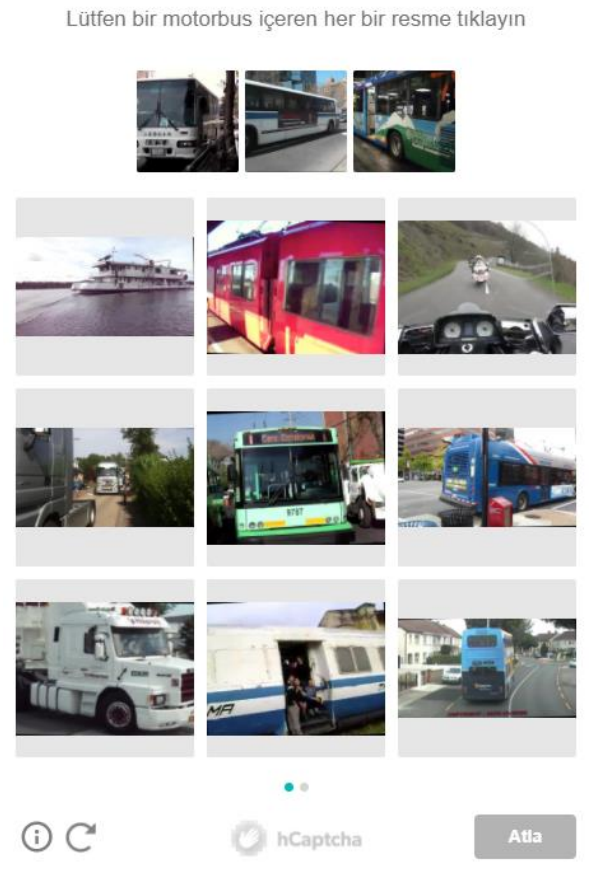

Şekil 5.2. Google ReCAPTCHA servisi

Şekil 5.2'de Google'ın geliştirmiş olduğu ReCAPTCHA servisinin sorgulama çeştlerinden biri görülmektedir. Bu resimde verilen fotoğrafllar arasında içinde bisiklet olanların seçilmesi istenmiştir.

Projemde daha önce bahsettiğim üzere HCAPTCHA servisini geçebilen bir sistem geliştireceğim. HCAPTCHA her sorguda her sayfada 9'ar adet farklı resim bulunmak üzere 1, 2 veya 3 sayfa resim ve tahmin etmemizi istediği bir sınıf veriyor.

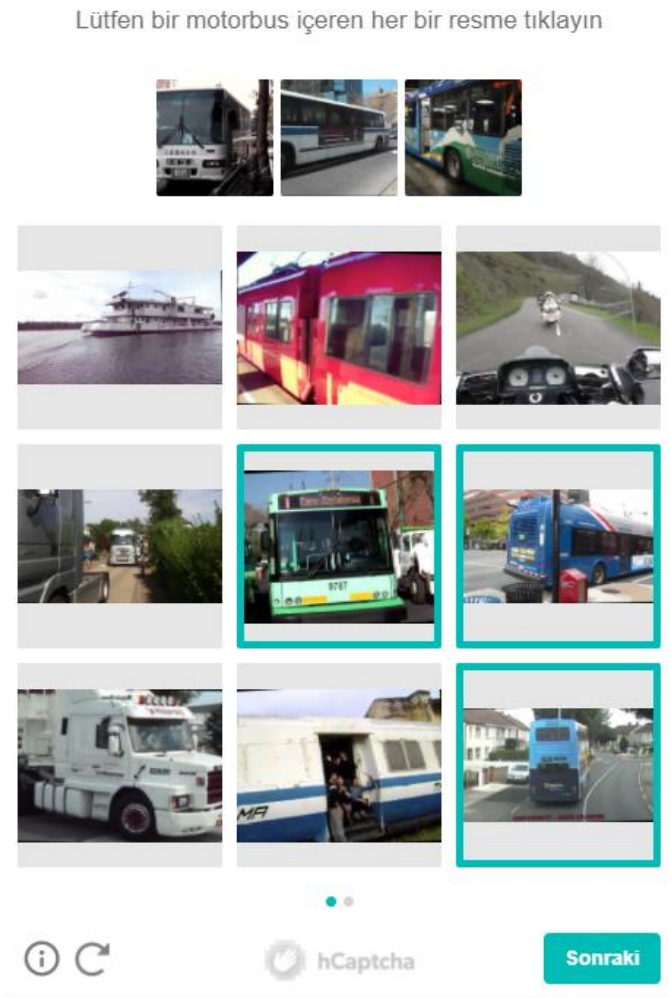
Bu sınıflar aşağıda listelenmiştir:

- Araba
- Bisiklet
- Kamyon
- Motosiklet
- Otobüs
- Tekne(yüzen deniz araçlarının hepsi)
- Bot(çok nadiren, ayrıca tekne sınıfı bunu da kapsıyor)
- Tren
- Uçak



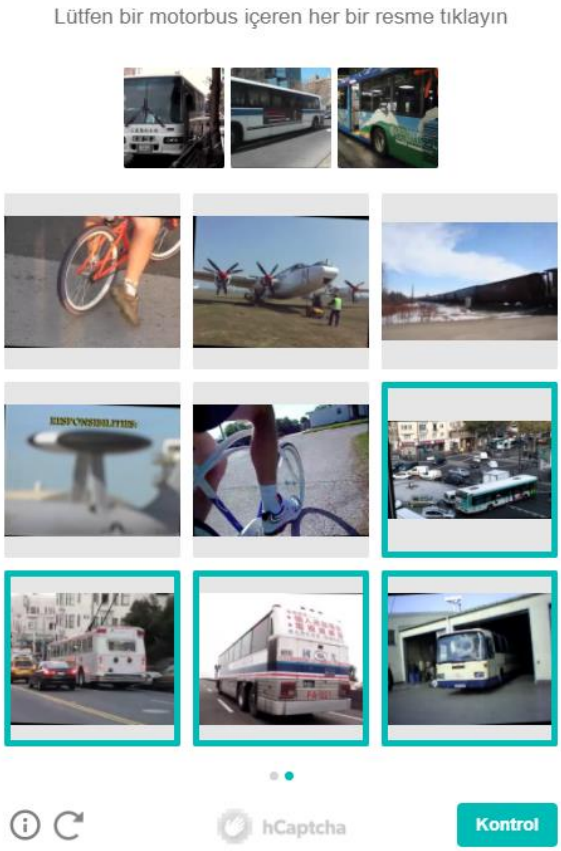

Şekil 5.3. HCAPTCHA servisi

Şekil 5.3'de içinde tekne bulunan resimlerin seçilmesi bizden istenmiştir. Sorgu 2 sayfadan yani toplam 18(2*9) sayfadan oluşmaktadır. Resimlerin bulunduğu kısmın altındaki mavi/gri noktaların sayısı sayfa sayısını vermektedir. Her bir sayfadan en az bir resim seçilerek diğer sayfaya geçilir. En az bir resim seçildikten sonra gri renkle gösterilmiş, pasif durumda olan "Atla" butonu renk değiştirecek, aktif duruma gelecek ve "Sonraki" kelimesini içinde barındıracaktır(Şekil 5.4).




Şekil 5.4. Sonraki sayfaya geçiş

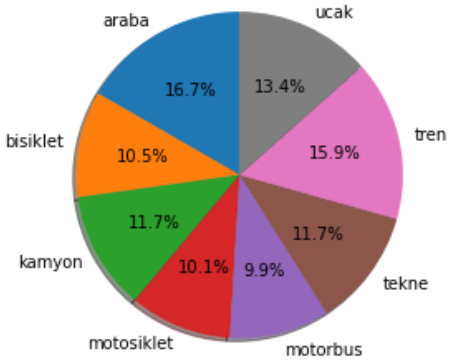
Son sayfaya gelindiğinde ve en az bir resim seçildiğinde(Şekil 5.4) ise "Kontrol" butonu aktif olur ve butona tıklandığında cevaplar sunucuya gönderilir. Sunucu cevaplardan en azından belirli bir kısmının doğru olmasını bekler. Eğer verilen cevapların doğruluğu istediği seviyede değilse yeni bir sorgu gönderir. Verilen cevaplar belirli bir doğruluk eşiğinin üstündeyse sorgu geçilmiş olur.




Şekil 5.5. Sorguyu gönder

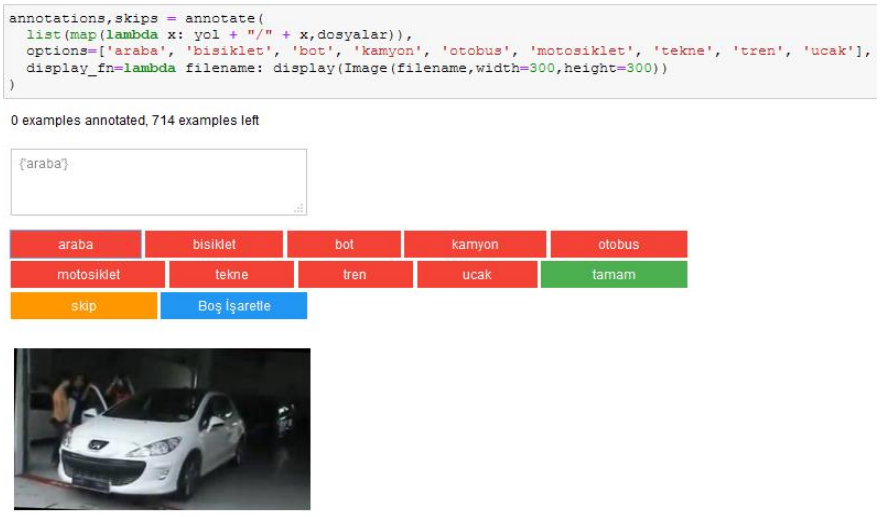
## 5.1. Verisetinin Çekilmesi Ve Düzenlenmesi

Modelimi eğititirken daha önce oluşturulmuş hazır bir veriseti kullanmadım. Captcha servisinin bize çözmemiz için sunduğu resimleri bir bot programı kullanarak ve ona özel script yazarak kendim çektim.




Şekil 5.6. Verisetindeki sınıfların dağılımı

Elimdeki veriseti etiketlenmemiş bir veriseti olduğu için onları tek tek elle etiketlemem gerekliydi. Bunu yapmak için “Pigeon” adlı jupyter network üzerinde resim etiketleme imkanı sunan basit bir python kütüphanesini kullandım. Modelimde kullanagim siniflarin sayisi 8(bot sınıfını da tekneye dahil ettim) tane. Bunlar; araba, bisiklet, bot, kamyon, otobüs, motosiklet, tekne, tren ve uçak. Bu sınıfların her birinden 300 küsür tanesini etiketledim. Projenin tüm adımlarını dolayısıyla bu veri etikleteme işlemini de tek başıma yaptığım için etiketlediğim veri sayısı sınırlı ve yetersizdi. Projemde her ne kadar transfer learning yapıyor olsam da veri sayısının azlığı problem oluşturmuştur. Imagenet gibi büyük bir veriseti üzerinde eğitilmiş, 12 milyon parametreli(aslında parametre sayısı diğer resim sınıflandırma modellerine çok göre daha az) bu modele çok küçük verisetleri ile transfer learning uygulamak riskli bir iştir. Modelin aşırı uydurma yapması olasıdır. Özellikle fine tunning işlemi sırasında asıl modelin sadece 2 konvolüsyonel katmanını çözmek durumunda kaldım. Aksi takdirde model overfitting yapıyordu. Bunun yanında modelin son halinde GlobalMaxPooling ile parametre sayısını ciddi seviyede düşürmüş olmam, dropout ile modelin rastgelelik katarak tek bir yola odaklanmasını engellemiş olmam ve veri zenginleştirme ile modeli daha fazla veri ile eğitmiş olmam overfittingin önüne geçmiş ve başarımı arttırmıştır. Fakat modelin başarımının daha da artması için daha fazla veri etiketlenmesi şarttı. Başlangıçta 2000 küsür resimle yaptığım eğitimi 3000 küsür resim ise yaptığımda başarımın %84.5 den %86’ lara kadar yükseldiği görülmüştür.




Şekil 5.7. Resim Etiketleme Arayüzü

Verisetinin yüzde 25'lik kısımını validation için ayırdım. Eğitim veriseti için veri zenginleştirme işlemi yaptım. Yaptığım veri zenginleştirme işlemleri : resmi döndürme, dikey eksende kaydırma, yatay eksende kaydırma, dikey eksende ters çevirme, yakınlaştırma, makaslama işlemleridir.

## 5.2. Modelin Oluşturulması, Düzenlenmesi, Eğitilmesi ve Sonuçlar

Resimlerimi modele vermeden önce boyutlarını eşitlememem gerekiyordu. Resimleri hangi boyutlara çevireceğimi seçerken son katmandaki öznitelik haritasının uzunluk ve genişlik değerlerini ne çok küçük ne de çok büyük seçmeliydim. Genel olarak son katmanda resmin boyutunun 7x7'ye olaracak şekilde modelin girdideki resimlerin boyutlandırıldığını gördüğüm için tüm resimleri 196x196 boyutlarına getirdim ve girdi katmanımın boyutunu da 196x196 olarak ayarladım. Transfer learning yaparken modelin sadece baş kısmını alacak şekilde yani tam bağlı katmanlarını almadan kullandım. Transfer learning yaptığım modelin son katmanındaki 3 boyutlu tensör 1536 kanallıydı. Bu katmandan sonra direkt olarak bir flatten katman ve hemen ardından bir tam bağlı katman eklemiş olsam : 7*7*1536*(tam bağlı katmandaki eleman sayısı) kadar eğitilecek parametrem daha olacaktı. Bu durum modelin eğitilmesini/çalışması yavaşlatacağı gibi overfitting problemine de sebep olabilirdi. Bunu engellemek için transfer learning yapılan modelin sonuna bir Global Max Pooling katmanı ekledim. Global Max Pooling, Max Pooling katmanından farklı olarak her kanaldaki en büyük olan değeri almakta. Modelin çıktısı 7x7x1536 boyutunda olduğu için GlobalMaxPooling 1536 adet 7x7 matris içindeki en büyük değeri seçecek ve sonuçta bize 1536 adet elemanlı bir vektör verecek. Bu sayede hem parametre sayımız çok ciddi oranda azalmış olacak hem de tam bağlı katmana bağlamaya hazır(tek boyutlu olduğu için) bir vektör elde etmiş olacağız.

```
top_conv (Conv2D)               (None, 7, 7, 1536)   589824
top_bn (BatchNormalization)     (None, 7, 7, 1536)   6144
top_activation (Activation)     (None, 7, 7, 1536)   0
=================================================================
Total params: 10,783,528
Trainable params: 10,696,232
Non-trainable params: 87,296
```

Şekil 5.8. Modelin sadece head kısmı alındığında parametre sayıları

| | | |
|---|---|---|
| top_conv (Conv2D) | (None, 7, 7, 1536) | 589824 |
| top_bn (BatchNormalization) | (None, 7, 7, 1536) | 6144 |
| top_activation (Activation) | (None, 7, 7, 1536) | 0 |
| global_max_pooling2d (GlobalMax | (None, 1536) | 0 |
| dropout (Dropout) | (None, 1536) | 0 |
| dense (Dense) | (None, 256) | 393472 |
| dense_1 (Dense) | (None, 8) | 2056 |

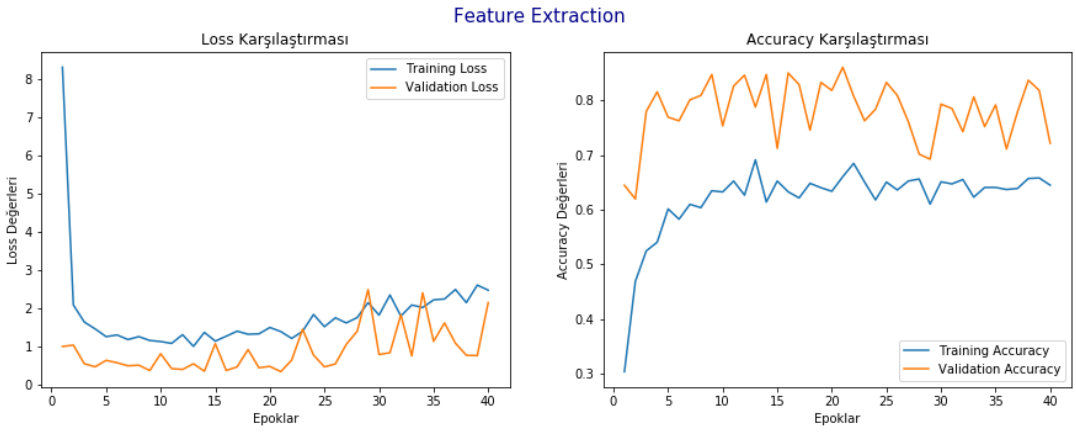
Total params: 11,179,056
Trainable params: 395,528
Non-trainable params: 10,783,528

Şekil 5.9. Model düzenlendikten sonraki son hal

Modelde kullanılan hiperparametreler, metrikler, algoritmalar:

- Son katmandaki aktivasyon fonksiyonu : softmax (çok sınıflı sınıflandırma yapıldığı için)
- Loss fonksiyonu : Categorical cross entropy
- Dropout yüzdesi : 0.30 (dropout eklenen 2 katman arasında çok ciddi sayılarda bağlantılar olmadığı için 0.30 yeterli görüldü)
- Optimizasyon algoritması : RMS Prop
- Batch size : 128

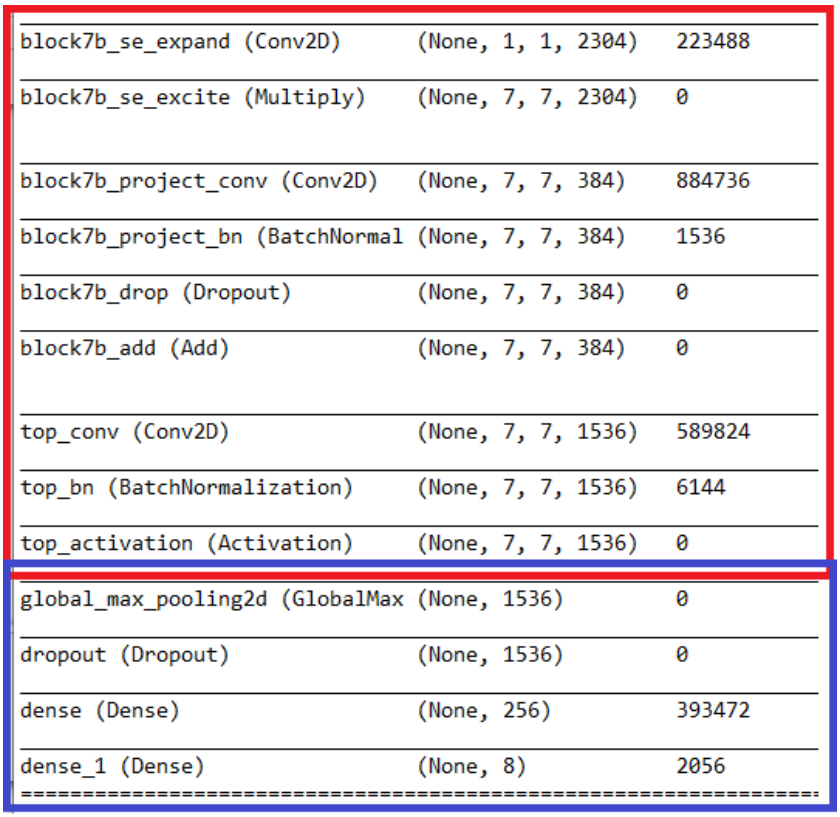
Model öncelikle transfer learning yapılan modelin eklenen tüm katmanları dondurularak sadece kendi eklediğim katmanlar eğitilir halde iken eğitildi. Bu aşama feature extraction aşaması olarak adlandırılmaktadır. Bu aşamada model 0.001 learning rate ile 40 epoch eğitilmiştir. Eğitim grafiği aşağıda gözükmektedir.




Şekil 5.10 Feature extraction aşamasındaki eğitim ve validation grafiği

Şekil 5.10'daki grafik incelenirse modelde bir genel ve ciddi anlamda bir overfitting problemi yoktur. Fakat 25. Epochtan sonra validation loss'taki artış overfittingin habercisidir. Muhtemelen model daha fazla eğitilmiş olsa overfitting problemi ile karşı karşıya kalacaktım.

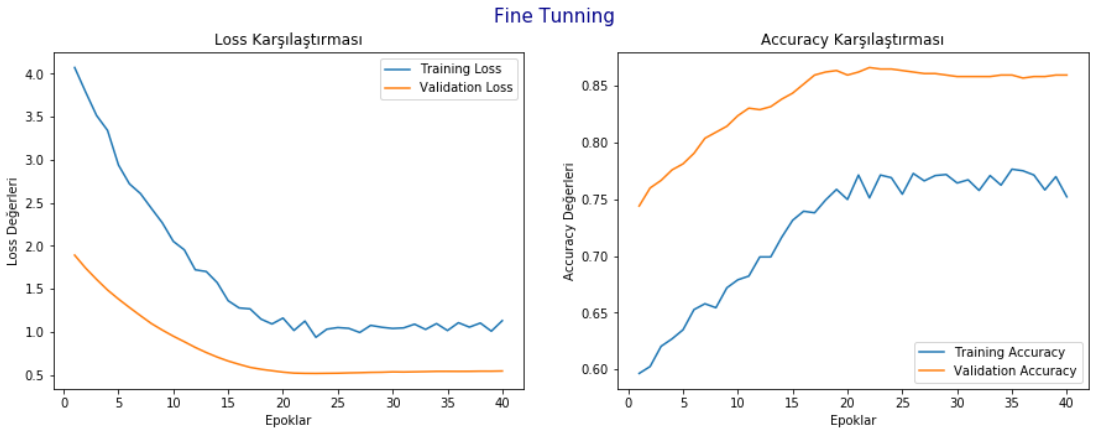
Feature extraction aşaması tamamlandıktan sonra transfer learning yapılan modelin son katmanlarından bir kısmı çözülerek fine tunnig işlemi yapılmıştır.




Şekil 5.11. Modelde çözülen kısımlar

Şekil 5.11'de mavi kutu içine alınanlar daha önce modele benim eklediğim ve feature extraction aşamasında eğitilen katmanlardır. Kırmızı kutu içerine alınanlar ise fine tunnig yapmak için eğitilebilir hale getirilmiş(çözülmüş) katmanlardır. Elimizdeki veriseti küçük olduğu için overfitting problemi yaşamamak adına çözülecek katmanların iyi seçilmesi gerekmektedir. Çözülecek katman sayısı /çözülecek eğitilen parametre sayısı çok olursa overfitting problemi yaşanacaktır.

Fine tunning işlemi ile daha önce başka verisetleri üzerinde öğrenilmiş bilgilere, modeli bir başka verisetini daha iyi genelleyecek/başarım verecek hale getirmek için ince ayarlamalar yapılır. Bu ince ayarlamalar yapılırken adımları küçük atmalı hali hazırda bulunan faydalı/önemli bilgilerin kaybolmasına engel olmalıyız. Bu yüzden fine tunnig işlemi sırasında learning rate oranı oldukça düşük seçilmelidir. Ben fine tunnig aşamasında learning rate değerini 0.000003 olarak seçtim ve modelimi 40 epoch kadar daha eğittim. Fine tunning sonucu oluşan eğitim grafiği aşağıdadır.




Şekil 5.12. Fine tunning aşamasındaki eğitim ve validation grafiği

Fine tunning sonucu yüzde 81 civarı olan validation başarımım yüzde 85'lere kadar çıkmıştır. Bu aşamada da overfitting problemi gözükmemektedir. İstenilen sonuçlar elde edilmiştir.

Buraya kadar yapılan tüm işlemler Python programlama dili vasıtasıyla yapılmıştır. Derin öğrenme işlemlerinin yapıldığı kısımlar Keras; verisetiyle ilgili birkaç küçük işlemin yapıldığı kısımlar Sklearn; resmi boyutlandırma, okuma vb işlemlerin yapılması için OpenCV; veri tutma, düzenleme, matematiksel işlemler için Numpy; grafiklerin çizilme işlemi içinse Matplotlib kütüphanesi kullanılmıştır.

### 5.3. Modelin Çalıştırabilir Bir Programa Dönüştürülmesi

Buraya kadar veriseti çekildi, etiketlendi, model oluşturuldu ve eğitildi. Buradan sonra bu modeli kullanacak bir programın yazıldığı kısım anlatılacaktır. Oluşturduğum model insanların yapması istenilen resim sınıflandırma işini yapmakta. Aynı şekilde

bu modele girdi verecek, modelin çıktılarına göre Captcha çözme işlemini otamitize edecek bir program yazmam gerekliydi. Bunu Selenium-Web Driver ile yaptım. Selenium test otomasyon aracı. Selenium-webdriver ise tarayıcıların otamatize edilmesini sağlayan bir Selenium modülü. Selenium web-driver ile bir tarayıcıya kendinizin yaptığını hemen hemen her şeyi yaptırabilirsiniz. Benim webdriver ile yapmam gereken ilk şey captchanın bulunduğu web sayfasına gitmek, buradaki captcha aracının checkbox'ına tıklayarak captcha sorgusunu almak ve bu sorgudaki resimlerin adreslerinin bulunarak diske kaydedilmesi işlemidir.

Öncelikle bir web sayfasındaki captcha servisine ait eklentinin içindeki "checkbox"'a tıklayarak captcha servisinden bir sorgu alabilmek için selenium gideceği web sayfasındaki hangi HTML elemanına tıklayacağını bildirmemiz gerekmektedir. Bu bildirme işlemi farklı şekillerde yapılabilir:

- HTML elemanın id'si ile : Bir web sayfasındaki id değerine sahip her bir elemanın id'si benzersizdir. Dolayısıyla selenium-webdrivera şu id değerine sahip elemanı bul diyerek bir istediğimiz elemanı seçmesini sağlayabiliriz.
- Class ismi ile : HTML elemanının class değeri ile seçim yapmayı sağlar.
- Tag ile : HTML elemanının tag değeri ile seçim yapılmasını sağlar.
- Name ile : HTML elemanının name eğeri ile seçim yapılmasını sağlar.
- CSS Selector ve XPATH : HTML elemanının sayfadaki tüm elemanlar içersindeki hiyerarşisini dikkate alarak benzersiz bir yolunu belirtirler.

Ben projemde seçmek istediğim elemanların çoğunun id değeri olmadığı için, class,tag,name değerleri olsa bile bu değerler benzersiz olmadığı(HTML'in yapısı gereği) için çoğunlukla XPATH ve CSS Selector'u kullandım.

Captcha eklentisi bir web sayfasında iframe içinde bulunmakta. Selenium gibi bir otomatasyon aracıyla onlarla iletişime geçmek istiyorsak öncelikle bu iframe yapısına geçiş yapmalıyız. Bu iframe yapısına geçiş yaptıktan sonra HCaptcha servisine ait checkboxa tıklayıp sorgu talebinde bulunuyoruz. Fakat sorgumuzun geldiği frame da başka bir iframe olarak karşımıza çıkacak. Bu yüzden ana frame'mimize yani başlangıçta sayfayı ziyaret ettğimizde olduğum frame'a geri dönüyoruz. Buradan da

sorgunun geldiği iframe'a geçiş yapıyoruz. Bu noktadan sonra sorgudan gelen resimler, butonlar, bilgiler gibi sorgu sonucu gelen her şeye erişebiliriz.

```
def checkboxa_tikla(self,iframe1,iframe2):
    self.browser.switch_to.frame(self.browser.find_element_by_xpath(iframe1))
    time.sleep(1.0)

    checkbox = self.browser.find_element_by_id("checkbox")
    checkbox.click()
    time.sleep(0.7)

    self.browser.switch_to.parent_frame()
    time.sleep(0.7)

    self.browser.switch_to.frame(self.browser.find_element_by_xpath(iframe2))
    time.sleep(0.7)
```

Şekil 5.13. Sorgunun alınması için gerekli işlemler

Bu noktadan sonra elimizde sorgu olduğuna göre artık diğer işlemleri yapabiliriz. Daha önce HCaptcha'nın farklı sayfa sayılarında sorgular gönderdiğinden bahsetmiştik. Bu farklı sayfa sayısını çalışma anında dinamik olarak yakalamalıyız. Bu sayıyı aldıktan sonra sonra bir ana döngü oluşturup bu döngüyü sayfa sayısı kadar döndürüceğiz. Ayrıca sorguda bizden hangi sınıfa ait nesnenin seçmemiz istendiğini için bu sınıfı da sorgudan yakalamalıyız. Sonrasında ise üzerinde tahmin yapıp, tahmin sonucuna göre seçeceğimiz resimleri kapsayan div yapılarını ve resimleri bulmamız ve bir yerde tutmamız gerekiyor. Her bir sayfadaki 9 resmin linki bulunduktan sonra resimler indirilerek diske kaydediliyor. Daha sonra resimler diskten okunuyor ve yeniden boyutlandırılıyor ve normalize ediliyor. Bu noktadan sonra resimler artık modele verilmeye hazır. Resimler modele veriliyor ve tahmin değerleri yakalanıyor. Tahmin değerleri (9,8) boyutunda. 9 resimlerimizin sayısı, 8 de sınıf sayısı. Yani 9 resmin her biri için 8 tane tahmin değeri üretiyor. Son katmanda softmax aktivasyon fonksiyonu kullandığımız için bu 8 tahmin değerinin toplamı 1'e eşit olacak şekilde olacak. 9 resme ait tahmin değeri arasından her resim için sorguda istenilen tahmin değeri 0.4'den büyükse bu resimde istenilen sınıfa ait nesne var olarak kabul ediyorum ve sonrasında bu resimlere selenium-webdriver aracılığıyla tıklıyorum. 0.4 değeri benim seçtiğim değer daha az veya daha çok olabilir ama ben bu değerde sistemin daha iyi başarım verdiğiniz gözlemledim. Bazı durumlarda 9 resmin hiçbiri istenilen sınıfa ait

tahmin eşik değerini geçemeyebilir. HCaptcha bizden her sorgu sayfasında en az bir resme tıklamamızı istiyor. Ben de bu yüzden eğer hiçbir max değeri eşik değerini geçmiyorsa bunlardan en büyük olanına tıklanmasını istedim ve programımı o şekilde düzenledim.

```
def calis(self):
    sayfa_sayisi = self.sayfaSayisiniBul()
    if (sayfa_sayisi == 0):
        sayfa_sayisi = 1
    tumResimler = np.empty((sayfa_sayisi * 9, 196, 196, 3))
    bilgi_satir = self.browser.find_element_by_xpath(self.bilgi_satir).text
    for i in range(sayfa_sayisi):
        # kapsayici divleri al
        resimlerDivTemp = list()
        for j in range(9):
            resimlerDivTemp.append(
                self.browser.
                    find_element_by_xpath(self.resimleri_kapsayan_divler.format(j + 1)))

        # kapsayici divden linkleri al
        resimLinkleriTemp = list()
        for resimDiv in resimlerDivTemp:
            eleman = resimDiv.find_element_by_css_selector("div.image-wrapper > div")
            style = eleman.get_attribute("style")
            link = self.url_ayikla(style)
            # print(link)
            resimLinkleriTemp.append(link)

        if (self.indir(resimLinkleriTemp, self.yol, i * 9 + 1)):
            resimler = self.resimleri_oku(i * 9 + 1, i * 9 + 10, self.yol)
            tumResimler[i * 9:i * 9 + 9] = resimler
            sonuclar = self.__model.predict(resimler)
            siniri_gecen_varmi=False;
            for k, sonuc in enumerate(sonuclar):
                #tahmin_k = list(self.uniqAdlar.keys())[sonuc.argmax()]
                temp = self.__site_to_veriseti[self.istenilen_nesneyi_bul(bilgi_satir)]
                if (sonuc[self.__uniqAdlar[temp]] > 0.4):
                    resimlerDivTemp[k].click()
                    time.sleep(0.3)
                    siniri_gecen_varmi=True
                    print(k)
            if(not siniri_gecen_varmi):
```

Şekil 5.14. Sorgunun işlenmesi, tahminlerin alınması ve sonrasınd yapılacaklar-1

```
        if(not siniri_gecen_varmi):
            temp = self.__site_to_veriseti[self.istenilen_nesneyi_bul(bilgi_satir)]
            max = np.argmax(sonuclar[:,self.__uniqAdlar[temp]])
            resimlerDivTemp[max].click()
        sonrakiButton = self.browser.find_element_by_xpath(self.sonraki_atla)
        sonrakiButton.click()
        time.sleep(1.5)
self.klasoru_temizle()
```

Şekil 5.15. Sorgunun işlenmesi, tahminlerin alınması ve sonrasınd yapılacaklar-2

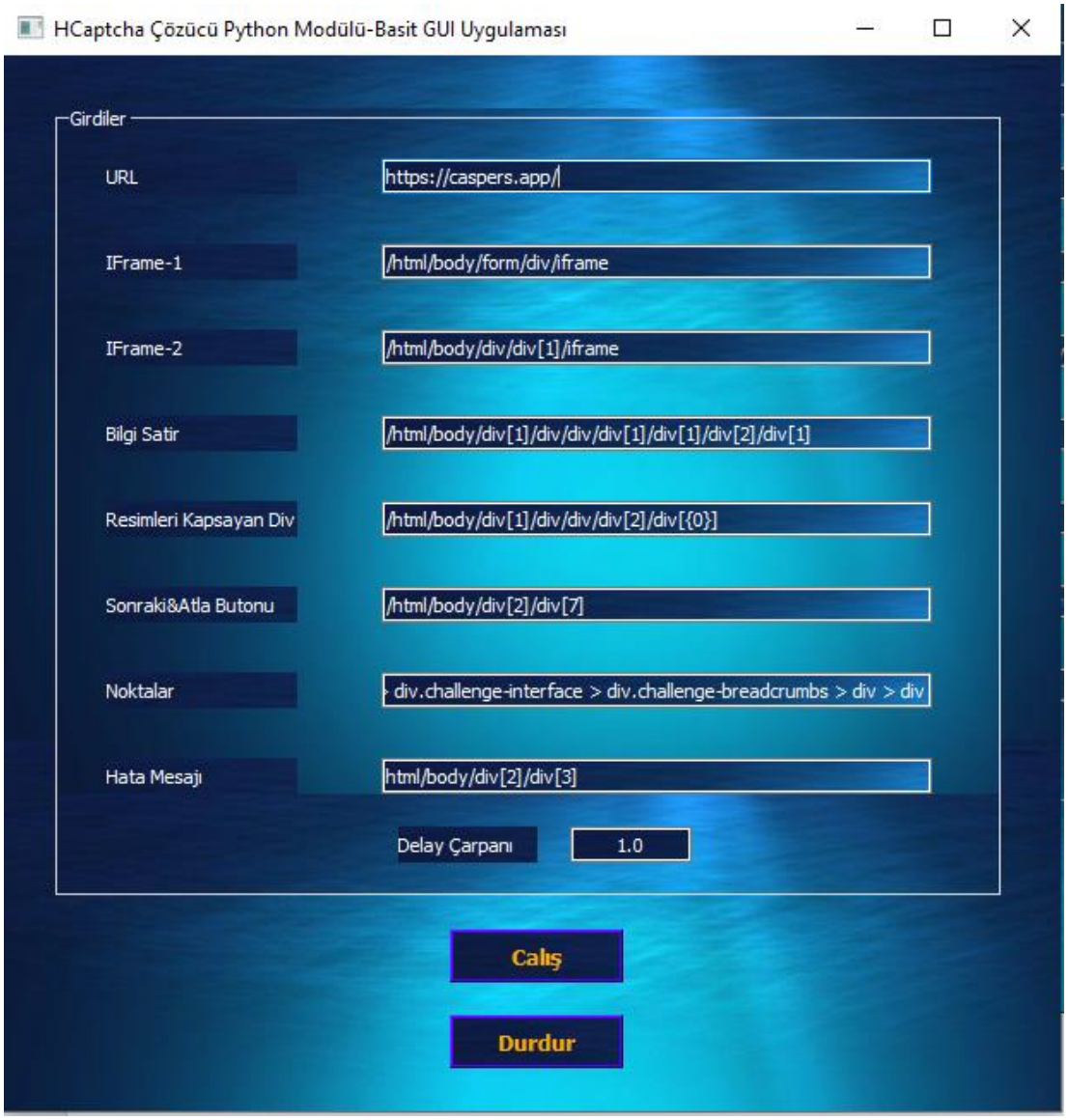
Her bir sayfa için 9 adet resim için de bu işlemleri tekrarlanıp resimlere tıklanıp sonraki aşamaya geçiliyor. Sonra sayfada resim seçimi yapıldıktan sonra sorguyu göndermek için bir buton olacak. Bu butona da tıklandıktan sonra sorgumuz gönderilmiş oluyor. Sonuçlar kabul edilirse sorgu ekranı kapanıyor ve yeşil bir tikle karşılaşıyorsunuz. Eğer kabul edilmezse sağ altta kırmızı bir uyarı mesajıyla karşılaşıyorsunuz.

Uygulamayı öncelikle bir Python kütüphanesi olarak tasarladım. İstenildiğinde gerekli ayarlamalar yapılarak farklı web sitelerinde kullanılabilir. Buraya kadar ki anlattıklarım kütüphane içinde yapılan işlemlerdi. Bir başka programda bu kütüphanenin import edilerek nasıl çok küçük kodlar ile çalıştıralabileceğini aşağıda görebilirsiniz.

```
from HCapCozucu import HCap
from selenium import webdriver
import time

YOL = "indirilenler"
URL = "https://caspers.app/"
IFRAME1 = "/html/body/form/div/iframe"
IFRAME2= "/html/body/div/div[1]/iframe"
BILGISATIR = "/html/body/div[1]/div/div/div[1]/div[1]/div[2]/div[1]"
RESIMLERIKAPSAYANDIVLER = "/html/body/div[1]/div/div/div[2]/div[{0}]"
SONRAKIATLA = "/html/body/div[2]/div[7]"
NOKTALAR = "body > div.challenge-interface > div.challenge-breadcrumbs > div > div"
HATAMESAJI = "html/body/div[2]/div[3]"

browser = webdriver.Firefox()
h = HCap(YOL,browser,BILGISATIR,RESIMLERIKAPSAYANDIVLER,SONRAKIATLA,NOKTALAR,HATAMESAJI)
browser.get(URL)
time.sleep(1.0)
h.checkboxa_tikla(IFRAME1,IFRAME2)

h.calis()
```

Şekil 5.16. Oluşturulan kütüphanenin kullanımı

Ayrıca programın daha kolay bir şekilde denenmesi amacıyla bir grafiksel arayüzü olan küçük bir uygulama tasarlandı. Bu uygulama yapılırken PyQt5 kullanıldı.




Şekil 5.17. Oluşturulan grafik arayüzü olan basit uygulama

# BÖLÜM 6. SONUÇLAR VE ÖNERİLER

Transfer learning, veri zenginleştirme, dropout, global max pooling gibi yöntemler kullanılarak yaklaşık 2000 kadar resimle eğitilmiş bir modelde yüzde 85 başarıma ulaşılmıştır. Bu kadar düşük sayıda veri ile 8 farklı sınıfın bu başarım ile sınıflandırılabiliyor olması modelin ve kullanılan yöntemlerin başarımını ortaya sermektedir. Daha önce bahsedildiği gibi daha fazla verinin etiketlenerek modelin daha fazla veriyle eğitilmesinin başarımı ciddi seviyede arttırması beklenmektedir. Çünkü HCaptcha her sorguda aynı sınıfa ait olanlarda bile çok farklı tipte resimler sunmaktadır. Resimler etiketlenirken, problem resim sınıflandırma problemi olarak seçildiği için her resim tek bir sınıfla etiketlenmiştir. Verisetinde bulunan sınıflardan birden fazlasının aynı resimde olduğu resimler etiketlenmekten kaçınılmıştır. Modelimin çıktılarından belli bir eşik değeri aşan resimleri seçtiğim için bu farklı sınıfların ikisini de modelin yakalamasını bekledim. Bazı durumlarda bu çalışıyor. Fakat modelin tüm çıktılarının toplamı 1 olduğu için(softmax fonksiyonundan dolayı) her iki sınıfın bulunduğu bir resimde eşik değeri olarak kabul ettiğim 0.4'ün üzerinde doğru olan sınıf için çıktı vermesi gerçekleşmeyebilir. Bu yüzden problem multi-class, multi-label classification veya object detection seçilerek de bu problem çözülebilirdi. Dahası bu 3 yöntemin en azından birinin kullanıldığı bir hibrit sistem düşünülebilirdi. Bu noktada seçilen modellerin çalışma hızının yüksek olması önemlidir. Örneğin object detection modeli olarak tiny-YOLO seçilebilir.

# KAYNAKLAR


[1] Yapay Zeka, Nasif Nabiyev

[2] http://dorukozsahin.com/yapay-zeka-tarihi

[3] https://www.udemy.com/course/makine-ogrenmesi

[4] https://www.udemy.com/course/python-egitimi

[5] https://www.youtube.com/watch?v=fODpu1-lNTw

[6] Python ile Derin Öğrenme, François Challet

[7] Derin Öğrenme, Ian Godfellow

# ÖZGEÇMİŞ

Ahmet Yasir Akbal, 30.12.1997 de İstanbul'da doğdu. İlk, orta ve lise eğitimini Kartal'da tamamladı. 2015 yılında Medine Tayfur Sökmen Lisesi'nden mezun oldu. 2016 yılında Sakarya Üniversitesi Bilgisayar Mühendisliği Bölümü'nü kazandı. Şu an Sakarya Üniversitesi'nde öğrencilik hayatına devam etmektedir.

# BSM 498 BİTİRME ÇALIŞMASI
# DEĞERLENDİRME VE SÖZLÜ SINAV TUTANAĞI

KONU :
ÖĞRENCİLER (Öğrenci No/AD/SOYAD):

| Değerlendirme Konusu | İstenenler | Not Aralığı | Not |
|---|---|---|---|
| **Yazılı Çalışma** | | | |
| **Çalışma klavuza uygun olarak hazırlanmış mı?** | x | 0-5 | |
| **Teknik Yönden** | | | |
| **Problemin tanımı yapılmış mı?** | x | 0-5 | |
| Geliştirilecek yazılımın/donanımın mimarisini içeren blok şeması (yazılımlar için veri akış şeması (dfd) da olabilir) çizilerek açıklanmış mı? | | | |
| Blok şemadaki birimler arasındaki bilgi akışına ait model/gösterim var mı? | | | |
| Yazılımın gereksinim listesi oluşturulmuş mu? | | | |
| Kullanılan/kullanılması düşünülen araçlar/teknolojiler anlatılmış mı? | | | |
| Donanımların programlanması/konfigürasyonu için yazılım gereksinimleri belirtilmiş mi? | | | |
| UML ile modelleme yapılmış mı? | | | |
| Veritabanları kullanılmış ise kavramsal model çıkarılmış mı? (Varlık ilişki modeli, noSQL kavramsal modelleri v.b.) | | | |
| Projeye yönelik iş-zaman çizelgesi çıkarılarak maliyet analizi yapılmış mı? | | | |
| Donanım bileşenlerinin maliyet analizi (prototip-adetli seri üretim vb.) çıkarılmış mı? | | | |
| Donanım için gerekli enerji analizi (minimum-uyku-aktif-maksimum) yapılmış mı? | | | |
| Grup çalışmalarında grup üyelerinin görev tanımları verilmiş mi (iş-zaman çizelgesinde belirtilebilir)? | | | |
| Sürüm denetim sistemi (Version Control System; Git, Subversion v.s.) kullanılmış mı? | | | |
| Sistemin genel testi için uygulanan metotlar ve iyileştirme süreçlerinin dökümü verilmiş mi? | | | |
| Yazılımın sızma testi yapılmış mı? | | | |
| Performans testi yapılmış mı? | | | |
| Tasarımın uygulamasında ortaya çıkan uyumsuzluklar ve aksaklıklar belirtilerek çözüm yöntemleri tartışılmış mı? | | | |
| **Yapılan işlerin zorluk derecesi?** | x | 0-25 | |
| **Sözlü Sınav** | | | |
| **Yapılan sunum başarılı mı?** | x | 0-5 | |
| **Soruları yanıtlama yetkinliği?** | x | 0-20 | |
| **Devam Durumu** | | | |
| **Öğrenci dönem içerisindeki raporlarını düzenli olarak hazırladı mı?** | x | 0-5 | |
| **Diğer Maddeler** | | | |
| | | | |
| | | | |
| | | | |
| **Toplam** | | | |

DANIŞMAN (JÜRİ ADINA):
DANIŞMAN İMZASI: