

YAPAY ZEKA VE MAKİNE ÖĞRENMESİ

KURS NOTLARI

Kaan ASLAN

C VE SİSTEM PROGRAMCILARI DERNEĞİ

Son Güncelleme Tarihi: 13/11/2021

Bu kurs notları Kaan ASLAN tarafından hazırlanmıştır. Kaynak belirtilmek koşuluyla her türlü alıntı yapılabılır.

GİRİŞ

Bu bölümde "yapay zeka (artificial intelligence) ve makine öğrenmesi (machine learning)" konusuna bir giriş yapılmaktadır.

Yapay Zeka (Artificial Intelligence) Nedir?

Zeka kapsamı, işlevleri ve yol açtığı sonuçları bakımından karmaşık bir olgudur. Zekanın ne olduğu konusunda psikologlar ve nörobilimciler arasında tam bir fikir birliği bulunmamaktadır. Çeşitli kuramcılar ve araştırmacılar tarafından zekanın çeşitli tanımları yapılmıştır. Bu kuramcılar ve araştırmacıların bazıları yaptıkları tanımdan hareketle zekayı ölçmek için çeşitli araçlar da geliştirmeye çalışmışlardır.

Charles Spearman zekayı "g" ve "s" biçiminde iki yeteneğin birleşimi olarak tanımlamıştır. Spearman'a göre "g" faktörü akıl yürütme ve problem çözmeyle ilgili olan "genel zekayı" belirtir. "s" faktörü ise müzik, sanat, iş yaşamı gibi özel alanlara yönelik "spesifik zekayı" belirtir. İnsanlar "zeka" denildiğinde daha çok genel zekayı kastetmektedirler.

Howard Gardner tarafından kuramsal hale getirilen "çoklu zeka (multiple intelligence)" zekayı "sözel/bilbilimsel (verbal/linguistic), müziksel (musical), mantıksal/matematiksel (logical/mathematical), görsel uzamsal (visual/spatial), kinestetik (kinesthetic), kişilerarası (interpersonalı, içsel (intrapsersonal)", olmak üzere başlangıçta yedi tür aayırılmıştır. Sonra bu yedi türde "doğasal (naturalistic), varoluşsal (existentialist)" biçiminde iki tür daha ekleyerek dokuzu çıkarmıştır.

Robert Sternberg'e göre ise "analitik (analytical), pratik (practical) ve yaratıcı (creative)" olmak üzere üç tür zeka vardır. Analitik zeka problemi parçalara ayırma, analiz etme ve çözme ile ilgili yetileri içermektedir. Bu yetiler aslında zeka testlerinin ölçmeye çalıştığı yetilerdir. Pratik zeka yaşamı sürdürmek için gerekli olan pratik becerilerle ilgilidir. Yaratıcı zeka ise yeni yöntemler bulmak, problemleri farklı biçimlerde çözebilmek, yenilikler yapabilmekle ilgili yetilerdir.

Catell-Horn-Carroll (CHC) Teorisi diye isimlendirilen çok katmanlı zeka teorisi üzerinde en çok durulan zeka teorisidir. (Raymond Catell aslında Spearman'ın John Horn ise Catell'in öğrencisidir.) Catell zekayı "kristalize zeka (crystalized intelligence)" ve "akıcı zeka (fluid intelligence)" biçiminde ikiye ayrmıştır. Kristalize zeka öğrenilmiş ve oturmuş bilgi ve becerilerle ilgili iken akıcı zeka problem çözme ve yeni durumlara uyum sağlama becerileriyle ilgilidir. John Horn ise Catell'in bu iki tür zekasını genişleterek ona görsel işitsel yetileri, belleğe erişimle ilgili yetileri, tepki zamanlarına ilişkin yetileri, niceliksel işlemlere yönelik yetileri, ve okuma yazma becerilerini de eklemiştir. Nihayet John Carroll zeka ile

İlgili 460 yeteneği faktör analizine sokarak üç katmanlı bir zeka teorisi oluşturmuştur. CHC teorisi Stanford Binet ve Wechles testlerinin ileri sürümlerinin benimediği zeka anlayışıdır.

Yapay zeka ise ismi üzerinde insan zekası ile ilgili bilişsel süreçlerin makineler tarafından sağlanmasına yönelik süreçleri belirtmektedir. Yapay zeka terimi ilk kez 1955 yılında John McCarthy tarafından uydurulmuştur. Doğal zekada olduğu gibi yapay zekanın da farklı kişiler tarafından pek çok tanımı yapılmaktadır. Ancak bu terim genel olarak "insana özgü nitelikler olduğu varsayılan akıl yürütme, anlam çıkartma, genelleme ve geçmiş deneyimlerden öğrenme gibi yüksek zihinsel süreçlerin makineler tarafından gerçekleştirilebilmesidir" biçiminde tanımlanabilir. Yapay zekanın diğer bazı tanımları şunlardır:

- Yapay zeka insan zekasına ilişkin "öğrenme", "akıl yürütme", "kendini düzeltme" gibi süreçlerin makineler tarafından simüle edilmesidir.
- Yapay zeka zeki makineler yaratma amacıyla olan bilgisayar bilimlerinin bir alt alanıdır.
- Bilgisayarların insanlar gibi davranışmasını sağlamayı hedefleyen bilgisayar bilimlerinin bir alt dalıdır.

Yapay zekanın simüle etmeye çalıştığı bilişsel süreçlerin şunlar olduğuna dikkat ediniz:

- Bir şeyin nasıl yapılacağını bilme (knowledge)
- Akıl yürütme (reasoning)
- Problem çözme (problem solving)
- Algılama (perception)
- Öğrenme (learning)
- Planlama (planning)
- Doğal dili açıklama ve konuşma
- Uzmanlık gerektiren alanlarda karar verme

Yapay Zeka Çalışmalarının Kısa Tarihi

Yapay zeka ile ilgili düşünceler ve görüşler antik çağ'a kadar götürülebilir. Ancak modern yapay zeka çalışmalarının 1950'li yıllarda başladığı söylenebilir. Şüphesiz yapay zeka alanındaki gelişmeler de aslında başka alanlardaki gelişmeler tetiklemiştir. Örneğin bugün kullandığımız elektronik bilgisayarlar olmasaydı yapay zeka bugünkü durumuna gelemeyecekti. İşte aslında pek çok bilimsel ve teknolojik gelişmeler belli bir noktaya gelmiş ve yapay zeka dediğimiz bu alan 1950'lerde ortaya çıkmaya başlamıştır.

Yapay zeka çalışmalarının ortayamasına yol açan gelişmeler şunlar olmuştu:

- **Mantıktaki Gelişmeler:** Bertrand Russell ve North Whitehead tarafından 1913 yılında yazılmış olan "Principia Mathematica" adlı kitap biçimsel mantık (formal logic) devrim niteliğinde etki yapmıştır.
- **Matematikteki Gelişmeler:** 1930'larda Alonzo Church "Lambda Calculus" geliştirmiş ve özyinelemeli fonksiyonel notasyonla hesaplanabilirliği araştırılmış ve sorgulamıştır. Yine 1930'larda Kurt Gödel "biçimsel sistemler (formal system)" üzerindeki çalışmalarıyla teorik bilgisayar bilimlerinin öncülüğünü yapmıştır.
- **Turing Makineleri:** Alan Turing'in henüz elektronik bilgisayarlar gerçekleştirilmeden 1930'lu yılların ortalarında (ilk kez 1936) tasarıladığı teorik bilgisayar yapısı olan "Turing Makineleri" bilgisayar bilimlerinin ve yapay zeka kavramının ortaya çıkışında etkili olmuştur. (Turing makinelerinin çeşitli modelleri vardır. Bugün hala Turing makineleri algoritmalar dünyasında algoritma analizinde ve algoritmik karmaşıklıkta teorik bir karşılaştırma amacıyla kullanılmaktadır.)

- Elektronik Bilgisayarların Ortaya Çıkması: 1940'lı yıllarda ilk elektronik bilgisayarlar gerçekleştirilmeye başlanmıştır. Bilgisayarlar yapay zeka alanının gelişmesinde için en önemli araçlar durumundadır.

Yapay Zeka (Artificial Intelligence) terimi ilk kez John McCarthy tarafından 1955 yılında uydurulmuştur. John McCarthy, Marvin Minsky, Nathan Rochester ve Claude Shannon tarafından 1956 yılında Dartmouth College'de bir konferans organize etmişlerdir. Bu konferans yapay zeka kavramının ortaya çıkışının bakımından çok önemlidir. Bu konferans yapay zekanın doğumu olarak kabul edilmektedir. Aynı zamanda yapay zeka terimi de bu konferansta katılımcılar tarafından kabul görmüştür. John McCarthy aynı zamanda dünyanın ilk programlama dillerinden biri olan Lisp'i de 1958 yılında tasarlamıştır. Lisp hala yapay zeka çalışmalarında kullanılmaktadır.

1956-1974 yapay zekanın altın yılları olmuştur. Bu yıllar arasında çeşitli algoritmik yöntemler geliştirilmiş ve pek çok uygulama üzerinde çalışılmıştır. Örneğin arama (search) yöntemleri uygulanmış ve arama uzayı (search space) sezgisel (heuristic) yöntemlerle daraltılmaya çalışılmıştır. Yine bu yıllarda doğal dili anlamaya yönelik ilk çalışmalar gerçekleştirilmiştir. Bu ilk çalışmalarдан elde edilen çeşitli başarılar yapay zeka alanında iyimser bir hava estirmiştir. Örneğin:

- 1958 yılında Simon ve Newell "10 yıl içinde dünya satranç şampiyonunun bir bilgisayar olacağını" iddia etmişlerdir. (Halbuki bu durum 90'lı yılların ikinci yarısında gerçekleşmeye başlamıştır.)
- 1970 yılında Minsky 3 yıldan 8 yıla kadar makinelerin ortalama bir insan zekasına sahip olabileceği iddia etmiştir.

1974-1980 yılları arasında yapay zeka alanında bir durgunluk yaşanmıştır. Daha önce yapılan tahminlerin çok iyimser olduğu görülmüş bu da biraz hayal kırıklığına yol açmıştır. Bu yıllarda yapay sinir ağları çalışmaları büyük ölçüde durmuştur. Yeni projeler için finans elde edilmesi zorlaşmıştır.

1980'li yıllarla birlikte yapay zeka çalışmalarında yine yükseliş başlamıştır. 80'li yıllarda en çok yükselişe geçen yapay zeka alanı "uzman sistemler" olmuştur. Japonya bu tür projelere önemli finans ayırmaya başlamıştır. Ayrıca Hopfield ve Rumelhart'in çalışmaları da "yapay sinir ağlarına" yenik bir soluk getirmiştir.

1987-1993 yılları arasında yine yapay zeka çalışmalarında bir duraklama baş göstermiştir. Bu konudaki çeşitli projeler için finans akynakları da kendilerini geri çekmiştir.

1993 yılından itibaren yapay zeka alanı yine canlanmaya başlamıştır. Bilgisayarların güçlenmesi, Internet teknolojisinin gelişmesi, mobil aygıtların gittikçe yaygınlaşması sonucunda veri analizinin önemi artmış ve bu da yapay zeka çalışmalarına yeni bir boyut getirmiştir. 1990'lı yılların ortalarından itibaren veri işlemesinde yeni bir dönem başlamıştır. Veri madenciliği bir alan olarak kendini kabul ettirmiştir. Özellikle 2011 yılından başlayarak büyük veri (big data) analizleri iyice yaygınlaşmış, yapay sinir ağlarının bir çeşidi olan derin öğrenme (deep learning) çalışmaları hızlanmış, IoT uygulamaları da yapay zekanın önemini heften artırmıştır.

Yapay Zekanın Alt Alanları

Yapay zeka aslında pek çok alt konuya ayrılabilen bir alandır. Yapay zekanın önemli alt alanları şunlardır:

- Makine Öğrenmesi
- Yapay Sinir Ağları ve Derin Öğrenme
- Bulanık Sistemler (Fuzzy Logic)
- Evrimsel Yöntemler (Genetic Algorithms, Differential Evolution)
- Üst Sezgisel (Meta Heuristic) Yöntemler (Karınca Kolonisi, Particle Swarm Optimization)
- Olasılıksal (Probabilistic) Yöntemler (Bayesian Network, Hidden Markov Model, Kalman Filter vs.)
- Uzman Sistemler (Expert Systems)

Yapay Zekanın Uygulama Alanları

Yapay zekanın tipik uygulama alanları şunlardır:

- Yapay Yaratıcılık Faaliyetleri (Artificial Creativity)
- Planlama ve Çizelgeleme Faaliyetleri
- Akıl Yürütme (Reasoning) Faaliyetleri
- Otomatik Hedef Belirlemesi (Automatic Target Recognition)
- Yüz Tanıma Sistemleri (Facial Recognition Systems)
- Konuşma Tanıma (Speech Recognition)
- Konuşanı Tanıma (Speaker Recognition)
- Bilgisayarlı Görü (Computer Vision)
- Görüntü İşleme
- OCR (Optical Character Recognition) ve Zeki Sözcük Tanıma (Intelligent Word Recognition)
- Nesne Tanıma (Object Recognition)
- El Yazısı Tanıma
- Yüz Tanıma
- Hastalığa Tanı Koyma
- Uzman Sistemler
- Karar Destek Sistemleri
- Klinik Karar Destek Sistemleri
- Zeka Oyunlarını Oynaması (Satranç, Go, vs.)
- Veri Madenciliği (Data Mining)
- Yazı Madenciliği (Text Mining)
- Süreç Madenciliği (Process Mining)
- E-Posta Spam Filtreleri
- Etkinlik Belirleme (Activity Recognition)
- Otomatik Resim Yorumlama (Automatic Image Annotation)
- İlişkili Olanları Bulma (Relationship Extraction)
- Referans Çözümlemesi (Coreference Resolution)
- Doğal Dil Algılama
- Dil Algılama
- Makine Çevirisi
- Semantic Web
- Soru Yanıtlama Sistemleri
- Örnek Tanıma
- İnsan Gibi Davranan Robotlar
- Sesli Konuşma Sistemleri (Text To Speech, Speech to Text)
- Sanal Gerçeklik Sistemleri

Makine Öğrenmesi Nedir?

Aslında makine öğrenmesinin ne olduğundan önce öğrenmenin ne olduğunu ele almak gereklidir. Psikolojide öğrenme "davranışta göreli biçimde kalıcı değişiklikler oluşturan süreçler" olarak tanımlanmaktadır. Bu tanımdaki davranış (behavior) klasik davranışçılar göre "gözlemlenebilen devinimleri" kapsamaktadır. Ancak daha sonra "radikal davranışçılar" bu davranış kavramını zihinsel süreçleri de kapsayacak biçimde genişletmiştir.

Psikoloji de öğrenme kabaca dört bölümde ele alınmaktadır.

1) Klasik Koşullanma (Classical Conditioning): Burada dışsal bir uyaran başka bir uyaranla zamansal bakımdan eşleşmiştir. Böylece birinci uyaran oluştugunda organizma ikinci uyaranın olusacagini öğrenir. Pavlov'un köpek deneyi klasik koşullanma ile ilgilidir. Klasik koşullanma süreci pek çok hayvan üzerinde denenmiştir. Hayvanların çok büyük çoğunluğu klasik koşullanma ile öğrenebilmektedir. Birinci kuşak davranışçılar pek çok davranışın nedenini klasik koşullanmaya açıklamışlardır. Gerekten de fobilerin çoğunda klasik koşullanmanın etkili olduğu görülmektedir. Özellikle olumsuz birtakım sonuçlar doğuran uyaranlar çok kısa süre içerisinde klasik koşullanmaya yol açabilmektedir. Örneğin karanlık bir sokakta saldırıcı ugrayan kişi yeniden karanlık bir sokağa girdiğinde saldırıcı ugrayacağı hissine kapılabilmektedir.

2) Edimsel Koşullanma (Operant Conditioning): Edimsel koşullanma en önemli insan öğrenmesi yollarından biridir. Pek çok süreç edimsel koşullanma ile öğrenilmektedir. Klasik koşullanmada önce uyaran sonra tepki gelmektedir. Halbuki edimsel koşullanmada önce tepki sonra uyaran gelir. Organizma bir faaliyette bulunur. Bunun sonucunda hoşa giden bir durum (yani ödül) oluşursa bu davranış tekrarlanır ve böylece öğrenme gerçekleşir. Yani özetle organizmada hoşa giden sonuçlar doğuran davranışlar tekrarlanma eğilimindedir. Bu öğrenme modelinde davranış sonucunda oluşan hoşa giden durumlara "pekiştireç (reinforcer)" denilmektedir. Davranış ne kadar pekiştirilirse o kadar iyi öğrenilmektedir. Pekiştireçler "pozitif" ve "negatif" olmak üzere ikiye ayrırlar. Pozitif pekiştireçler doğrudan organizmanın hoşuna gidecek uyarlardır. Negatif pekiştireçler ise organizmanın içinde bulunduğu hoş olmayan durumu ortadan kaldırın uyaranlardır. Edimsel koşullanma için bazı örnekler şöyle verilebilir:

- Ödevini yapan öğrenciye öğretmenin ödül vermesi ödev yapma davranışını artırmaktadır.
- Maddenin bunaltısı (anxiety) ortadan kaldırması kişiyi madde kullanımına teşvik etmektedir.
- Arabada emniyet kemeri bağlı değilken ses çıkmaktadır. Bu sesi ortadan kaldırmak için emniyet kemерinin bağlanması bir edimsel koşullanma sürecidir.

3) Sosyal Bilişsel Öğrenme (Social Cognitive Learning): Bu yönteme taklit yoluyla öğrenme de denilmektedir. Biz başkalarını taklit ederek de davranışlarımızı değiştirebilmekteyiz. Aslında sosyal bilişsel öğrenmede de bir bakıma pekiştirmeler söz konusudur. Ancak bu pekiştirme doğrudan değil dolaylı (vicarious) olmaktadır. Sosyal bilişsel öğrenmede birtakım bilişsel süreçlerin de devreye girdiğine dikkat ediniz. Çünkü bu süreçte kişinin başkalarının yaptığı davranışları izleme, izlediklerini bellekte saklama ve onlardan sonuçlar çıkartma süreçleri de söz konusu olmaktadır.

4) Bilişsel Öğrenme (Cognitive Learning): Biliş (cognition) organizmanın bilgi işlem faaliyetlerini anlatan bir terimdir. Biliş denildiğinde düşünme, bellek, dikkat, bilinç, akıl yürütme gibi faaliyetler anlaşılmaktadır. Araştırmacılar hiç pekiştireç olmadan öğrenmenin hayvanlarda da insanlarda da mümkün olduğunu göstermişlerdir. Yani biz klasik koşullanma, edimsel koşullanma ve sosyal öğrenme süreçleri olmadan da yalnızca bilişsel faaliyetlerle de öğrenebiliriz.

Halk arasında öğrenme denildiğinde genellikle sürecin davranışsal boyutu göz ardı edilmekte yalnızca bilişsel tarafı değerlendirilmektedir. Halbuki her türlü öğrenmede açık ya da örtük görelî bir biçimde kalıcı bir davranışın ortaya çıkması beklenir. Ancak "davranış (behavior)" sözcüğünün tanımı konusunda da tam bir anlaşma bulunmamaktadır. İlk davranışçılar yalnızca gözlemlenebilen devinimleri davranış olarak tanımlarken radikal davranışçılar zihinsel süreçleri de davranış tanımının içine katmaktadır.

O halde makine öğrenmesi (machine learning) nedir? Aslında psikolojideki öğrenme kavramı makine öğrenmesinde de geçerlidir. Biz makinenin (makine demekle donanımı ve yazılımı kastediyoruz) bir biçimde davranışını arzu edilen yönde değiştirmesini isteriz. Yani makinenin davranışını bizim istediğimiz yönde ve istediğimiz hedefleri gerçekleştirmeye anlamında değişimlidir. İşte makine öğrenmesi kabaca geçmiş bilgilerden ve deneyimlerden faydalı birtakım sonuçlar (davranışlar) ortaya çıkartan algoritmalar ve yöntemler topluluğudur. Makine öğrenmesinde üç bileşen vardır: Deneyim, Görev ve Performans. Deneyim canlılarda olduğu gibi makine öğrenmesinde de en önemli öğelerdendir. Makine öğrenmesinde deneyim birtakım verilerin analiz edilmesini ve onlardan bir kestirim ya da faydalı sonuçlar çıkartılması sürecidir. Görev makinenin yapmasını istediğimiz şeydir. Görevler düşük bir deneyimle düşük bir performansla gerçekleştirilebilirler. Deneyim arttıkça görevin yerine getirilme performansı da artabilir. İşte makine öğrenmesi temelde bunu hedeflemektedir. O halde makine öğrenmesinde bir veri grubu incelenir, analiz edilir, bundan

sonuçlar çıkartılır, sonra hedeflenen görev yerine getirilmeye çalışılır. Bu görevin yerine getirilmesi de gitgide iyileştirilir. Bu süreç çeşitli algoritmalarla ve yöntemlerle değişik biçimlerde ve yaklaşımalarla yürütülmektedir.

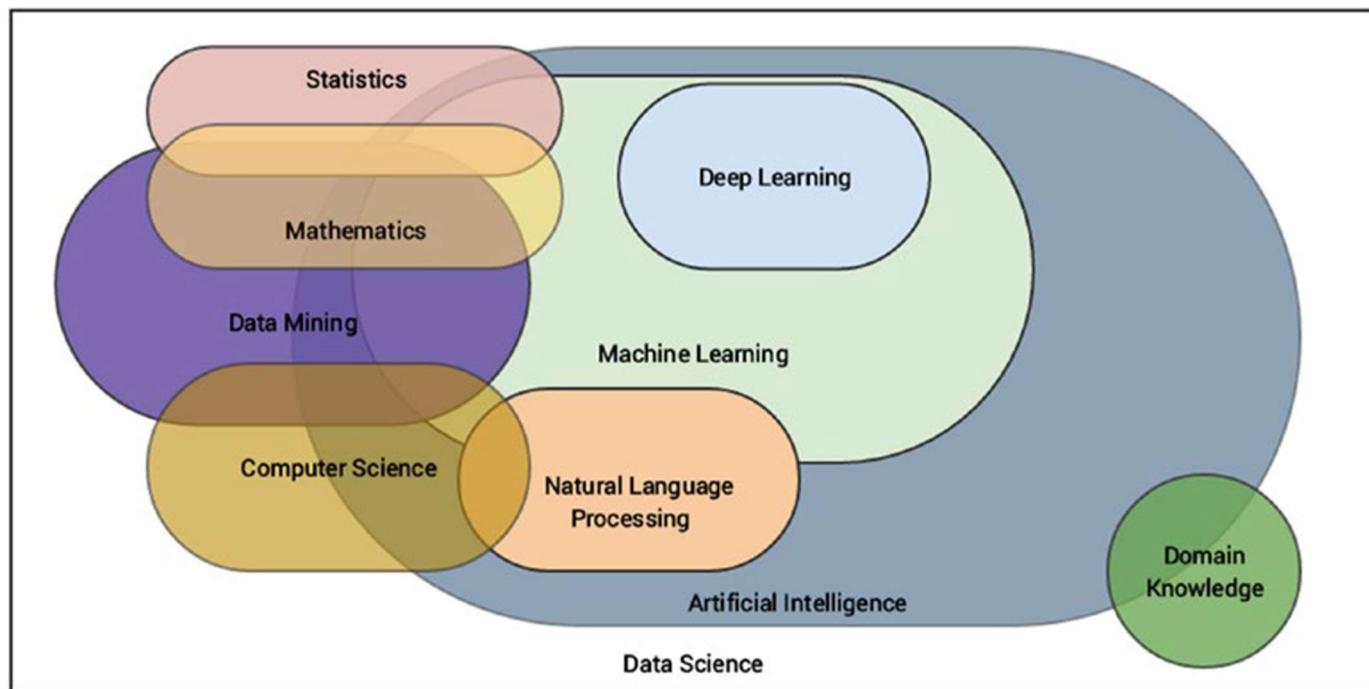
Makine öğrenmesi genel olarak üç bölümde ele alıp incelemektedir:

- 1) Denetimli Öğrenme (Supervised Learning)
- 2) Denetimsiz Öğrenme (Unsupervised Learning)
- 3) Pekiştirmeli Öğrenme (Reinforcement Learning)

Denetimli (supervised) öğrenmede makineye (yani algoritmaya) biz daha önce gerçekleşmiş olan olayları ve sonuçları girdi olarak veririz. Makine bu olaylarla sonuçlar arasında bağlantı kurar. Daha sonra biz yeni bir olayı makineye verdiğimizde onun sonucunu makineden kestirmesini isteriz. Denetimsiz öğrenmede biz makineye yalnızca olayları veririz. Makine bunların arasındaki benzerliklerden ve farklılıklardan hareketle bizim istediğimiz sonuçları çıkartmaya çalışır. Örneğin biz makineye resimler verip bunların elma mı armut mu olduğunu söyleyelim. Ve bunu çok miktarda yapalım. Sonra ona bir elma resmi verdığımızda o daha önceki deneyimlerden hareketle bunun elma olduğu sonucunu çıkartabilecektir. İşte bu denetimli öğrenmeye bir örnektir. Şimdi biz makineye elma ve armut resimlerini verelim ama bunların ne olduğunu ona söylemeyeelim. Ondan bu resimleri ortak özelliklerine göre iki gruba ayırmasını isteyelim. Bu da denetimsiz öğrenmeye örnektir. Pekiştirmeli öğrenmede ise tıpkı edimsel koşullamada olduğu gibi hedefe yaklaşan durumlar ödüllendirilerek makinenin öğrenmesi sağlanır.

Yapay Zekanın Diğer Disiplinlerle İlgisi

Yapay zeka alanının diğer pek çok disiplinle yakın ilgisi vardır. Aşağıdaki şekilde bu ilgi betimlenmektedir.



Alıntı Notu: Görsel "Practical Machine Learning with Python (APress)" kitabından alınmıştır.

İstatistik: İstatistik temelde iki bölüme ayrılmaktadır:

- Betimsel istatistik (descriptive statistics)
- Sonuç çıkartıcı istatistik (inferential statistics)

Betimsel istatistik verilerin gruplanması, gösterilmesi ile ilgilidir. Yani betimleyici istatistik zaten var olan durumu betimlemektedir. Sonuç çıkartıcı istatistik ise kestirim yapmakla ilgilidir. Makine öğrenmesi istatistiğin kestirimsel

yöntemlerini açıkça kullanmaktadır. Örneğin regresyon analizi, kümeleme analizi, karar ağaçları, faktör analizi vs. gibi pek çok makine öğrenmesi yöntemi aslında istatistiğin bir konusu olarak ortaya çıkmıştır. Ancak bu istatistiksel yöntemler makine öğrenmesi temelinde genişletilmiş ve dinamik bir biçimde dönüştürülmüştür.

Vari Madenciliği (Data Mining): Veri madenciliği verilerin içerisindeki çeşitli faydalı bilgilerin bulunması, onların çekilerek elde edilmesi işlemleriyle ilgilenmektedir. Şüphesiz bu süreç istatistiksel birtakım bilgilerin yanı sıra yazılımsal uygulamaları da bünyesinde barındırmaktadır.

Bilgisayar Bilimleri (Computer Science): Bilgi işlem ve programlama etkinlikleriyle ilgili geniş kapsamlı bir bilim dalıdır.

İlgili Konudaki Özel Bilgiler: Şüphesiz her türlü algoritmik yöntem için bir biçimde hedeflenen konuda belli bir bilgi birikiminin var olması gereklidir. Örneğin ne kadar iyi programlama ve istatistik bilirseniz bilin görüntüselle verilerle çalışmak için bir görüntünün (resmin) nasıl bir organizasyona sahip olduğunu bilmeniz gereklidir. Ya da örneğin hiç muhasebe bilmeyen iyi bir programcının bir muhasebe programı yazabileceğini bekleyebilir miyiz?

Makine Öğrenmesi Uygulamaları İçin Kullanılan Programlama Dilleri

Veri bilimi (data science) ve makine öğrenmesi uygulamaları için pek çok dil tercih edilmektedir. Tercih konusunda ilk akla gelen dil şüphesiz Python'dur. Ancak C++, Java, C# gibi popüler programlama dilleri de bu amaçla gittikçe daha fazla kullanılır hale gelmektedir. Python programlama dili son on yıldır bir atak yaparak ve dünyanın en popüler ilk üç dili arasına girmiştir. Python dilinin özellikle veri bilimi ve makine öğrenmesi konusunda popüleritesinin neden bu kadar arttığına ilişkin görüşlerimiz şöyledir:

- Son yıllarda veri işleme ve verilerden kestirim yapma gereksiniminin gittikçe artmıştır ve Python dili de veri analizi için iyi bir araç olarak düşünülmektedir.
- Veri bilimi ve makine öğrenmesi için Python dilinden kullanılabilen pek çok kütüphane vardır. (Bu konudaki kütüphaneler diğer dillerden -şimdilik- daha fazladır.)
- Python nispeten basit bir dildir. Bu basitlik ana hatları veri analizi olan konularda uygulamacılara kolaylıklar sunmaktadır. Bu nedenle Python diğer disiplinlerden gelip de veri analizi ve makine öğrenmesi uygulaması yapmak isteyenler için nispeten daha kolay bir araç durumundadır.
- Python genel amaçlı bir programlama dili olmasının yanı sıra aynı zamanda matematiksel alana da yakın bir programlama dilidir. Yani Python'ın matematiksel alana yönelik ifade gücü (expressivity) popüler diğer programlama dillerinden daha yüksektir.
- Python dilinin çeşitli prestijli üniversitelerde "programlamaya giriş" gibi derslerde kullanılmaya başlanmış olması onun popüleritesini artırılmıştır. Python özellikle 3'lü versiyonlarla birlikte dikkate değer biçimde iyileştirilmiştir.
- Python dilinin veri analizi için diğer dillere göre daha erken yola çıktığı söylenebilir. Bu alanda algoritma geliştiren araştırmacılar algoritmalarını daha çok Python kullanarak gerçekleştirmiştir.

Peki Python dilinin veri analizi ve makine öğrenmesi konusunda hangi dezavantajları vardır? Bu dezavantajları da şöyle ifade edebiliriz:

- Python nispeten yavaş bir dildir. Bu yavaşlık büyük ölçüde Python dilinin dinamik tür sistemine sahip olmasından, Python programlarının yorumlayıcı yoluyla çalıştırılmasından ve dilin seviyesinin yüksek olmasından kaynaklanmaktadır. Her ne kadar veri analizi ve makine öğrenmesinde kullanılan kütüphaneler (numpy, pandas, scipy, keras gibi) asıl olarak C programlama dili ile yazılmış olsalar da bu C rutinlerinin Python'dan çağrılmaması ve diğer birtakım işlemler yavaşlığa yol açmaktadır.
- Python etkin (effective) bir programlama dili değildir. Dilin olanakları ince birtakım işlemlerin yapılabilmesine olanak sağlamamaktadır.

Pekiyi Python genel olarak yavaş bir dilse bu durum veri analizi ve makine öğrenmesi uygulamalarında bir sorun oluşturmaz mı? Bizim yanıtımız "bazı durumlarda" olacaktır.

İSTATİSTİĞE İLİŞKİN BAZI TEMEL BİLGİLER

Yapay zeka ve özellikle de makine öğrenmesi ile ilgili çalışmalar yapacak kişilerin belli düzeyde istatistiksel bilgilere sahip olması gerekmektedir. Şüphesiz istatistik pek çok alt alanı olan geniş bir bilim dalıdır. Bu nedenle istatistiksel konulara ilişkin pek çok ayrıntı vardır. Biz bu bölümde temel bilgiler vermekle yetineceğiz. Çeşitli ayrıntılar ilgili konuların anlatıldığı bölümde gerektiğiinde açıklanacaktır. (Örneğin "kümeleme analizi (cluster analysis)" aslında istatistikte çok uzun süredir incelenen bir konudur. Ancak son yıllarda makine öğrenmesi bağlamında konunun önemi çok daha fazla artmış ve bu bağlamda pek çok algoritmik yöntem geliştirilmiştir. Dolayısıyla örneğin kümeleme analizi çok değişkenli istatigin bir konusu olduğu halde biz bu tekniğin ayrıntılarını "denetimsiz öğrenme (unsupervised learning)" içerisinde ele alacağız.)

İstatistiksel Ölçek Türleri

İstatistikte ölçülen ya da ölçülmüş olan değerlerin sınıflarına genel olarak "ölçek (scale)" denilmektedir. Pek çok kişi ölçeklerin yalnızca sayısal olduğunu sanmaktadır. Halbuki ölçekler başka biçimlerde de karşımıza çıkmaktadır. İstatistikte ölçekler tipik olarak şu sınıflara ayrılmaktadır:

Kategorik (Nominal) Ölçekler: Bu ölçeklerde söz konusu kümenin elemanları kategorik olgulardır. Örneğin cinsiyet, renk, coğrafi bölge gibi. Bu ölçekteki ölçülen ya da ifade edilen değerlerin sayısal karşılıkları yoktur. Örneğin "kadınlarla erkekler arasında sigara içme miktarı arasında anlamlı bir fark olup olmadığını" anlamak için gerçekleştirilen bir araştırmada ölçülmesi istenen değişkenlerden "cinsiyet" kategorik (nominal) bir ölçüye ilişkindir. Benzer biçimde kişilerin renk tercihleriyle ilgili bir araştırmada renkler (siyah, beyaz, kırmızı gibi) kategorik bir ölçekle ifade edilirler.

Sırasal (Ordinal) Ölçekler: Bu ölçeklerdeki değerler de birer kategori belirtmekle birlikte bu kategoriler arasında büyüklik-küçüklik ilişkisi söz konusudur. Örneğin eğitim durumu için kategorik değerler "ilköğretim", "lise", "üniversite" olabilir ve bunlar arasında sıra ilişkisi vardır. Bu nedenle "eğitim durumu" bir sıralı ölçek belirtmektedir.

Aralıklı (Interval) Ölçekler: Aralıklı ölçekler sayısal bilgi içerirler. Bu tür ölçeklerde iki puan arasındaki fark aynı miktar uzaklığı ya da yakınlığı ifade eder. Örneğin bir teste 20 puan alan 10 puan alandan beli miktarda daha iyidir. 30 puan alan da 20 puan alandan aynı miktar kadar daha iyidir. Bu tür ölçeklerde mutlak sıfır noktası yoktur. Başka bir deyişle bu tür ölçeklerde "sıfır" yokluğu ya da mevcut olmamayı belirtmemektedir. Alınan puanlar her zaman belli bir göreli orijine göre anlamlıdır. Örneğin aslında sınavlardan alınan puanlar böyle bir ölçek türündedir. Sınavdan sıfır alınabilir. Ancak bu sıfır o kişinin o konu hakkında hiçbir şey bilmediği anlamına gelmez. Yani mutlak sıfır değildir. Ya da örneğin ısı belirten "derece (celcius)" bir aralık ölçüği belirtmektedir. 50 derece ile 40 derece arasındaki ısı farkı 40 derece ile 30 derece arasındaki fark kadardır ancak sıfır derece isının olmadığı anlamına gelmez.

Oransal (Ratio) Ölçekler: Bu ölçekler de sayısal bilgi içerirler. Oransal ölçekler aralık ölçeklerin tüm özelliklerine sahiptirler. Ancak ek olarak oransal ölçeklerde mutlak bir sıfır noktası da vardır. Dolayısıyla puanlar arasındaki oranlar mutlak olarak anlamlıdır. Örneğin uzunluk, kütle gibi temel fiziksel özellikler oransal ölçek türlerindendir. Bir nesnenin uzunluğunun sıfır olması onun uzunluğunun olmadığı, kütlesinin sıfır olması da onun kütlesinin olmadığı anlamına gelmektedir.

Provides:	Nominal	Ordinal	Interval	Ratio
The "order" of values is known		✓	✓	✓
"Counts," aka "Frequency of Distribution"	✓	✓	✓	✓
Mode	✓	✓	✓	✓
Median		✓	✓	✓
Mean			✓	✓
Can quantify the difference between each value			✓	✓
Can add or subtract values			✓	✓
Can multiple and divide values				✓
Has "true zero"				✓

Summary of data types and scale measures

Akıntı Notu: GörSEL <https://www.mymarketresearchmethods.com/wp-content/uploads/2016/05/summary-of-data-types-and-scales.png> adresinden alınmıştır.

Aritmetik Ortalama, Medyan, Mod, Standart Sapma ve Varyans Kavramları

Betimsel istatistikte bir grup verinin ortalaması için çeşitli ortalama hesaplama yöntemleri (merkezi eğilim ölçütleri) kullanılabilmektedir. Bu yöntemlerden en önemlilerinden biri aritmetik ortalamadır. Bildiğiniz gibi aritmetik ortalama değerlerin toplamının değer sayısına bölünmesiyle elde edilmektedir:

$$\bar{X} = \frac{\sum_{i=1}^n X_i}{N} = \frac{X_1 + X_2 + X_3 + \dots + X_n}{N}$$

Aritmetik ortalama için Python'da statistics modülündeki mean fonksiyonu bulundurulmuştur. Bu fonksiyon dolaşılabilir herhangi bir nesneyi parametre olarak alabilmektedir. Örneğin:

In [20]: `import statistics`

In [21]: `a = [3, 6, 2, 8, 5]`

In [22]: `statistics.mean(a)`

Out[22]: 4.8

Ancak numpy içerisindeki mean fonksiyonu daha fazla olanaklara sahiptir. Örneğin:

In [23]: `import numpy as np`

In [24]: `a = np.array([3, 6, 2, 8, 5])`

In [25]: `np.mean(a)`

Out[25]: 4.8

Numpy'in mean fonksiyonu iki boyutlu (ya da daha fazla boyutlu) dizilerden satır ve sütun temelinde ortalamaları da hesaplayabilmektedir. Örneğin:

```
In [36]: a = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])  
In [37]: np.mean(a)  
Out[37]: 5.0  
  
In [38]: np.mean(a, axis=0)  
Out[38]: array([4., 5., 6.])  
  
In [39]: np.mean(a, axis=1)  
Out[39]: array([2., 5., 8.])
```

Pandas kütüphanesinde aritmetik ortalama için Series sınıfının mean metodu kullanılmaktadır. Örneğin:

```
In [1]: import pandas as pd  
  
In [2]: s = pd.Series([3, 6, 2, 8, 5])  
  
In [3]: s.mean()  
Out[3]: 4.8
```

Aritmetik ortalama ancak sıralı (ordinal) ya da oransal (ratio) ölçekler için kullanılabilir. Aritmetik ortalamaların üç değerlerden çok etkilenebileceğine dikkat ediniz. Bu nedenle bazı durumlarda aritmetik ortalama yerine medyan değeri de tercih edilebilmektedir. Medyan küçükten büyüğe sıraya dizilmiş değerlerin ortasında bulunan değere denilmektedir. Eleman sayısı çift ise medyan değeri ortadaki iki elemanın toplamının yarısı ile hesaplanmaktadır.

Medyan hesabı için Python standart kütüphenesindeki median fonksiyonu kullanılmaktadır. Örneğin:

```
In [13]: a = [1, 5, 9, 2, 7]  
  
In [14]: statistics.median(a)  
Out[14]: 5  
  
In [15]: a = [1, 5, 9, 2, 7, 10]
```

Örneğin:

```
In [15]: a = [1, 5, 9, 2, 7, 10]  
  
In [16]: statistics.median(a)  
Out[16]: 6.0
```

Numpy kütüphanesinde de yine aynı işlem median fonksiyonuyla yapılmaktadır. Örneğin:

```
In [22]: a = np.array([[1, 5, 9, 4], [3, 8, 4, 7], [4, 9, 3, 8]])  
  
In [23]: np.median(a)  
Out[23]: 4.5  
  
In [24]: np.median(a, axis=0)  
Out[24]: array([3., 8., 4., 7.])  
  
In [25]: np.median(a, axis=1)  
Out[25]: array([4.5, 5.5, 6. ])
```

Medyan işlemi Pandas kütüphasında de Series sınıfının median metodu ile de yapılabilmektedir. Örneğin:

```
In [25]: np.median(a, axis=1)  
Out[25]: array([4.5, 5.5, 6. ])  
  
In [26]: a = pd.Series([1, 5, 9, 2, 7, 10])  
  
In [27]: a.median()  
Out[27]: 6.0
```

Merkezi ölçü olarak aritmetik ortalama ile medyan arasında nasıl bir farklılık vardır? Aritmetik ortalamada tüm değerlerin işlemde etkili olduğuna ancak medyanda etkili olmadığına dikkat ediniz. Dağılım içerisindeki en büyük değeri daha da büütsek bunun medyan hesabına bir etkisi olabilir mi? Ayrıca aritmetik ortalamanın üç değerlerden çok etkilenebildiğini ancak medyanın etkilenmediğini de belirtmek istiyoruz. Medyan işlemi sıraya dizme gerektirdiği için algoritma ancak $O(n \log n)$ karmaşıklıkta gerçekleştirilebilmektedir. Halbuki aritmetik ortalama $O(n)$ karmaşıklıkta hesaplanabilmektedir. Medyanın da sıralı, aralıklı ve oransal ölçekteki bilgilere uyarlanabileceğine dikkat ediniz.

Bir dağılımin modu en çok yinelenen değeri belirtir. Mod özellikle kategorik ve sıralı ölçeklerde ortalamanın yerini tutan bir işlem olarak kullanılmaktadır. `mod` değeri Python standart kütüphanesindeki `statistics` modülü içerisindeki `mode` fonksiyonuyla elde edilebilmektedir. Örneğin:

```
In [8]: a = [3, 5, 7, 3, 8, 150, 4, 7, 1, 7]
```

```
In [9]: statistics.mode(a)
```

```
Out[9]: 7
```

Eğer en çok yinelenen değerler birden fazla ise `mode` fonksiyonu bunlardan ilk karşılaştığı değeri vermektedir. Eşit sayıda tüm yinelenen mod değerlerini elde etmek için `multimode` fonksiyonu kullanılmaktadır. Örneğin:

```
In [9]: statistics.mode(a)
```

```
Out[9]: 7
```

```
In [10]: a = [1, 2, 2, 1, 3, 3, 6, 8, 6]
```

```
In [11]: statistics.multimode(a)
```

```
Out[11]: [1, 2, 3, 6]
```

Numpy kütüphanesinde `mode` işlemi için hazır bir fonksiyon bulunmamaktadır. Ancak `scipy` kütüphanesindeki `stats` modülünde `mode` fonksiyonu vardır. Bu fonksiyon `axis` parametresi de alarak `mode` işlemi yapar. Fonksiyon `ModeResult` türünden bir sınıf nesnesine geri dönmektedir. Sınıfın `mode` örnek özniteliği mod değerlerini, `count` örnek özniteliği ise onların sayılarını vermektedir. Örneğin:

```
import numpy as np
import scipy.stats

a = np.random.randint(0, 10, (10, 10))
print(a)

mr = scipy.stats.mode(a, axis=0)
print()
print(mr.mode)
print(mr.count)
```

Programdan şöyle bir çıktı elde edilmiştir:

```
[[0 0 8 7 0 3 5 9 4 6]
 [4 4 9 0 3 5 0 2 7 3]
 [6 1 7 1 4 4 4 9 4 0]
 [5 4 3 6 7 5 2 7 6 0]
 [6 6 7 5 7 1 7 3 7 1]
 [1 3 5 2 4 4 5 7 7 4]
 [2 3 9 3 8 0 6 9 3 7]
 [9 0 5 5 3 1 6 5 4 1]
 [3 2 2 7 0 8 1 5 0 1]
 [2 0 4 1 1 6 7 7 3 2]]

[[2 0 5 1 0 1 5 7 4 1]]
[[2 3 2 2 2 2 3 3 3]]
```

Eksen içerisinde birden fazla aynı değer varsa `mod` olarak fonksiyon en küçük değeri vermektedir. Örneğin:

```

import numpy as np
import scipy.stats

a = np.random.choice(['A', 'B', 'C', 'D', 'E'], (10, 10))
mr = scipy.stats.mode(a)
print(a)
print()
print(mr.mode)
print(mr.count)

```

Şöyledir bir çıktı elde edilmiştir:

```

[['D' 'E' 'E' 'E' 'B' 'E' 'C' 'B' 'E' 'A']
 ['B' 'E' 'B' 'E' 'B' 'D' 'A' 'E' 'B' 'B']
 ['D' 'D' 'E' 'D' 'D' 'C' 'D' 'B' 'D' 'A']
 ['E' 'C' 'E' 'E' 'A' 'E' 'A' 'E' 'A' 'C']
 ['C' 'A' 'D' 'B' 'A' 'E' 'A' 'B' 'E' 'E']
 ['D' 'D' 'B' 'D' 'C' 'E' 'A' 'A' 'D' 'C']
 ['E' 'B' 'C' 'D' 'E' 'C' 'D' 'C' 'B' 'C']
 ['D' 'D' 'B' 'D' 'C' 'A' 'B' 'C' 'B' 'D']
 ['A' 'A' 'D' 'D' 'A' 'D' 'B' 'E' 'E' 'E']
 ['D' 'E' 'B' 'A' 'E' 'E' 'C' 'E' 'B' 'B']]
[[['D' 'D' 'B' 'D' 'A' 'E' 'A' 'E' 'B' 'C']]
 [[5 3 4 5 3 5 4 4 4 3]]]

```

Aritmetik ortalama, medyan ve mod değerlerine "merkezi eğilim ölçütleri (measures of central tendency)" denilmektedir. Bu değerler dağılımin merkezine yönelik bilgileri bize verir. Bunların dışında ayrıca değerlerin merkezden ne kadar uzaklıkta konumlandığı da önemli bir bilgidir. Bu bilgiyi veren değerlere ise "merkezi yayılım ölçütleri (measures of dispersion)" denilmektedir. Örneğin bir ülkede kişi başına düşen milli gelir ortalamasının 10000 dolar olduğunu varsayıyalım. Bu ortalama herkesin eline yılda 10000 dolar geçtiği anlamına gelmez değil mi? Bu ülkede gelirler birbirlerine yakın da olabilir gelir dağılımında büyük bir adaletsizlik de olabilir.

Merkezi yayılım ölçütleri ortalamadan ortalama uzaklığını hesap etme temeline dayanmaktadır. Ancak bu hesabı yaparken dikkat etmek gereklidir. Örneğin eğer değerleri ortalamadan çıkartıp toplamlarının ortalamasını alırsak 0 elde ederiz. Bu da bize bir bilgi vermez. Örneğin:

```

In [27]: print(a)
[1 0 4 3 3 3 8 2 5 1 2 7 7 3 2 7 8 2 1 0]

In [28]: np.sum(np.mean(a) - a) / len(a)
Out[28]: 1.7763568394002506e-16

```

Sonuç 0'a çok yakındır. Yuvarlama hatasından dolayı 0 olamamıştır. Şimdi akılınızda mutlak değer almak gelebilir. Örneğin:

```

In [29]: a = np.random.randint(0, 10, 20)

In [30]: print(a)
[4 3 1 5 6 0 7 8 3 0 2 9 5 5 1 7 3 6 1 9]

In [31]: np.sum(np.abs(np.mean(a) - a)) / len(a)
Out[31]: 2.45

```

Gördüğünüz gibi ortalamaya yakınlık konusunda daha iyi bir ölçü elde etmiş olduk. Ancak ortalama mutlak değer yöntemi de aslında ortalamaya uzaklık hesabı için çok iyi bir yöntem değildir. Özellikle normal dağılımda bu yöntem bizi amacımızdan saptırabilir. Ayrıca ortalama mutlak değeri aynı olan birden fazla dağılım söz konusu olduğunda bunların aralarındaki farklılıklar da bu yöntemde iyi açığa çıkartılamamaktadır. İşte bu nedenle merkezi yayılım ölçüsü olarak genellikle "standart sapma (standard deviation)" kullanılmaktadır. Standart sapmada kare alma işlemi değerleri daha fazla farklılaştırılmaktadır.

Bir dağılımın standart sapması değerlerin ortalamadan farklarının karelerinin ortalamasının karekökü alınarak hesaplanır. Değerlerin sayısı n olmak üzere ortalama alınırken anakütle için N değerine örneklem için $N - 1$ değerine bölüm kullanılmaktadır:

$$\sigma = \sqrt{\frac{\sum(X_i - \bar{X})^2}{N}}$$

Anakütle standart sapmasının sigma simgesiyle gösterildiğine dikkat ediniz. Bir anakütleden çekilen örnek söz konusuysa bölme işlemi n 'e değil $N - 1$ 'e yapılmaktadır:

$$s = \sqrt{\frac{\sum(X_i - \bar{X})^2}{N - 1}}$$

Örnek için $N - 1$ değerine bölmeye "Bessel düzeltmesi (Bessel's correction)" denilmektedir. Bessel düzeltmesinin anlamı üzerinde burada durmayacağımız. Bunun için Internet'te pek çok kaynak bulabilirsiniz. Düzeltmedeki $N - 1$ değerine "serbestlik derecesi (degrees of freedom)" de denilmektedir. Şimdi standart sapma için basit bir fonksiyon yazalım:

```
import numpy as np

def stdev(a):
    return np.sqrt(np.sum((a - np.mean(a)) ** 2) / len(a))
```

Tabii aslında Python statistics modülündeki pstdev fonksiyonu anakütle için standart sapma hesabını yapmaktadır. Eğer örneklem standart sapması hesaplanacaksa (yeni $N - 1$ değerine bölme isteniyorsa) bu durumda stdev fonksiyonu kullanılmalıdır. Örneğin:

```
In [33]: a = [1, 2, 3, 4, 5]
In [34]: statistics.pstdev(a)
Out[34]: 1.4142135623730951
In [35]: statistics.stdev(a)
Out[35]: 1.5811388300841898
```

Numpy kütüphanesindeki std fonksiyonu axis parametresi de alarak standart sapmayı hesaplamaktadır. Fonksiyonun ddof (delta degrees of freedom) parametresi bölümün kaçını belirtir. Bu parametrenin default değeri 0'dır. Yani bölüm N değerine (anakütle standart sapması) yapılmaktadır. Örneğin:

```
In [40]: a = np.random.randint(0, 100, (10, 10))
In [41]: a
Out[41]:
array([[25, 1, 44, 12, 25, 70, 35, 17, 58, 91],
       [14, 97, 12, 24, 21, 83, 4, 93, 90, 4],
       [88, 20, 67, 53, 87, 29, 21, 75, 84, 30],
       [48, 38, 98, 11, 66, 38, 78, 59, 82, 32],
       [ 9, 88, 30, 68, 51, 24, 43, 80, 75, 75],
       [92, 59, 82, 46, 17, 72, 28, 11, 16, 74],
       [33, 98, 62, 86, 43, 66, 87, 3, 73, 82],
       [ 9, 89, 8, 51, 66, 95, 80, 95, 17, 57],
       [44, 68, 80, 23, 67, 46, 14, 77, 53, 13],
       [82, 85, 58, 96, 98, 39, 76, 94, 50, 56]])

In [42]: np.std(a, axis=1)
Out[42]:
array([26.64882737, 38.6, 26.8, 25.17141236, 25.57361922,
       28.62533843, 27.74905404, 32.95770016, 23.82120904, 20.06589146])
```

Pandas kütüphanesinde Series ve DataFrame sınıflarının da std metotları vardır. Bu metodların ddof parametreleri default 1'dir (örnek standart sapması). Örneğin:

```
In [47]: import pandas as pd
In [48]: s = pd.Series([1, 2, 3, 4, 5])
In [49]: s.std()
Out[49]: 1.5811388300841898

In [64]: df = pd.DataFrame(np.random.randint(0, 100, (10, 10)))
In [65]: df
Out[65]:
   0   1   2   3   4   5   6   7   8   9
0  71  95  81  53  61  31  80  39  73  77
1  76   0  62  89   0  18  23  72  15  85
2  95  64  13  49   5  73  18  44  58  29
3  67  78  29  11  37  15  35  19  60  75
4  80  20  63  21  51  83  54  61  99  67
5  33  26  72  87   2  86  21  89  89  46
6  37  61  64   7  67  90  70  67  93  97
7  16  52  81  93  44  68  88  82  46  94
8  51  79   0  73  27  63   4  22  43  54
9  55  86  92  51  94  19  27  74  70  15

In [66]: df.std()
Out[66]:
0    24.301120
1    31.395152
2    31.027049
3    32.273828
4    31.011826
5    30.489342
6    29.028721
7    24.587486
8    25.868041
9    27.444692
dtype: float64
```

Standart sapmanın karesine varyans denilmektedir. Varyans işlemi statistics modülündeki pvariance ve variance fonksiyonlarıyla, Numpy kütüphanesindeki var fonksiyonuyla, Pandas kütüphanesindeki Series ve DataFrame sınıflarının var metotlarıyla yapılabilmektedir. Örneğin:

```

In [87]: a = [1, 2, 3, 4, 5]
In [88]: statistics.pvariance(a)
Out[88]: 2

In [89]: np.var(a)
Out[89]: 2.0

In [90]: s = pd.Series(a)

In [91]: s.var()
Out[91]: 2.5

In [92]: df = pd.DataFrame(np.random.randint(0, 10, (10, 10)))

In [93]: df
Out[93]:
   0  1  2  3  4  5  6  7  8  9
0  5  8  6  9  7  5  1  4  2  9
1  4  1  4  5  7  4  9  6  3  9
2  3  4  6  0  2  3  8  7  7  4
3  0  1  4  2  3  2  8  6  3  2
4  1  0  1  5  5  5  0  4  5  0
5  5  1  7  1  5  2  4  7  0  4
6  0  5  9  1  5  0  9  7  7  4
7  7  2  5  0  5  4  4  5  6  6
8  5  8  8  0  4  8  4  3  1  5
9  4  0  3  3  7  3  4  8  9  3

In [94]: df.var()
Out[94]:
0      5.600000
1      9.555556
2      5.788889
3      8.711111
4      2.888889
5      4.711111
6     10.544444
7      2.677778
8      8.677778
9      8.044444
dtype: float64

```

Olasılık Kavramı

Bir olsunun gerçekleşme beklenisini anlatan olasılığın pek çok tanımı yapılmaktadır. Bir paranın atılması durumunda tura gelme olasılığının 0.5 (%50) olduğunu biliriz. Ancak bir parayı 10 kez attığımızda 5 kere tura geleceğinin bir garantisini yoktur. O halde neden paranın atılması tura gelme olasılığı 0.5'tir? İşte olasılığın en yaygın kullanılan tanımlarından biri göreli sıklık tanımıdır. Paranın atılması gibi bir rassal olay n defa yinelendiğinde ve bu n sayısı artırıldığında "n / tura gelme sayısı" gittikçe 0.5'e yakınsayacaktır. Para 1000000 kere atıldığındaysa yine yazı ile tura gelme sayısı aynı olmaz. Ancak tura gelme oranı gitgide 0.5'e yakınsar. O halde biz bir paranın atılması durumunda tura gelme olasılığının 0.5 olması demekle aslında bir limit durumunu kastetmiş oluyoruz.

Aşağıda 0 ile 1 arasında rastgele sayı üretecek yazı-tura atan örnek bir fonksiyon görüyorsunuz:

```

import random

def head_tail(n):
    head = 0
    for _ in range(n):
        head += random.randint(0, 1) # tail = 0, head = 1
    return head / n

```

Fonksiyonun parametresi paranın kaç kere atılacağını belirtmektedir. Şimdi çeşitli denemeler yapalım:

```

In [15]: head_tail(1)
Out[15]: 0.0

In [16]: head_tail(10)
Out[16]: 0.3

In [17]: head_tail(100)
Out[17]: 0.47

In [18]: head_tail(1000)
Out[18]: 0.467

In [19]: head_tail(10_000)
Out[19]: 0.5011

In [20]: head_tail(100_000)
Out[20]: 0.50393

In [21]: head_tail(1000_000)
Out[21]: 0.500036

In [22]: head_tail(10_000_000)
Out[22]: 0.4998814

In [23]: head_tail(100_000_000)
Out[23]: 0.50002317

```

Gördüğünüz gibi para atım sayısı arttıkça tura gelme olasılığı da 0.5'e yakınsamaktadır. Bu olguya istatistikte "büyük sayılar yasası (law of large numbers)" da denilmektedir.

Rassal Deneyler, Örnek Uzayı ve Rassal Olaylar

Sonucu önceden kesin olarak belirlenemeyen deneylere "rassal deney (random experiment)" denilmektedir. Örneğin bir paranın ya da bir zarın atılması birer rassal deneydir. Rassal deneyler sonucunda olusabilecek tüm sonuçlara "örnek uzayı (sample space)" denilmektedir. Örneğin bir paranın atılması ile ilgili örnek uzay $S = \{Y, T\}$ bir zarın atılması ile ilgili örnek uzay ise $S = \{1, 2, 3, 4, 5, 6\}$ biçimindedir.

Örnek uzayın her bir alt kümesine "olay (event)" denilmeketedir. Örneğin bir zarın atılması durumunda bazı olaylar şunlar olabilir:

```

E1 = {1, 3}
E2 = {4, 5, 6}
E3 = {2}

```

Örnek uzaydaki tek elemanlı olaylara ise "basit olaylar (simple events)" denilmektedir. Yani basit olaylar örnek uzayıın elemanlarıdır. Örneğin zarın atılmasındaki basit olaylar şunlardır:

```

E1 = {1}
E2 = {2}
E3 = {3}
E4 = {4}
E5 = {5}
E6 = {6}

```

E rassal olayının olasılığı $s(E) / s(S)$ biçiminde ifade edilir. Bu aksiyom büyük sayılar yasasına göre anlamlı ve aynı zamanda da sezgisel olarak da kabul edilebilir bir aksiyomdur. Örneğin bir zar atımı için E olayı $E = \{3, 4\}$ olsun. Bu durumda zar atıldığında 3 ya da 4 gelme olasılığı $P(E) = 2/6 = 1/3$ olacaktır. $P(S) = 1$ olduğuna ve $P(\text{boş küme}) = 0$ olduğuna dikkat ediniz. Dolayısıyla E olayının olasılığı aynı zamanda $P(E) = 1 - P(E')$ biçimindedir.

Rassal Değişkenler (Random Variables)

Rassal olaylar birer küme belirttiği için matematiksel işlemlere uygun değildir. Bu nedenle kümeler yerine sayısal değerlerin kullanılabilmesi için rassal değişken kavramından faydalılmaktadır. Bir rassal değişken örnek uzayının her bir elemanını (yani bir basit olayını) bir gerçek sayıya eşleyen bir fonksiyondur. Örneğin:

$$X: S \rightarrow R$$

Burada X rassal değişkeni (yani fonksiyonu) S örnek uzayındaki her bir olayı bir gerçek sayıya eşlemektedir. Rassal değişkenler genellikle anlatımlarda sözcüklerle ifade edilseler de aslında fonksiyon belirtirler. Örneğin X rassal değişkeni için biz "bir toplulukta rastgele seçilen bir kişinin boy uzunluğu" diyebiliriz. Bu durumda aslında o toplulukta kişiler vardır. Bu kişiler örnek uzayı oluşturmaktadır. X rassal değişkeni de oradaki bireyleri onların boy uzunluklarıyla eşleyen bir fonksiyondur. Örneğin Y rassal değişkeni için de biz "rastgele seçilen bir sözcüğün karakter uzunluğu" diyebiliriz. Bu durumda örnek uzay tüm sözcüklerden oluşmaktadır. Y rassal değişkeni de bu sözcükleri onların karakter uzunluklarına eşleyen bir fonksiyon durumundadır. Rassal değişkenler sayesinde artık kümeler yerine sayılarla konuşabildiğimize dikkat ediniz.

Mademki örnek uzayındaki her olayın bir olasılığı var o halde bir rassal değişkenin belli bir değeri almasının da bir olasılığı vardır. $X: S \rightarrow R$ ve E de bir olay olmak üzere $P(X=N)$ olasılığının eşdeğeri şöyledir:

$$P(X = N) = P(w \in S | X(w) = N)$$

Bu eşitlik şu anlamda gelmektedir: X rassal değişkeninin N değerini alma olasılığı S kümesindeki X fonksiyonunu N değerine eşleyen elemanların oluşturduğu olayın olasılığına eşittir. Yani örneğin X rassal değişkeni iki zarın atılması durumunda üste gelen değerlerin toplamını belirtiyor olsun. Bu durumda $P(X = 7)$ aslında $P(\{(1, 6), (2, 5), (3, 4), (4, 3), (5, 2), (6, 1)\})$ ile aynı anlamdadır. Bundan sonra biz de kümelerle konuşmak yerine rassal değişkenlerle konuşmayı tercih edeceğiz. Yani bizim için X rassal değişkeninin N değerine eşit olması aslında X fonksiyonuyla N 'e eşlenen kümeyi belirtecektir.

Kesikli ve Sürekli Rassal Değişkenler

Eğer bir rassal değişken belli bir aralıkta tüm gerçek değerleri değil de yalnızca bazı değerleri alabiliyorsa bu tür rassal değişkenlere kesikli (discrete) rassal değişkenler denilmektedir. Örneğin iki zar atıldığında üste gelen sayıların toplamını belirten X rassal değişkeni $[2, 12]$ aralığında yalnızca belirli değerleri alabilmektedir, bu rassal değişken $[2, 12]$ aralığında tüm gerçek sayı değerlerini alamamaktadır.

Eğer rassal değişken belli bir aralıkta tüm gerçek sayı değerlerinden oluşabiliyorsa bu tür rassal değişkenlere de "sürekli (contiguous)" rassal değişkenler denilmektedir. Y rassal değişkeni bir topluluktan rastgele seçilen bir kişinin kilosunu belirtiyor olsun. Bu Y rassal değişkeni sürekli bir rassal değişkendir. Çünkü teorik olarak belli aralıkta tüm gerçek sayı değerlerini alabilmektedir. (Yani örneğin kişinin kilosu 76.234567 olabileceği gibi 83.2323728765 de olabilir.)

Kesikli rassal değişkenlerin olasılık değerlerinin hesaplanması nokta temelli olarak ve tamsayı işlemleriyle yapılmaktadır. Ancak sürekli rassal değişkenlerin olasılıklarının hesaplanması ancak aralık temelli ve gerçek sayılar kullanılarak yapılabilir ki bu aralık temelli hesaplama da akla integral hesabı getirmektedir. Örneğin topluluktan rastgele seçilen kişinin kilosunu belirten K rassal değişkeninin 72 olma olasılığı aslında sıfırdır. Çünkü kişinin kilosu sonsuz sayıda değerden biri olabilir. Oysa 72 sayısı gerçek sayı doğrusunda yalnızca tek bir nokta belirtmektedir (sayı / sonsuz'un 0 olduğuna dikkat ediniz. Örneğin kişinin kilosu 72.000000000000001 olabilir, 71.999999999999999 olabilir ama tam 72 olmayabilir.) O halde sürekli rassal değişkenlerin olasılıklarını biz ancak aralık temelli olarak ve bir integral hesapla bulabiliyoruz. Bu örneğimizde K rassal değişkeninin 72 olma olasılığı 0'dır, ancak 71 ile 73 arasında olma olasılığı sıfır değildir. Tabii integral hesap için bir fonksiyona gereksinimimiz olacaktır. Integral hesap yapmakta kullandığımız bu tür fonksiyonlara "olasılık yoğunluk fonksiyonları (probability density function)" denilmektedir.

Sürekli Rassal Değişkenlerin Olasılık Yoğunluk Fonksiyonları

Sürekli rassal değişkenler için olasılık yoğunluk fonksiyonları (probability density functions) sürekli rassal değişkenlerin belli aralıktaki olasılıklarını integral hesabı ile bulmak için kullanılan fonksiyonlardır. Aşağıdaki fonksiyonu inceleyiniz:

$$f_X(x) = \lim_{\Delta x \rightarrow 0} \frac{P(x \leq X \leq x + \Delta x)}{\Delta x}$$

Burada f fonksiyonu x ve $x + \Delta x$ arasında değer alabilen ve her noktada türevlenebilen bir fonksiyon olsun. Bu f fonksiyonu aşağıdaki özellikleri sağlıyorsa ona olasılık yoğunluk fonksiyonu denilmektedir:

1) $f_X(x) \geq 0$

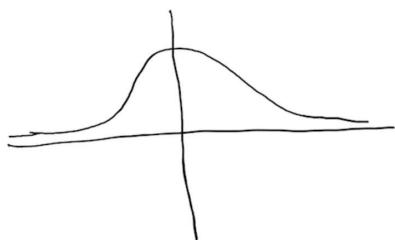
2) $\int_{-\infty}^{\infty} f_X(x) dx = 1$

3) $P(x_1 < X \leq x_2) = \int_{x_1}^{x_2} f_X(x) dx = F_X(x_2) - F_X(x_1)$

4) $F_X(x) = P(X \leq x) = \int_{-\infty}^x f_X(s) ds$

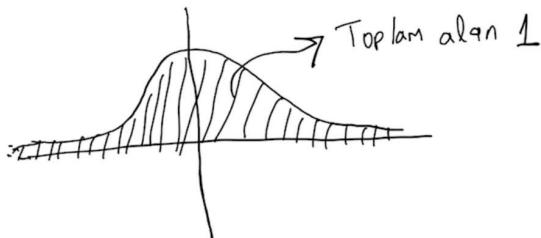
Alıntı Notu: Bu görsel avys.omu.edu.tr adresinden elde edilmiştir.

Birinci özellik olasılık yoğunluk fonksiyonlarının her zaman 0 ya da 0'dan büyük bir değer vermesi gerektiğini belirtir. Başka bir deyişle bu fonksiyonların grafikleri X ekseninin üzerinde kalmaktadır. İkinci özellik eksi sonsuzdan artı sonsuza kadar olasılık yoğunluk fonksiyonlarının eğri altında kalan alanının 1 olması gerektiğini belirtmektedir. Hiçbir olasılığın 1'den büyük ve 0'dan küçük olamayacağına dikkat ediniz. Üçüncü özellik bir rassal değişkenin belli bir aralıktaki olasılığının, olasılık yoğunluk fonksiyonun o aralıktaki integrali ile hesaplanabileceğini belirtmektedir. Yani başka bir deyişle bir rassal değişkenin belli bir aralıkta olma olasılığı o rassal değişkene ilişkin olasılık yoğunluk fonksiyonunun o aralıkta eğri altında kalan alanına (intergraline) eşittir. Dördüncü özellik X rassal değişkeninin x gibi bir değerden küçük olma olasılığının o rassal değişkenin olasılık yoğunluk fonksiyonun x değeriin solundaki eğri altında kalan alanına eşit olduğunu belirtmektedir. Aşağıda bir olasılık yoğunluk fonksiyonun bu özellikler bakımından değerlendirilmesini görüyorsunuz:

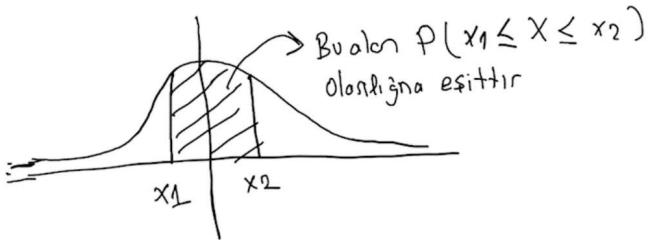


Buradaki f fonksiyonunun X ekseninin üzerinde kaldığını görüyorsunuz (1. Özellik).

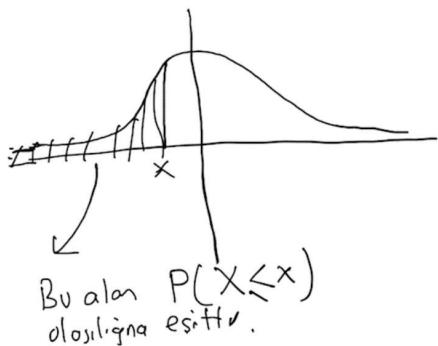
Eğri altında kalan toplam alan 1'dir (2. Özellik):



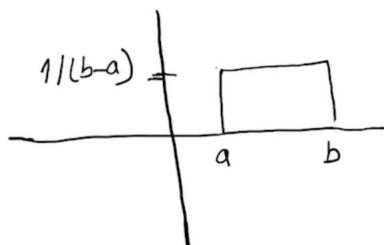
X rassal değişkeninin x_1 ile x_2 arasında olma olasılığı $X=x_1$ ve $X=x_2$ doğruları ile belirlenen alana eşittir (3. Özellik):



X rassal değişkenin x'ten küçük olma olasılığı $X = x$ doğrusunun solundaki tüm toplam alan kadardır (4. Özellik):



Aşağıdaki fonksiyonuna bakınız:



Burada ilgili fonksiyon bir olasılık yoğunluk fonksiyonudur. Toplam eğri altında kalan alanın 1 olduğunu dikkat ediniz. Bu durumda c değeri a ve b arasında olmak üzere $P(X \leq c)$ olasılığı $(c - a) / (b - a)$ biçimindedir.

Çok Karşılaşılan Bazı Sürekli Dağılımlar

Yukarıda da belirttiğimiz gibi sürekli rassal değişkenlerle olasılık hesabı yapabilmemiz için o rassal değişkene ilişkin bir olasılık yoğunluk fonksiyonunu biliyor olmamız gereklidir. Örneğin X rassal değişkenimiz şöyle olsun:

X: Rastgele seçilen kişinin kilosu

Burada aslında örnek uzayı tüm insanlardır. (Kurs notlarının yazıldığı sırada dünyada 8 milyar civarı insan olduğu tahmin edilmektedir.) O halde X rassal değişkeni tüm insanları tek tek onların kilolarına eşleyen bir fonksiyondur. Burada kilonun sürekli bir rassal değişken olduğuna dikkat ediniz. Şimdi bizim rastgele seçilen bir kişinin 60 ile 80 arasında bir kiloya sahip olma olasılığını bulmamız için bu rassal değişkene ilişkin olasılık yoğunluk fonksiyonunu biliyor olmamız gereklidir. İşte her ne kadar sonsuz sayıda olasılık yoğunluk fonksiyonu söz konusu olabilirse de belli olguların olasılık yoğunluk fonksiyonlarının belli bir kalıba uygun olduğu görülmüştür. Genellikle uygulamalarda anakütlenin olasılık yoğunluk fonksiyonunun (buna anakütle dağılımı da denilmektedir) bu kalıplardan biri içerisinde girdiği varsayılar. Örneğin doğada pek çok olguya ilişkin sürekli rassal değişkenlerin "normal dağılım" denilen dağılıma uyduğu ve bunların olasılık yoğunluk fonksiyonlarının Gauss fonksiyonuna benzettiği görülmektedir.

Peki bir rassal değişkenin olasılık yoğunluk fonksiyonunun ne olacağına nasıl karar verilmektedir? Bunun en pratik yöntemlerinden biri histogram çizip eğrinin neye benzediğine bakmak olabilir. Eğer eğri bildiğimiz bir dağılıma benziyorsa biz elimizdeki bilgilerle onun olasılık yoğunluk fonksiyonunu kolaylıkla belirleyebiliriz. Peki eğer

elimizdeki histogram bildiğimiz hiçbir dağılımın şekline benzemiyorsa bu durumda ne yapabiliriz? İşte bu durumda birikimli dağılım fonksiyonundan hareketle rassal değişkenin olasılık yoğunluk fonksiyonu bulunabilmektedir. Bunun için şu makaleyi inceleyebilirsiniz: <https://online.stat.psu.edu/stat414/lesson/22/22.1>

Yukarıda da belirttiğimiz gibi sürekli bir rassal değişkenin olasılık yoğunluk fonksiyonuna kısaca o rassal değişkenin dağılımı da denilmektedir. İşte biz de bu bölümde doğada doğrudan ya da dolaylı olarak çok karşılaştığımız bazı olasılık dağılımlarını (yani olasılık yoğunluk fonksiyonlarını) açıklayacağız.

Normal Dağılım

Şüphesiz normal dağılım doğada en çok karşılaşılan sürekli dağılımdır. Bu dağılıma Gauss dağılımı da denilmektedir. Gerçekten boy, kilo, zeka gibi pek çok özellik normal dağılıma eğilimindedir. Normal dağılımın doğada neden bu kadar çok karşılaşıldığının önemli bir nedeni de merkezi limit teoremdir (central limit theorem). Bu teoreme göre bir ana kütleden çekilen örnek ortalamaları normal dağılıma eğilimindedir.

Normal dağılıma ilişkin olasılık yoğunluk fonksiyonu şöyledir:

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Fonksiyonda μ ve σ biçiminde iki parametrenin olduğunu görüp sunuz. μ dağılımın ortalamasını, σ ise standart sapmasını belirtmektedir. Gauss fonksiyonundaki μ eğrinin orta noktasının X eksenindeki yeri üzerine, σ ise eğrinin genişliği üzerinde etkili olmaktadır. $\mu = 0$ ve $\sigma = 1$ olan normal dağılıma standart normal dağılım denilmektedir. μ ve σ değerleri farklı olan normal dağılımlar standart normal dağılımlara dönüştürülebilmektedir. Gerçekten de bilgisayarların yoğun kullanılmadığı zamanlarda eğer bir dağılımin normal olduğu sonucuna varılmışa oradaki hesaplar onun standart normal dağılıma dönüştürülmesiyle gerçekleştirilmektedir.

Şimdi Gauss eğrisini çizdiren bir fonksiyon yazalım:

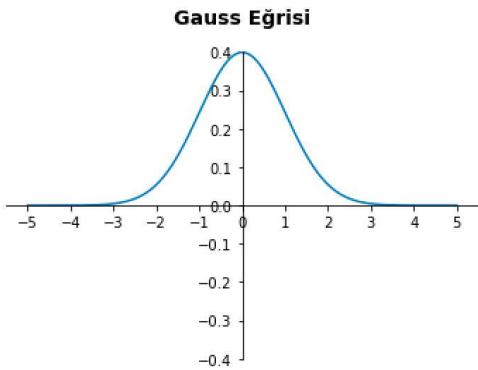
```
import numpy as np
import matplotlib.pyplot as plt

def draw_gauss(mu = 0, sigma = 1):
    x = np.linspace(-5, 5, 1000)
    y = 1 / np.sqrt(2 * np.pi * sigma ** 2) * np.e ** (- (x - mu) ** 2 / (2 * sigma ** 2))

    axis = plt.gca()
    axis.set_title('Gauss Eğrisi', fontsize=14, fontweight='bold', pad=20)
    axis.set_yticks([-0.4, 0.4])
    axis.set_xticks(range(-5, 6))
    axis.spines['left'].set_position('center')
    axis.spines['bottom'].set_position('center')
    axis.spines['top'].set_color(None)
    axis.spines['right'].set_color(None)

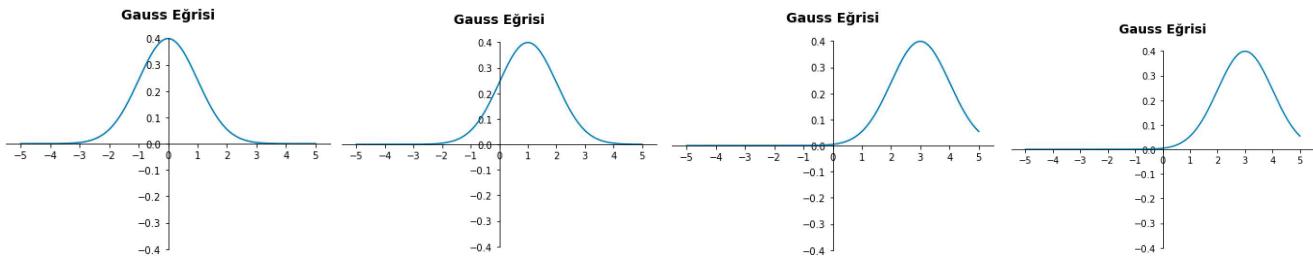
    plt.plot(x, y)
    plt.show()

draw_gauss()
```



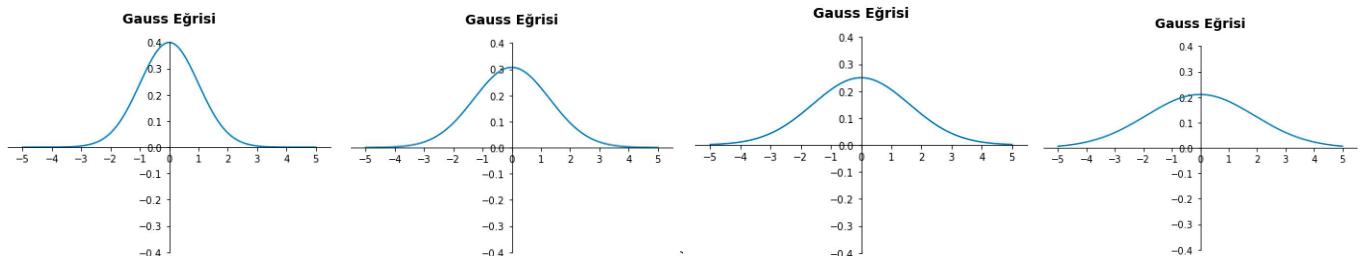
Şimdi çeşitli mü değerleri için birden fazla grafik çizelim:

```
draw_gauss(mu=0)
draw_gauss(mu=1)
draw_gauss(mu=2)
draw_gauss(mu=3)
```



Şimdi de mu değerini sabit tutup sigma (standart sapma) değerini değiştirelim:

```
draw_gauss(sigma=1)
draw_gauss(sigma=1.3)
draw_gauss(sigma=1.6)
draw_gauss(sigma=1.9)
```



Gördüğünüz gibi dağılımin standart sapma değeri değiştirildiğinde Gauss eğrisi şişmanlaşmaktadır. Standart sapmanın ortalamadan uzaklığı ilişkin bir değer belirttiğine dikkat ediniz. Yukarıdaki draw_gauss fonksiyonunu daha parametrik bir biçimde de yazabiliriz:

```
import numpy as np
import matplotlib.pyplot as plt

def draw_gauss(mu = 0, sigma = 1, axispos = None):
    if axispos == None:
        axispos = mu
    x = np.linspace(axispos - 5 * sigma, axispos + 5 * sigma , 1000)
    y = 1 / np.sqrt(2 * np.pi * sigma ** 2) * np.e ** (- (x - mu) ** 2 / (2 * sigma ** 2))

    axis = plt.gca()
    axis.set_title('Gauss Eğrisi', fontsize=14, fontweight='bold', pad=20)
    axis.set_xlim([-0.4 / sigma, 0.4 / sigma])
```

```

axis.set_xticks(np.arange(int(mu - 5 * sigma), int(mu + 5 * sigma + sigma), sigma))
axis.spines['left'].set_position('center')
axis.spines['bottom'].set_position('center')
axis.spines['top'].set_color(None)
axis.spines['right'].set_color(None)

plt.plot(x, y)
plt.show()

draw_gauss(mu=100, sigma=2, axispos=100)

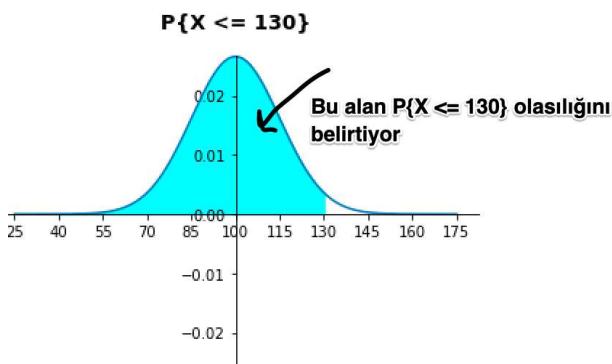
```

Fonksiyonun axispos parametresi Y eksenin çizileceği yeri belirtmektedir.

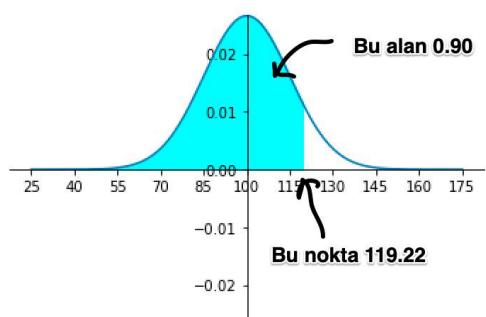
Python programcısı olarak normal dağılımla ilgili dört önemli işlemi yapabiliyor olmamız gereklidir. Bunlardan ilki belli bir X değeri için (standart normal dağılımdaki X değerlerine Z değerleri de denilmektedir) eğri altında kalan alanı bulmak. Belli bir değerden küçük olan eğri altında kalan alanın fonksiyona kümülatif dağılım fonksiyonu (cumulative distribution function) denilmektedir. Kümülatif dağılım fonksiyonunu şöyle ifade edebiliriz:

$$F(x) = P\{X \leq x\}$$

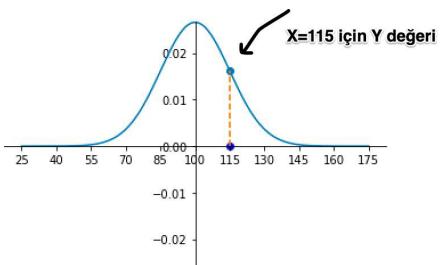
Örneğin ortalaması 100, standart sapması 15 olan bir normal dağılımda X'in 130'dan küçük olma olasılığı aslında F(130) değeridir.



İkincisi bunun tersi olan işlemidir. Yani kümülatif dağılım değerinin X değerine dönüştürülmesi. Örneğin ortalaması 100 standart sapması 15 olan bir normal dağılımda eğri altında kalan alanın 0.90 olduğu X değerinin bulunması istenebilir.



Üçüncü önemli işlem belli bir X değeri için Gauss fonksiyonundaki Y değerinin elde edilmesidir. Örneğin ortalaması 100 standart sapması 15 olan Gauss fonksiyonunda X = 115'e karşılık gelen Y değeri bulunmak istenebilir.



Nihayet son önemli işlem normal dağılıma uygun rastgele sayıların elde edilmesidir. Örneğin ortalaması 100, standart sapması 15 olan normal dağılıma ilişkin 1000 tane rastgele X değeri elde etmek isteyebiliriz.

Normal dağılıma ilişkin işlemler Python standart kütüphanesindeki `NormalDist` sınıfı ile yapılmaktadır. Bu sınıf nesnesi yaratılırken sınıfın `__init__` metodunda dağılımın ortalaması ve standart sapması girilir:

```
class statistics.NormalDist(mu=0.0, sigma=1.0)
```

Sınıfın `cdf` (cummulative distribution function) birikimli olasılık değerini bize verir. Örneğin ortalaması 100 standart sapması 15 olan normal dağılımda $P\{X \leq 130\}$ değeri şöyle edilir:

```
In [16]: import statistics
```

```
In [17]: nd = statistics.NormalDist(100, 15)
```

```
In [18]: nd.cdf(130)
```

```
Out[18]: 0.9772498680518208
```

Bu işlemin tersi yani birikimli olasılığı belli değere eşit olan X değeri `inv_cdf` metoduyla elde edilmektedir. Örneğin aynı dağılım için 0.95 birikimli olasılığa karşı gelen X değerini bulalım:

```
In [19]: nd.inv_cdf(0.95)
```

```
Out[19]: 124.67280440427207
```

Normal dağılımın olasılık yoğunluk fonksiyonu sınıfının `pdf` metoduyla temsil edilmiştir. Örneğin $X = 100$ için Y değeri şöyle elde edilebilir:

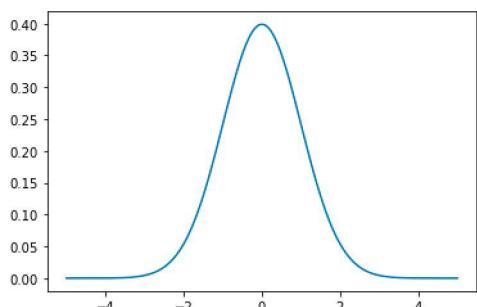
```
In [20]: nd.pdf(100)
```

```
Out[20]: 0.02659615202676218
```

Bu metot yoluyla Gauss eğrisini şöyle çizdirebiliriz:

```
import statistics
import matplotlib.pyplot as plt

nd = statistics.NormalDist()
x = [i * 0.001 for i in range(-5000, 5000)]
y = [nd.pdf(i) for i in x]
plt.plot(x, y)
```



Normal dağılıma ilişkin X rassal değerleri NormalDist sınıfının samples metodu ile elde edilebilir. Fonksiyon argüman olarak kaç rassal sayı üretileceğini alır; rassal sayılarından oluşan float bir listeye geri döner. Örneğin ortalaması 0, standart sapması 1 olan normal dağılıma ilişkin 10 tane X değeri elde edelim:

```
In [25]: nd.samples(10)
Out[25]:
[-0.4447316124433328,
 -0.7224285313191506,
 -0.1929356446968533,
 0.7350544803051189,
 -1.796611072264187,
 0.015753067776417353,
 0.13667455475444215,
 1.0778962661930618,
 -1.5975623709468105,
 1.4683966292696242]
```

Normal dağılıma ilişkin rassal sayıların elde edilmesi için çeşitli yöntemler önerilmektedir. Ancak en basit yöntem 0 ile 1 arasında rasgele sayı üretip inv_cdf metodu ile buna karşı gelen X değerini elde etmektedir. Örneğin:

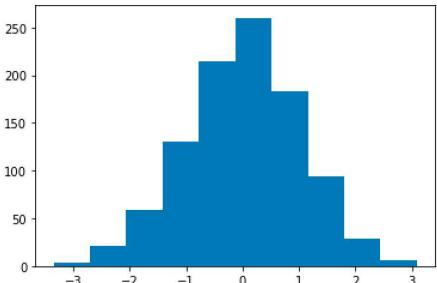
```
In [26]: import random
```

```
In [27]: nd.inv_cdf(random.random())
Out[27]: -0.4817643279327754
```

Şimdi de normal dağılıma ilişkin rassal sayılar elde edip histogramını çizelim:

```
import statistics
import matplotlib.pyplot as plt

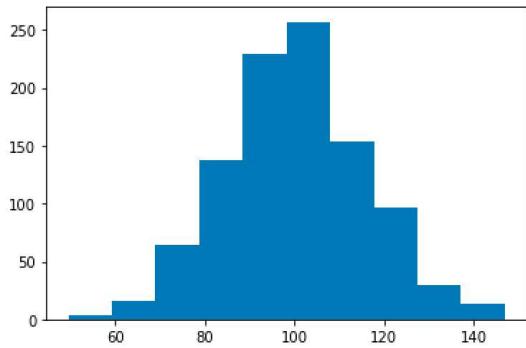
nd = statistics.NormalDist()
vals = nd.samples(1000)
plt.hist(vals)
```



Python standart kütüphanesindeki random modülünde bulunan gauss fonksiyonu da normal dağılmış rassal sayı üretmektedir. Örneğin:

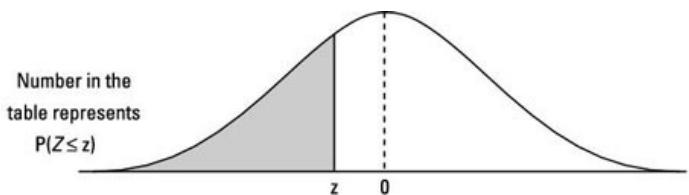
```
import random
import matplotlib.pyplot as plt

vals = [random.gauss(100, 15) for _ in range(1000)]
plt.hist(vals)
```



Aynı modülde thread güvenli olan ancak biraz daha yavaş çalışma potansiyelinde olan randomvariate isimli bir fonksiyon da bulunmaktadır.

Bilgisayarların istatistikte bu kadar yoğun kullanılmadığı zamanlarda bu tür işlemler elle yapılmıyordu. Bunun için ortalaması 0 olan standart sapması 1 olan standart normal dağılım için Z tabloları oluşturulmuştur. Bu tablolar belli bir Z değeri için (standart normal dağılımda X değeri yerine Z değeri denildiğini anımsayınız) birikimli olasılıkları göstermektedir. Aşağıda örnek bir Z tablosu görüyorsunuz:



z	0.00	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09
-3.6	.0002	.0002	.0001	.0001	.0001	.0001	.0001	.0001	.0001	.0001
-3.5	.0002	.0002	.0002	.0002	.0002	.0002	.0002	.0002	.0002	.0002
-3.4	.0003	.0003	.0003	.0003	.0003	.0003	.0003	.0003	.0003	.0002
-3.3	.0005	.0005	.0005	.0004	.0004	.0004	.0004	.0004	.0004	.0003
-3.2	.0007	.0007	.0006	.0006	.0006	.0006	.0006	.0005	.0005	.0005
-3.1	.0010	.0009	.0009	.0009	.0008	.0008	.0008	.0008	.0007	.0007
-3.0	.0013	.0013	.0013	.0012	.0012	.0011	.0011	.0011	.0010	.0010
-2.9	.0019	.0018	.0018	.0017	.0016	.0016	.0015	.0015	.0014	.0014
-2.8	.0026	.0025	.0024	.0023	.0023	.0022	.0021	.0021	.0020	.0019
-2.7	.0035	.0034	.0033	.0032	.0031	.0030	.0029	.0028	.0027	.0026
-2.6	.0047	.0045	.0044	.0043	.0041	.0040	.0039	.0038	.0037	.0036
-2.5	.0062	.0060	.0059	.0057	.0055	.0054	.0052	.0051	.0049	.0048
-2.4	.0082	.0080	.0078	.0075	.0073	.0071	.0069	.0068	.0066	.0064
-2.3	.0107	.0104	.0102	.0099	.0096	.0094	.0091	.0089	.0087	.0084
-2.2	.0139	.0136	.0132	.0129	.0125	.0122	.0119	.0116	.0113	.0110
-2.1	.0179	.0174	.0170	.0166	.0162	.0158	.0154	.0150	.0146	.0143
-2.0	.0228	.0222	.0217	.0212	.0207	.0202	.0197	.0192	.0188	.0183
-1.9	.0287	.0281	.0274	.0268	.0262	.0256	.0250	.0244	.0239	.0233
-1.8	.0359	.0351	.0344	.0336	.0329	.0322	.0314	.0307	.0301	.0294
-1.7	.0446	.0436	.0427	.0418	.0409	.0401	.0392	.0384	.0375	.0367
-1.6	.0548	.0537	.0526	.0516	.0505	.0495	.0485	.0475	.0465	.0455
-1.5	.0668	.0655	.0643	.0630	.0618	.0606	.0594	.0582	.0571	.0559
-1.4	.0808	.0793	.0778	.0764	.0749	.0735	.0721	.0708	.0694	.0681
-1.3	.0968	.0951	.0934	.0918	.0901	.0885	.0869	.0853	.0838	.0823
-1.2	.1151	.1131	.1112	.1093	.1075	.1056	.1038	.1020	.1003	.0985
-1.1	.1357	.1335	.1314	.1292	.1271	.1251	.1230	.1210	.1190	.1170
-1.0	.1587	.1562	.1539	.1515	.1492	.1469	.1446	.1423	.1401	.1379
-0.9	.1841	.1814	.1788	.1762	.1736	.1711	.1685	.1660	.1635	.1611
-0.8	.2119	.2090	.2061	.2033	.2005	.1977	.1949	.1922	.1894	.1867
-0.7	.2420	.2389	.2358	.2327	.2296	.2266	.2236	.2206	.2177	.2148
-0.6	.2743	.2709	.2676	.2643	.2611	.2578	.2546	.2514	.2483	.2451
-0.5	.3085	.3050	.3015	.2981	.2946	.2912	.2877	.2843	.2810	.2776
-0.4	.3446	.3409	.3372	.3336	.3300	.3264	.3228	.3192	.3156	.3121
-0.3	.3821	.3783	.3745	.3707	.3669	.3632	.3594	.3557	.3520	.3483
-0.2	.4207	.4168	.4129	.4090	.4052	.4013	.3974	.3936	.3897	.3859
-0.1	.4602	.4562	.4522	.4483	.4443	.4404	.4364	.4325	.4286	.4247
-0.0	.5000	.4960	.4920	.4880	.4840	.4801	.4761	.4721	.4681	.4641

Akıntı Notu: Görüel <https://www.dummies.com/education/math/statistics/how-to-use-the-z-table/> adresinden alınmıştır.

Burada ilgili satır ile sütunun birleşimi Z değerini, onların kesimlerinde bulunan değerler ise birikimli olasılık değerlerini belirtmektedir. Tabloda yalnızca normal dağılımın sol yarımlarının bulunduğuuna dikkat ediniz. Eğrinin iki yarısı simetrik olduğuna göre sağ yarısı için birikimli değerler de kolaylıkla elde edilebilmektedir. Yukarıdaki Z tablosunun benzerini çeken basit bir Python programını şöyle yazabiliriz:

```
import statistics

def disp_ztable():
    nd = statistics.NormalDist()

    print('-' * 76)
    print(' z' + ' + 6 * ' , end=' ')
    for i in range(10):
        f = i * 0.01
```

```

    print(f'{f:<7.2f}', end=' ')
print()
print('-' * 76)

z = -3.6
while z <= 0.05:
    print(f'{-0.0:<8.1f}' if z > 0 else f'{z:<8.1f}', end='')

    for i in range(10):
        f = i * 0.01
        cd = nd.cdf(z - f)
        print(f'{cd:<8.4f}'[1:], end=' ')
    print()
    z += 0.1

```

z	0.00	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09
-3.6	.0002	.0002	.0001	.0001	.0001	.0001	.0001	.0001	.0001	.0001
-3.5	.0002	.0002	.0002	.0002	.0002	.0002	.0002	.0002	.0002	.0002
-3.4	.0003	.0003	.0003	.0003	.0003	.0003	.0003	.0003	.0003	.0002
-3.3	.0005	.0005	.0005	.0004	.0004	.0004	.0004	.0004	.0004	.0003
-3.2	.0007	.0007	.0006	.0006	.0006	.0006	.0006	.0005	.0005	.0005
-3.1	.0010	.0009	.0009	.0008	.0008	.0008	.0008	.0007	.0007	.0007
-3.0	.0013	.0013	.0012	.0012	.0011	.0011	.0011	.0010	.0010	.0010
-2.9	.0019	.0018	.0018	.0017	.0016	.0016	.0015	.0015	.0014	.0014
-2.8	.0026	.0025	.0024	.0023	.0023	.0022	.0021	.0021	.0020	.0019
-2.7	.0035	.0034	.0033	.0032	.0031	.0030	.0029	.0028	.0027	.0026
-2.6	.0047	.0045	.0044	.0043	.0041	.0040	.0039	.0038	.0037	.0036
-2.5	.0062	.0060	.0059	.0057	.0055	.0054	.0052	.0051	.0049	.0048
-2.4	.0082	.0080	.0078	.0075	.0073	.0071	.0069	.0068	.0066	.0064
-2.3	.0107	.0104	.0102	.0099	.0096	.0094	.0091	.0089	.0087	.0084
-2.2	.0139	.0136	.0132	.0129	.0125	.0122	.0119	.0116	.0113	.0110
-2.1	.0179	.0174	.0170	.0166	.0162	.0158	.0154	.0150	.0146	.0143
-2.0	.0228	.0222	.0217	.0212	.0207	.0202	.0197	.0192	.0188	.0183
-1.9	.0287	.0281	.0274	.0268	.0262	.0256	.0250	.0244	.0239	.0233
-1.8	.0359	.0351	.0344	.0336	.0329	.0322	.0314	.0307	.0301	.0294
-1.7	.0446	.0436	.0427	.0418	.0409	.0401	.0392	.0384	.0375	.0367
-1.6	.0548	.0537	.0526	.0516	.0505	.0495	.0485	.0475	.0465	.0455
-1.5	.0668	.0655	.0643	.0630	.0618	.0606	.0594	.0582	.0571	.0559
-1.4	.0808	.0793	.0778	.0764	.0749	.0735	.0721	.0708	.0694	.0681
-1.3	.0968	.0951	.0934	.0918	.0901	.0885	.0869	.0853	.0838	.0823
-1.2	.1151	.1131	.1112	.1093	.1075	.1056	.1038	.1020	.1003	.0985
-1.1	.1357	.1335	.1314	.1292	.1271	.1251	.1230	.1210	.1190	.1170
-1.0	.1587	.1562	.1539	.1515	.1492	.1469	.1446	.1423	.1401	.1379
-0.9	.1841	.1814	.1788	.1762	.1736	.1711	.1685	.1660	.1635	.1611
-0.8	.2119	.2090	.2061	.2033	.2005	.1977	.1949	.1922	.1894	.1867
-0.7	.2420	.2389	.2358	.2327	.2296	.2266	.2236	.2206	.2177	.2148
-0.6	.2743	.2709	.2676	.2643	.2611	.2578	.2546	.2514	.2483	.2451
-0.5	.3085	.3050	.3015	.2981	.2946	.2912	.2877	.2843	.2810	.2776
-0.4	.3446	.3409	.3372	.3336	.3300	.3264	.3228	.3192	.3156	.3121
-0.3	.3821	.3783	.3745	.3707	.3669	.3632	.3594	.3557	.3520	.3483
-0.2	.4207	.4168	.4129	.4090	.4052	.4013	.3974	.3936	.3897	.3859
-0.1	.4602	.4562	.4522	.4483	.4443	.4404	.4364	.4325	.4286	.4247
-0.0	.5000	.4960	.4920	.4880	.4840	.4801	.4761	.4721	.4681	.4641

Bu tür kodlarda yuvarlama hatalarına dikkat ediniz. Python'da yuvarlama hatalarına maruz kalmadan noktalı sayılar üzerinde işlem yapmak için tasarılanmış decimal isimli standart bir modülün olduğunu anımsatmak istiyoruz. Ancak bu decimal türü Numpy ya da Pandas kütüphanelerinde kullanılamamaktadır.

Belli bir ortalama ve standart sapmaya ilişkin normal dağılımdaki X değerini standart normal dağılımdaki Z değerine dönüştürmek için (yani ortalaması 0, standart sapması 1 olan normal dağılımdaki X değerine dönüştürmek için) aşağıdaki işlem uygulanır:

$$Z = \frac{X - \mu}{\sigma}$$

Örneğin ortalaması 100 standart sapması 15 olan bir normal dağılımdaki $X = 125$ değerinin standart normal dağılımdaki karşılığı şöyle hesaplanmaktadır:

$$Z = \frac{125 - 100}{15} = 1.666$$

Python 3.9 ile birlikte NormalDist sınıfına bu dönüşümü yapan zscore isimli bir metot da eklenmiştir.

Normal dağılıma ilişkin işlemler scipy kütüphanesindeki `scipy.stats.norm` isimli "singleton" sınıf nesnesi ile de yapılabilmektedir. Scipy kütüphanesi numpy kütüphanesi kullanılarak gerçekleştirildiği için değerler üzerinde tek tek değil vektörel işlemler yapabilmektedir. `norm` nesnesinin ilişkin olduğu sınıfın `cdf` isimli metodu birikimli olasılık değerini elde etmekte kullanılmaktadır. Metodun birinci parametresi birikimli olasılığı hesaplanacak değerlerin bulunduğu dolaşılabilir nesneyi alır. İkinci ve üçüncü parametreler normal dağılımin ortalama ve standart sapmasını belirtmektedir. Örneğin:

```
from scipy.stats import norm

result = norm.cdf([100, 130, 120], 100, 15)
print(result)
```

Şöyle bir çıktı elde edilmiştir:

```
[0.5 0.97724987 0.90878878]
```

Bu işlemin tersi yani birikimli olasılığa karşı gelen X değerleri ilgili sınıfın `ppf` (percent point function) metoduyla elde edilmektedir:

```
from scipy.stats import norm

result = norm.ppf([0.5, 0.3, 0.05], 100, 15)
print(result)
```

Şöyle bir çıktı elde edilmiştir:

```
[100. 92.13399231 75.3271956 ]
```

Belli X değerleri için Gauss eğrisindeki Y değerlerinin elde edilmesi ilgili sınıfın `pdf` isimli metoduyla yapılmaktadır. Örneğin:

```
from scipy.stats import norm

result = norm.pdf([100, 120, 130], 100, 15)
print(result)
```

Bu işlemenin aşağıdaki gibi bir çıktı elde edilmiştir:

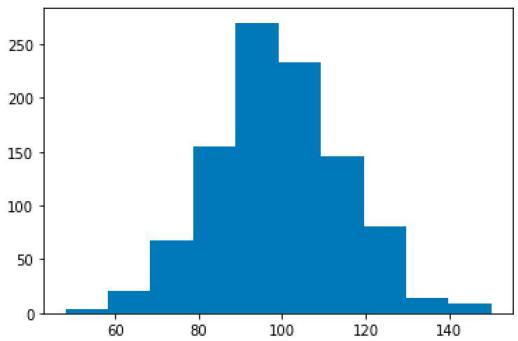
```
[0.02659615 0.010934 0.0035994 ]
```

Normal dağılıma ilişkin rastgele sayı elde edebilmek için ise ilgili sınıfın `rvs` metodu kullanılmaktadır. Örneğin:

```
from scipy.stats import norm
import matplotlib.pyplot as plt

result = norm.rvs(100, 15, 1000)
plt.hist(result)
```

Elde edilen histogram şöyledir:



rvs fonksiyonun üçüncü parametresi bir demet olarak da girilebilmektedir. Bu durumda çok boyutlu normal dağılmış rassal sayılar da üretilmektektir. Yine benzer biçimde numpy.random modülündeki normal isimli fonksiyon da normal dağılmış rastgele sayılar üretmektektir.

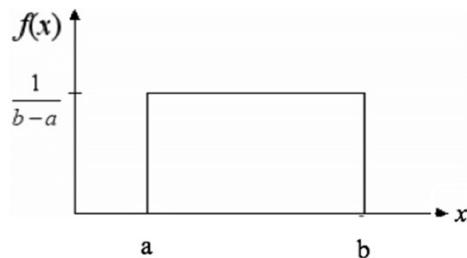
Sürekli Düzgün Dağılım (Continuous Uniform Distribution)

Düzgün dağılım herkesin aşina olduğu bir dağılımdır. Düzgün dağılımin olasılık yoğunluk fonksiyonu şöyledir:

$$f(x) = \begin{cases} \frac{1}{b-a} & \text{for } a \leq x \leq b, \\ 0 & \text{for } x < a \text{ or } x > b \end{cases}$$

Alıntı Notu: Görsel https://en.wikipedia.org/wiki/Continuous_uniform_distribution adresinden alınmıştır.

Olasılık yoğunluk fonksiyonun grafiği de şöyledir:



Alıntı Notu: Görsel <https://www.researchgate.net/publication/332236648/figure/fig8/AS:865450120445954@1583350796843/Continuous-uniform-distribution-Source-14.ppm> adresinden elde edilmiştir.

Bu grafikte dikdörtgenin içerisindeki alanın 1 olması gerektigine dikkat ediniz. Bu diktörtgensel alanda aynı uzunluktaki aralığın olasılığı aynı olacaktır.

Düzgün dağılımla işlemler scipy.stats kütüphanesindeki uniform isimli "singleton" nesne ile yapılabilmektedir. Nesnenin kullanımı norm nesnesiyle benzerdir. Örneğin:

```
from scipy.stats import uniform
import matplotlib.pyplot as plt

result = uniform.cdf([50, 60, 70], 0, 100)
print(result)

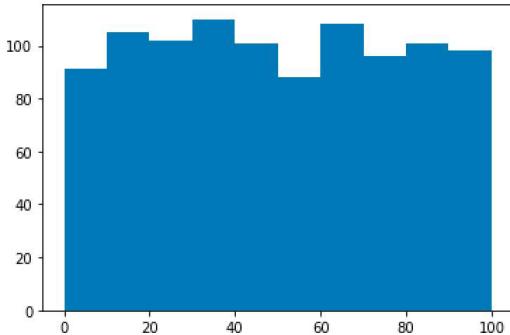
result = uniform.ppf([0.3, 0.5, 0.7], 0, 100)
print(result)

result = uniform.pdf([0.3, 0.5, 0.7], 0, 100)
print(result)

result = uniform.rvs(0, 100, 1000)
plt.hist(result)
```

Şöyledir bir çıktı elde edilmiştir:

```
[0.5 0.6 0.7]  
[30. 50. 70.]  
[0.01 0.01 0.01]
```



t Dağılımı (Student's t Distribution)

Özellikle hipotez testlerinde karşımıza çıkan diğer bir dağılım da t dağılımıdır. (Dağılım W. S. Gosset tarafından 1908 yılında geliştirilmiştir. Gosset makalesini "Student (Öğrenci)" takma adıyla yayımladığı için dağılım bu isimle de anılmaktadır.) t Dağılımı normal dağılıma benzerdir. Dağılımin olasılık yoğunluk fonksiyonunu biraz karışık olduğu gerekçesiyle burada vermeyeceğiz. Ancak t dağılımı normal dağılıma göre daha düz, daha az yüksek ancak daha geniş bir görünümdedir. Örneklem miktarı arttıkça t dağılımı standart normal dağılıma benzer. t dağılımindaki değerlere (yani X değerlerine) t değerleri denilmektedir. (Standart normal dağılımdaki X değerlerine Z değerleri denildiğini anımsayınız.) t dağılımının şekli "serbestlik derecesi (degrees of freedom)" denilen bir değere bağlıdır. Serbestlik derecesi örneklem büyüğünün bir eksigidir. Yani n örneklem büyüğü olmak üzere serbestlik derecesi şöyle hesaplanmaktadır:

$$df = n - 1$$

t dağılımının ortalaması standart normal dağılımda olduğu gibi 0'dır. Ancak standart sapması 1 değildir. 1'den biraz daha fazladır. t dağılımının standart sapması şöyle hesaplanır:

$$\sigma = \sqrt{\frac{df}{df - 2}}$$

Standart sapmanın 1'den büyük olması standart normal dağılıma göre eğrinin daha geniş (yani daha şişman) olmasına yol açmaktadır. Bu eşitlikte serbestlik derecesi olan df arttıkça standart sapmanın 1'e yaklaşmasına dikkat ediniz.

t dağılımı `scipy.stats` modülü içerisindeki `t` isimli nesne ile yapılmaktadır. Nesnenin ilişkin olduğu sınıfın `cdf` metodunu yine birikimli olasılığı hesaplar. `ppf` fonksiyonu ters işlemi yapmaktadır. `pdf` fonksiyonu ise X değerleri için olasılık yoğunluk fonksiyonun Y değerini vermektedir. Bu fonksiyonların hepsinin birinci parametreleri hesaplanacak değerleri, ikinci parametreleri serbestlik derecesini, üçüncü ver dördüncü parametreleri ise ortalama ve standart sapma değerlerini almaktadır. Örneğin:

```
from scipy.stats import t  
from scipy.stats import norm  
  
p = t.cdf([0, 1, 2], 10)  
print(p)
```

```
p = norm.cdf([0, 1, 2])
print(p)
```

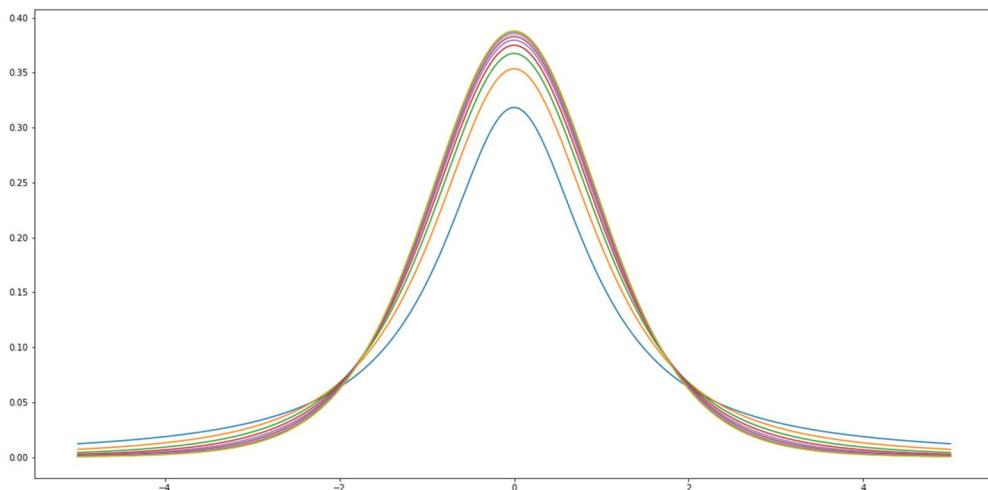
Programdan şöyle bir çıktı elde edilmiştir:

```
[0.5      0.82955343 0.96330598]
[0.5      0.84134475 0.97724987]
```

Şimdi de çeşitli serbestlik dereceleri için t dağılımının eğrilerini çizelim:

```
import numpy as np
from scipy.stats import t
import matplotlib.pyplot as plt

fig = plt.gcf()
fig.set_size_inches(20, 10)
x = np.linspace(-5, 5, 1000)
for df in range(1, 10):
    y = t.pdf(x, df)
    plt.plot(x, y)
```



Uygulamada t dağılımı anakütle parametresinin bilinmediği durumlarda örneklemden anakütle parametrelerinin tahmininde kullanılmaktadır.

Diger Sürekli Dağılımlar

İstatistikte çeşitli olguların olasılıklarını modellemek için ya da çeşitli konularda dolaylı olarak kullanılan pek çok sürekli dağılım vardır. Bu dağılımların büyük çoğunluğu için `scipy.stats` modülünde hazır nesneler bulunmaktadır. Temel işlemler yine ilgili sınıfların benzer metodlarıyla yapılmaktadır.

Merkezi Limit Teoremi (Central Limit Theorem)

Süphesiz sonuç çıkarıcı istatistiğin (inferential statistics) en önemli teoremi merkezi limit teoremidir. Bu teorem normal dağılımın bizim için neden bu kadar önemli olduğunu da göstermektedir. Teorem uzun süredir biliniyor olmasına karşın bu isimle ilk kez George Pólya tarafından 1920 yılında kullanılmıştır. Merkezi limit teoreminin biçimsel ifadesi biraz karmaşık bir görünümdedir. Biz burada sözel anlatımı üzerinde duracağız.

Bu teoreme göre bir anakülteden elde edilen örnek ortalamaları normal dağılma eğilimindedir. Eğer anakütle normal dağılmışsa küçük örneklerin ortalamaları da normal dağılır. Ancak ana kütle normal dağılmamışsa örnek ortalamalarının normal dağılması için örneklerin belli bir büyülükte (tipik olarak ≥ 30) olması gerekmektedir. Yine bu teoreme göre örnek ortalamalarının ortalaması ana kütle ortalamasına eşittir. Örnek ortalamalarının standart

sapması ise anakütle standart sapmasının örneklem büyülüğünün kareköküne bölümüne eşittir. Teoremin bu kısmını matematiksel olarak şöyle ifade edebiliriz:

$$\mu_{\bar{x}} = \mu$$

$$\sigma_{\bar{x}} = \frac{\sigma}{\sqrt{n}}$$

Buradaki n örneklem büyülüğünü belirtmektedir. n değeri örneklem büyülüğünü, N ise anakütle büyülüğünü belirtmek üzere,

$\frac{n}{N} > 0.05$ ise yani örneklem büyülüği anakütle büyülüğüne oranı %5'ten büyük ise örneklem standart sapması şöyle olur:

$$\sigma_{\bar{x}} = \frac{\sigma}{\sqrt{n}} \sqrt{\frac{N-n}{N-1}}$$

Burada $\sqrt{\frac{N-n}{N-1}}$ oranına düzeltme faktörü denilmektedir.

Terminoloji İlişkin Not: Biz burada "örnek (sample)" terimi ile "örneklem (sampling)" terimlerini farklı anlamlarda kullanacağız. Bir anakütleden seçilen belli bir büyülükteki tek bir alt küme için "örnek" terimini ancak anakütleden çekilen belli bir büyülükteki tüm alt küümeler için "örneklem" terimini tercih edeceğiz.

Şimdi merkezi limit teoremini deneme yoluyla doğrulamaya çalışalım. Önce anakütle normal dağılmış olsun ve küçük bir örnekler üzerinde çalışalım. Bunun için anakütleyi temsil eden 1000 tane normal dağılmış rastgele sayı üreteceğiz. Sonra bu rastgele sayılarından örneğin 5'lik ($n < 30$) rastgele 1000_000 örnek çekip onların ortalamalarına ilişkin histogram çizeceğiz. (Şüphesiz 1000 elemanlı bir kümenin 5'li tüm alt kümelerini ele alamayız. $C(1000, 5)$ çok büyük bir değerdir. Biz 5'li rastgele 1000_000 değer çekmekle yetineceğiz. Bunun bir hata kaynağı oluşturacağı muhakkaktır.

```
import numpy as np
import matplotlib.pyplot as plt

NPOPULATION = 1000
NSAMPLES = 1000_000
SAMPLE_SIZE = 5

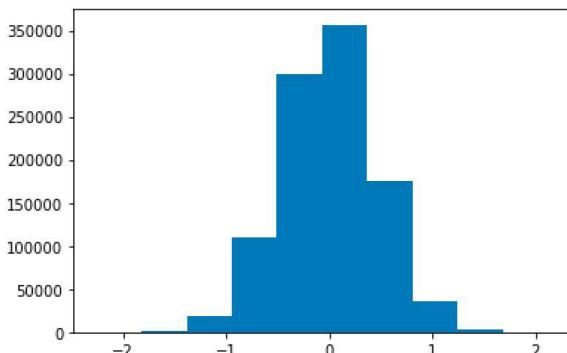
population = np.random.normal(0, 1, NPOPULATION)
samples = np.random.choice(population, (NSAMPLES, SAMPLE_SIZE))
smeans = np.mean(samples, axis=1)

plt.hist(smeans)
plt.show()

smeans_mean = np.mean(smeans)
population_mean = np.mean(population)
population_std = np.std(population)
smeans_std = np.std(smeans)
sigma_slash_sqrt_n = population_std / np.sqrt(SAMPLE_SIZE)

print(f'Örneklem ortalamalarının ortalaması: {smeans_mean}')
print(f'Anakütle ortalaması: {population_mean}')
print(f'Anakütle standart sapması: {population_std}')
print(f'Örneklem ortalamalarının standart sapması: {smeans_std}')
print(f'sigma / sqrt(n) değeri: {sigma_slash_sqrt_n}'')
```

Programın çalıştırılması sonucunda şöyle bir çıktı elde edilmiştir:



Örneklem ortalamalarının ortalaması: 0.006341131037852003
 Anakütle ortalaması: 0.006536944315343625
 Anakütle standart sapması: 1.0182981841666945
 Örneklem ortalamalarının standart sapması: 0.4561456393309987
 sigma / sqrt(n) değeri: 0.45539679223226576

Bu çıktıdan da gördüğünüz gibi örnek ortalamalarının histogramı Gauss eğrisine benzemektedir. Örneklem ortalaması ile anakütle ortalaması arasındaki ve örneklem standart sapması ile anakütleden hareketle elde edilen standart sapma arasındaki küçük fark alınan örneklerin toplam sayısının (1000_000) az olmasından ve yuvarlama hatalarından kaynaklanmaktadır.

Şimdi aynı denemeyi anakütenin normal dağılmadığı durum için yineleyelim. Bu kez anakütle düzgün dağılmış olsun ve örnek büyülüğünü 50 ($n \geq 30$) olarak seçelim. Yine anakütle büyülüğü 10000 olsun:

```
import numpy as np
import matplotlib.pyplot as plt

NPOPULATION = 10000
NSAMPLES = 1000_000
SAMPLE_SIZE = 50

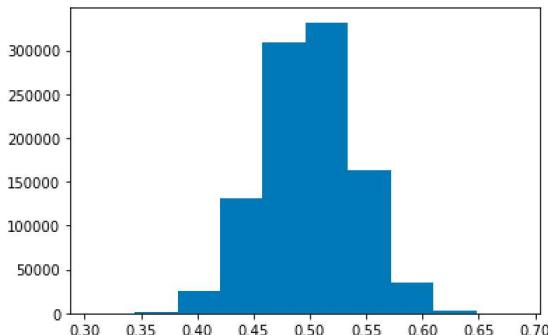
population = np.random.uniform(0, 1, NPOPULATION)
samples = np.random.choice(population, (NSAMPLES, SAMPLE_SIZE))
smeans = np.mean(samples, axis=1)

plt.hist(smeans)
plt.show()

smeans_mean = np.mean(smeans)
population_mean = np.mean(population)
population_std = np.std(population)
smeans_std = np.std(smeans)
sigma_slash_sqrt_n = population_std / np.sqrt(SAMPLE_SIZE)

print(f'Örneklem ortalamalarının ortalaması: {smeans_mean}')
print(f'Anakütle ortalaması: {population_mean}')
print(f'Anakütle standart sapması: {population_std}')
print(f'Örneklem ortalamalarının standart sapması: {smeans_std}')
print(f'sigma / sqrt(n) değeri: {sigma_slash_sqrt_n}')
```

Programın çalıştırılması sonucunda elde edilen çıktı şöyledir:



Örneklem ortalamalarının ortalaması: 0.4995729042905874
 Anakütle ortalaması: 0.4995699925304928
 Anakütle standart sapması: 0.29005119517418737
 Örneklem ortalamalarının standart sapması: 0.04108047000586231
 sigma / sqrt(n) değeri: 0.041019433399786136

Gördüğünüz gibi anakütlenin normal dağılmadığı ve örneklem büyülüğünün ≥ 30 olduğu durum için de merkezi limit teoremi deneyel olarak doğrulanıyor.

Merkezi limit teoreminin biçimsel (formal) ispatını burada yapmayacağız. Bunun için başka kaynaklara başvurmalısınız.

Normalliğinin Test Edilmesi

Parametrik istatistiksel yöntemlerde anakütle veya örneklem dağılımının normal olduğu varsayımları bulunmaktadır. Bu nedenle örneklem dağılımının normal olup olmadığından belli bir örneğe dayalı olarak test edilmesi gerekmektedir. Aslında normallik testi gözle üstünkörü yapılabılır. Bunun için belli sayıda örnek için histogram çizilip şeşlin Gauss eğrisine benzeyip benzemediğine bakılabilir.

Normallik testi için çeşitli istatistiksel hipotez testleri geliştirilmiştir. Bunların bazıları bakımlardan birbirlerine üstünlükleri ve zayıflıkları vardır. Burada biz konunun ayrıntılarına girmeyeceğiz. Normallik testlerinden birisi Kolmogorov-Smirnov testidir. Bu test scipy.stats modülündeki kstest fonksiyonuyla yapılabilmektedir.

```
scipy.stats.kstest(rvs, cdf, args=(), N=20, alternative='two-sided', mode='auto')
```

Kolmogorov-Smirnov testi parametrik olmayan istatistiksel bir hipotez testidir. Bu testte H_0 hipotezi söz konusu değerlerin düzgün dağılmış bir anakütleden geldiğini H_1 hipotezi ise söz konusu değerlerin düzgün dağılmış bir anakütleden gelmediği biçimindedir. Test işleminden sonra p değeri belirlenen kritik değerden (örneğin 0.05) küçükse H_0 hipotezi reddedilir, H_1 hipotezi kabul edilir. Bu durumda değerler normal dağılıma sahip olan bir anakütleden gelmemektedir. Eğer p değeri bu kritik değerden büyükse bu durumda H_0 hipotezi kabul edilir, H_1 hipotezi reddedilir. Bu da değerlerin normal dağılmış bir anakütleden çekildiği anlamına gelmektedir.

kstest fonksiyonunu KTestResult isimli bir sınıf nesnesine (isimli demet nesnesine) geri dönmektedir. Nesnenin statistic elemanı Kolmogorov-Smirnov test istatistiğini pvalue elemanı ise p değerini vermektedir. Burada uygulamacı p değerine bakmalı eğer bu p değeri belirlediği kritik değerden (alfa) düşükse H_0 hipotezini, yüksekse H_1 hipotezini kabul etmelidir. Yani dağılımin normal olduğunu kabul edebilmemiz için bu pvalue değerinin 0.05 gibi bir eşik değerinden büyük olması gerekmektedir.

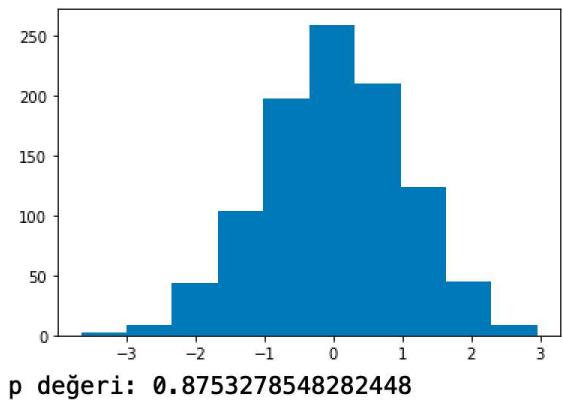
Şimdi test amaçlı 1000 tane standart normal dağılıma ilişkin rastgele sayı üretelim. Sonra bu sayılarla Kolmogorov-Smirnov testi uygulayalım:

```
from scipy.stats import kstest, norm
import matplotlib.pyplot as plt

a = norm.rvs(size=1000)
plt.hist(a)
plt.show()
```

```
tresult = kstest(a, 'norm')
print(f'p değeri: {tresult.pvalue}')
```

Programın çalıştırılması sonucunda şöyle bir çıktı elde edilmiştir:

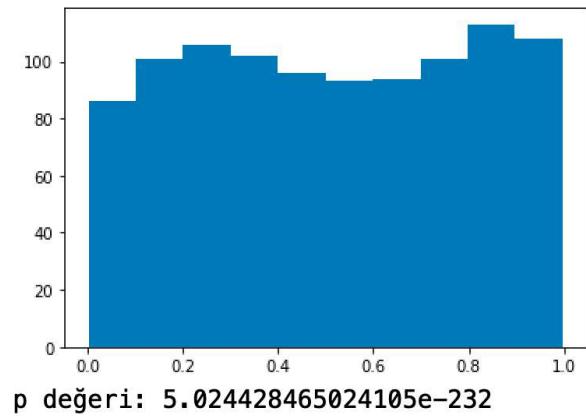


Burada p değerinin 0.05 gibi bir kritik değerden büyük olduğu görülmektedir. O halde bu verilerin normal dağıldığını ya da normal dağılmış bir anakütleden geldiğini varsayılabılır. Şimdi aynı testi düzgün dağılmış rastgele değerlerle gerçekleştirelim:

```
from scipy.stats import kstest, uniform
import matplotlib.pyplot as plt

a = uniform.rvs(size=1000)
plt.hist(a)
plt.show()
tresult = kstest(a, 'norm')
print(f'p değeri: {tresult.pvalue}')
```

Program çalıştırıldığında şöyle bir çıktı elde edilmiştir:



Göründüğü gibi p değeri 0'a çok yakındır. Bu durumda H₀ hipotezi reddedilmektedir. Yani veriler normal dağılıma uygun değildir ya da normal dağılmış bir anakütleden gelmemektedir.

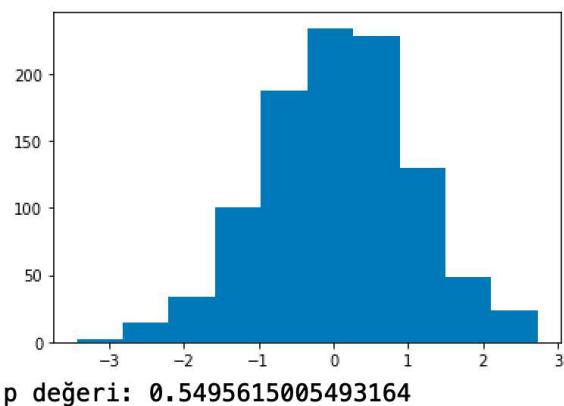
Diğer yaygın kullanılan normallik testlerinden biri de Shapiro-Wilk testidir. Burada biz bu iki testin teknik farklılıklarını üzerinde durmayacağız. Bunun için ilgili kaynaklara başvurabilirsiniz. Shapiro-Wilk testinde de H₀ hipotezi verilerin normal dağılıma ilişkin anakütleden gelmediğini, H₁ hipotezi verilerin normal dağılıma ilişkin bir anakütleden geldiğini biçimindedir. Karar yine test sonucunda elde edilen p değerinin kritik değerden küçük olup olmadığına göre verilmektedir.

Shapiro-Wilk testi `scipy.stats` modülü içerisindeki `shapiro` fonksiyonu kullanılmaktadır. Fonksiyon dağılım verilerini parametre olarak alır. Şimdi normal dağılmış rastgele değerlerden oluşan veri kümesi üzerinde Shapiro-Wilk testi uygulayalım:

```
from scipy.stats import shapiro, norm
import matplotlib.pyplot as plt

a = norm.rvs(size=1000)
plt.hist(a)
plt.show()
tresult = shapiro(a)
print(f'p değeri: {tresult.pvalue}')
```

Program çalıştırıldığında şöyle bir sonuç elde edilmiştir:

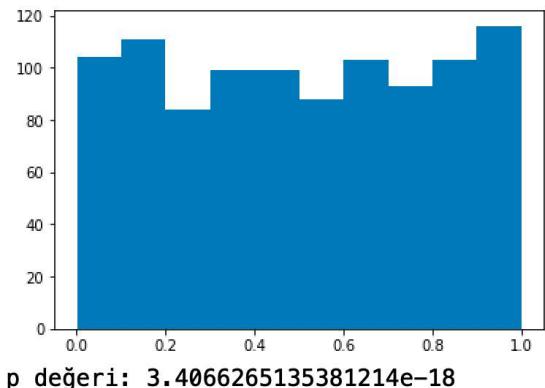


Göründüğü gibi p değeri oldukça yüksektir. Bu durumda H₀ kabul edilir. Dolayısıyla örnekleme değerleri normal dağılmış bir anakütleden gelmektedir. Yine aynı denemeyi düzgün dağılmış bir rassal değişken için deneyelim:

```
from scipy.stats import shapiro, uniform
import matplotlib.pyplot as plt

a = uniform.rvs(size=1000)
plt.hist(a)
plt.show()
tresult = shapiro(a)
print(f'p değeri: {tresult.pvalue}')
```

Program çalıştırıldığında şöyle bir sonuç elde edilmiştir:



p değerinin sıfıra çok yakın olduğunu görüyorsunuz. Bu durumda H₀ hipotezi reddedilmektedir. Yani değerler normal dağılmış bir anakütleden gelmemektedir.

Örnekten Hareketle Anakütle Ortalamasının Tahmin Edilmesi ve Güven Aralıkları

Güven aralıkları (confidence intervals) bir anakütleden çekilen örneğe bağlı olarak anakütle parametrelerinin belli bir güven düzeyi (confidence level) içerisinde aralıksal olarak belirlenmesi için kullanılan istatistiksel bir yöntemdir. Merkezi limit teoremine göre bir anakütleden çekilen örneklemelerin ortalamalarının normal dağıldığını görmüştük. O halde biz bir anakütleden rastgele bir örnek seçip onun ortalamasına bakarak anakütle ortalamasını belli bir güven düzeyinde tahmin edebiliriz.

Güven aralıklarının oluşturulması örnekleme dağılımına bakılarak yapılmaktadır. Eğer anakütle varyansı biliniyorsa anakütleden seçilen bir örneğin ortalamasından hareketle ana kütle parametreleri tahmin edilebilir. Örneğin elimizde ortalaması 100 standart sapması 15 olan normal dağılmış 1000 adet değer olsun. Biz de bu anakütleden 10 elemanlık rastgele bir örnek seçip onun ortalamasını bulalım:

```
import numpy as np
from scipy.stats import norm
import matplotlib.pyplot as plt

POPULATION_SIZE = 1000
SAMPLE_SIZE = 10

population = norm.rvs(100, 15, POPULATION_SIZE)
population_mean = np.mean(population)
population_std = np.std(population)
sample = np.random.choice(population, SAMPLE_SIZE)
sample_mean = np.mean(sample)

print(f'Anakütle ortalaması: {population_mean}')
print(f'Anakütle standart sapması: {population_std}')
print(f'Selezion örneğin ortalaması: {sample_mean}'')
```

Programın çalıştırılmasıyla şöyle bir çıktı elde edilmiştir:

```
Anakütle ortalaması: 99.96597670217074
Anakütle standart sapması: 14.996297931176969
Seçilen örneğin ortalaması: 104.10574797536367
```

Merkezi limit teoremine göre anakütle normal dağılırsa örnek ortalamalarının dağılımı da normaldir. Daha önce de belirttiğimiz gibi örneklem dağılımının ortalaması ve standart sapması şöyledir:

$$\mu_{\bar{x}} = \mu$$

$$\sigma_{\bar{x}} = \frac{\sigma}{\sqrt{n}}$$

O halde eğer bir anakütle standart sapmasını biliyorsak allığımız tek bir örneğin ortalamasından hareketle anakütle ortalamasını güven düzeyinde aralıksal olarak tahmin edebiliriz. Yukarıda verdigimiz örnekte anakütle standart sapmasını bildiğimizi varsayırsak örneklem dağılımının standart sapması şöyle olacaktır:

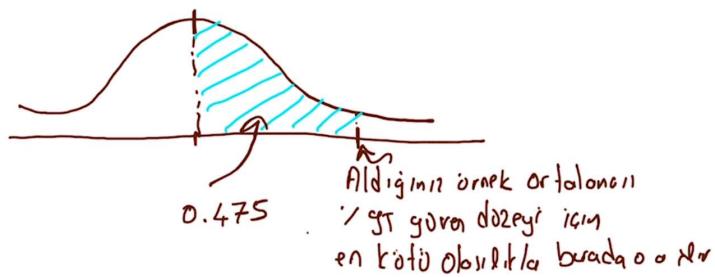
```
sampling_std = population_std / np.sqrt(SAMPLE_SIZE)
print(sampling_std)
```

Bu işlemden şöyle bir değer elde edilmiştir:

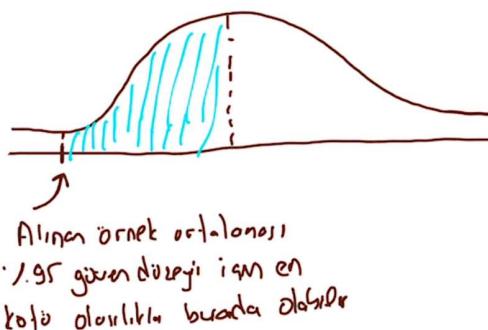
4.742245793299021

Bu durumda biz örneklem dağılımının standart sapmasını biliyoruz ancak örneklem dağılımının ortalamasını bilmiyoruz. Örneklem dağılımının ortalamasının anakütle ortalamasına eşit olması gerektiğini anımsayınız. Normal dağılım eğrisinde ortalama, eğrinin X ekseniye göre konumu üzerinde etkili olmaktadır. O halde tüm bu bilgilerden

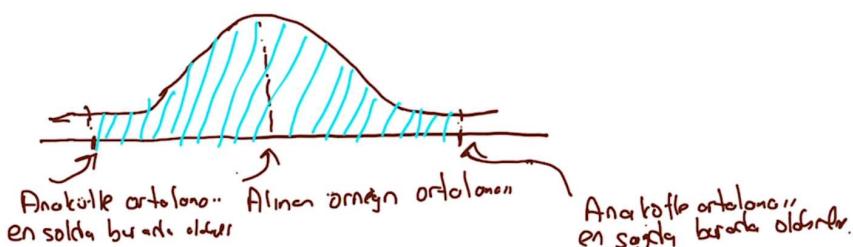
hareketle rastgele seçilen bir örnekten hareketle anakütle ortalamasının belli bir güven düzeyinde hangi aralıklarda olabileceğini belirleyebiliriz. Örneğin rastgele seçilen bir örnekten hareketle anakütle ortalamasının %95 güven düzeyinde hangi aralıkta olabileceğini anlamaya çalışalım. Aldığımız örnek en kötü olsılıkla anakütle ortalamasının 0.475 sağında olabilir. Aşağıdaki şekli inceleyiniz:



Yine alınan örnek en kötü olsılıkla anakütle ortalamasının 0.475 solunda olabilir:



Bu durumda anakütle ortalaması için 0.95 güven aralığı aslında alınan örnek ortalamasının solunda ve sağında 0.475 alana ilişkin X değerleridir:



Şimdi yukarıdaki örnek için 0.95 güven düzeyinde güven aralıklarını oluşturalım. Aldığımız örneğin ortalaması 104.10574797536367 idi. O halde ortalaması bu olan standart sapması 4.742245793299021 olan normal dağılımda 0.025 ve 0.975 birikimli olasılık değerini veren noktaları bulalım:

```
lower_bound = norm.ppf(0.025, 104.1057479753636, 4.742245793299021)
upper_bound = norm.ppf(0.975, 104.1057479753636, 4.742245793299021)
```

```
print(lower_bound, upper_bound)
```

Buradan şöyle bir çıktı elde edilmiştir:

```
94.81111701466094 113.40037893606626
```

O halde bu anakülteden çektiğimiz örneklem dayanarak anakütle ortalamasının %95 güven aralığında bu sınırlar içerisinde olabileceğini gördük. Aslında bunu standart normal dağılımın kullanılacağı biçimde şöyle de formülüze edebiliriz:

$$\bar{x} \pm z\sigma_{\bar{x}}$$

Buradaki z değeri güven düzeyine karşı gelen X eksenindeki değerdir. $\sigma_{\bar{X}}$ ise örneklem dağılımının standart sapması olan σ / \sqrt{n} değeridir.

Şimdi standart sapması 10 olan bir anakütleden çekilen 100'lük bir örneğin ortalamasının 65 olduğunu varsayıyalım. Anakütle ortalaması 0.95 güven düzeyi içerisinde şöyle hesaplanacaktır:

```
import numpy as np
from scipy.stats import norm

lower_bound = 65 + norm.ppf(0.975) * 10 / np.sqrt(100)
upper_bound = 65 - norm.ppf(0.975) * 10 / np.sqrt(100)
print(lower_bound, upper_bound)
```

Buradan şu çıktı elde edilmiştir:

63.04003601545995 66.95996398454005

Tabii aynı işlemi şöyle de yapabilirdik:

```
lower_bound = norm.ppf(0.975, 65, 10 / np.sqrt(100))
upper_bound = norm.ppf(0.025, 65, 10 / np.sqrt(100))
print(lower_bound, upper_bound)
```

Aslında güven aralıkları için norm nesnesine ilişkin sınıfta interval isimli bir metod da bulundurulmuştur. interval metodu üç parametre almaktadır. Metodun birinci parametresi güven aralığını, ikinci parametresi ortalama değeri, üçüncü parametresi ise standart sapmayı belirtir. Fonksiyon güven aralığının düşük ve yüksek değerlerine ilişkin bir demete geri dönmektedir. Bu durumda standart sapması 10 olan bir anakütleden çekilen 65 ortalamaya sahip 100'lük bir örnekten hareketle anakülenin ortalamasını %95 güven düzeyinde şöyle belirleyebiliriz:

```
import numpy as np
from scipy.stats import norm

ci = norm.interval(0.95, 65, 10 / np.sqrt(100))
print(ci)
```

Programın çalıştırılmasıyla şu çıktı elde edilmiştir:

(63.04003601545995, 66.95996398454005)

Merkezi limit teoremine göre eğer ana kütle normal dağılmamışsa ancak $n \geq 30$ koşulunu sağlayan örneklem dağılımlarının normal dağıldığı kabul edilmektedir. Yani örneklerimizdeki gibi anakütle ortalamasının tahmin edilmesi ve güven aralıklarının oluşturulması için şu iki koşuldan en az biri sağlanmalıdır:

- 1) Anakütle normal dağılmıştır ve örneklem dağılımı için $n < 30$ durumu söz konusudur.
- 2) Anakütle normal dağılmamıştır ve örneklem dağılımı için $n \geq 30$ durumu söz konusudur.

Pekiyi anakütle normal dağılmamışsa ve $n < 30$ ise ne olacaktır? İşte bu durumda biz yukarıda uygulandığı biçimde anakütle ortalamasını ve güven aralıklarını belirleyemeyiz. Bu durumda "parametrik olmayan" başka yöntemler kullanılabilmektedir. Ancak biz bu yöntemleri burada ele almayacağız.

Merkezi limit teoremine göre bizim anakütle ortalamasını örnekten hareketle tahmin edebilmemiz için anakütle standart sapmasını biliyor olmamız gereklidir. Pekiyi ya bunu bilmiyorsak ne yapabiliriz? Örneğin aşağıdaki gibi bir soruyu nasıl çözebiliriz?

İşte eğer anakütle standart sapması bilinmiyorsa bu durumda t dağılımı kullanılmaktadır. Alınan örneğin standart sapması sanki anakülenin standart sapmasıymış gibi ele alınmaktadır. t dağılımının bir serbestlik derecesi (degrees

of freedom) parametresinin olduğunu anımsayınız. Burada serbestlik derecesi örnek büyülüğünün 1 eksik değeridir. Anakütle standart sapmasının bilinmemesi durumunda da örneklem dağılımına ilişkin örneklem büyülüğu önemli olmaktadır. Eğer alınan örnek büyük değilse (< 30) anakütlenin normal dağılmış olması gerekmektedir. Eğer alınan örnek yeteri kadar büyükse (≥ 30) bu durumda anakütle dağılımı normal olmak zorunda değildir.

Şimdi elimizde bir anakütleden çekilmiş olan 35 elemandan oluşan aşağıdaki gibi bir örneklem olsun:

```
sample = np.array([101.93386212, 106.66664836, 127.72179427, 67.18904948,
    87.1273706 , 76.37932669, 87.99167058, 95.16206704,
    101.78211828, 80.71674993, 126.3793041 , 105.07860807,
    98.4475209 , 124.47749601, 82.79645255, 82.65166373,
    92.17531189, 117.31491413, 105.75232982, 94.46720598,
    100.3795159 , 94.34234528, 86.78805744, 97.79039692,
    81.77519378, 117.61282039, 109.08162784, 119.30896688,
    98.3008706 , 96.21075454, 100.52072909, 127.48794967,
    100.96706301, 104.24326515, 101.49111644])
```

Şimdi %95 güven düzeyinde anakütle ortalamasının hangi aralıklar içerisinde olabileceğini bulmaya çalışalım. Burada biz anakütle standart sapmasını bilmiyoruz. Anakütlenin normal dağılıp dağılmadığını da bilmemişimizi düşünelim. (Örnekten hareketle anakütlenin normal dağılıp dağılmadığını daha önce görmüş olduğumuz normalilik testleriyle belirleyebilirsiniz.) Bu durumda anakütle ortalaması için aralık tahmini t dağılımı kullanılarak yapılmalıdır. Öncelikle bu örneğin ortalamasını ve standart sapmasını bulalım:

```
sample_mean = np.mean(sample)
sample_std = np.std(sample)
```

Örneğin standart sapmasını anakütlenin standart sapması kabul ederek örneklem dağılımının standart sapmasını bulalım:

```
sampling_std = sample_std / np.sqrt(35)
```

Örnek büyülüğu 35 olduğuna göre serbestlik derecesi 34 olacaktır. Artık t dağılımından hareketle güven aralığını bulabiliriz:

```
from scipy.stats import t

lower_bound = t.ppf(0.025, 34, sample_mean, sampling_std)
upper_bound = t.ppf(0.975, 34, sample_mean, sampling_std)
print(lower_bound, upper_bound)
```

Şöyledir bir çıktı elde edilmiştir:

```
94.89296371821018 105.02201556521837
```

Tabii aynı işlemleri t nesnesine ilişkin sınıfın interval metoduyla da yapabiliyoruz:

```
ci = t.interval(0.95, 34, sample_mean, sampling_std)
print(ci)
```

YAPAY SİNİR AĞLARI

Yapay sinir ağları yapay zekanın ve makine öğrenmesinin önemli alt alanlarından biridir. Derin öğrenme (deep learning) de bir çeşit yapay sinir ağı yöntemidir. Bu bölümde yapay sinir ağları belli düzeylerde teorik bakımdan ele alınacak ve daha çok uygulamaları üzerinde durulacaktır.

Yapay Sinir Ağlarının Tarihi

Yapay Sinir ağlarının teorisi ilk zamanlar sinir bilimle (neuroscience), psikolojiyle ve matematikle uğraşan bilim adamları tarafından geliştirilmiştir. Yapay sinir ağları ilk kez Warren McCulloch ve Walter Pitts isimli kişiler tarafından 1943 yılında ortaya atılmıştır. 1940'lı yılların sonlarına doğru Donald Hebb isimli psikolog da "Hebbian Learning" kavramıyla, 1950'li yıllarda da Frank Rosenblatt isimli araştırmacı da "Perceptron" kavramıyla alana önemli katkılarında bulunmuştur. Bu yıllar henüz elektronik bilgisayarların çok yeni olduğu yıllardır. Halbuki yapay sinir ağlarına yönelik algoritmalar için önemli bir CPU gücü gerekmekteydi. Bu nedenle özellikle 1960 yıllarda bu konuda bir motivasyon eksikliği oluşmuştur. Yapay sinir ağları sonraki dönemlerde yeniden popüler olmaya başlamıştır. Derin öğrenme konusunun gündeme gelmesiyle de popüleritesi hepten artmıştır.

Yapay Sinir Ağlarının Uygulama Alanları

Yapay Sinir ağlarının pek çok farklı alanda uygulaması vardır. Örneğin:

- Örütü Tanıma
- Finansal Uygulamalar (Portföy yönetimi, Kredi değerlendirmesi vs.)
- Endüstriyel problemlerin çözümü
- Biyomedikal Mühendisliğinde (Örneğin hastalığa tanı koyma ve tedavi planı oluşturma)
- Optimizasyon problemlerinde
- Pazarlama Süreçlerinde

İnsanın Sinir Sistemi

Sinir sistemi iki bölüme ayrılmaktadır:

- Merkezi Sinir Sistemi (Central Nervous System)
- Çevresel Sinir Sistemi (Peripheral Nervous System)

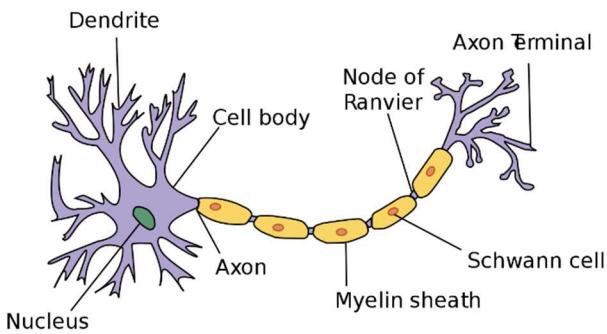
Merkezi sinir sistemi beyin ve omurilikten oluşmaktadır. Beyinden çıkan nöral iletiler omurilikten geçerek tüm vücuda yayılmaktadır. Beyin ve omuriliğin dışındaki nöral ağa çevresel sinir sistemi denilmektedir. Kaslarımız nöronlar tarafından harekete geçirilirler. Fakat bu emir çoğu kez beyin tarafından verilmektedir. Kasları harekete getiren nöronlara motor nöron denilmektedir. Merkezi sinir sistemi de kendi içerisinde "somatik sinir sistemi" ve "otonom sinir sistemi" olmak üzere ikiye ayrılır. Somatik sinir sistemi bilinçli eylemlere yol açan kısım için otonom sinir sistemi ise bilinçsiz eylemlere yol açan kısım için kullanılmaktadır. Kalp kaslarının kasılmaları, solunum gibi faaliyetler bilinçli olarak gerçekleşmezler. Otomatik biçimde (otonom) gerçekleşirler.

Duyumlar (sensation) bu konuda özelleşmiş nöronlar vasıtasiyla gerçekleşmektedir. Fiziksel uyaranlar bu nöronları uyarır. Bu nöronlar bu iletiyi kimyasal düzeye dönüştürürler (transduction) sonra duruma göre omuriliğe ve oradan da beynin ilgili bölümüne iletirler. Fiziksel uyaranları alan özelleşmiş bu nöronlara "duyusal nöronlar (sensory neurons)" denilmektedir.

Felç olgusunun çeşitli nedenleri vardır. En çok karşılaşılan nedenleri beyindeki lezyonlar (beynin kanaması sonucunda ya da tümörel biçimde) ya da kazalardır. Diğer nedenler arasında omurilik zedelenmeleri, fitikler gibi sorunlar bulunmaktadır.

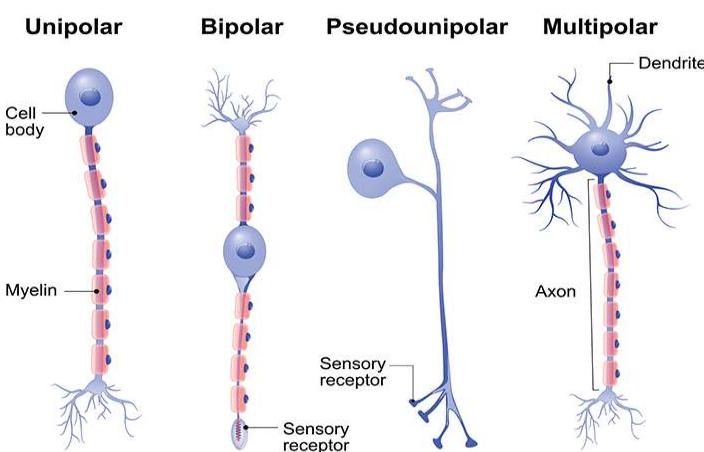
Sinir sisteminin en önemli hücreleri nöronlardır. Nöronların en fazla bulunduğu yer beyindir. Beyinde 100 milyar civarında nöron vardır. Nöronların dışında sinir sisteminde gliya gibi nöromodülör hücreler de bulunmaktadır.

Bir nöron hücresi tipik olarak aşağıdaki gibidir:



Alıntı Notu: GörSEL <https://simple.wikipedia.org/wiki/Neuron> adresinden alınmıştır.

Nöronların da birkaç çeşidi vardır. Yukarıdaki şekilde görüldüğü tipteki nöronlara "multipolar nöronlar" denilmektedir. Bu nöronlar sinir sisteminde en fazla bulunan nöronlardır. Aşağıda diğer bazı nöron çeşitlerini görüyorsunuz:

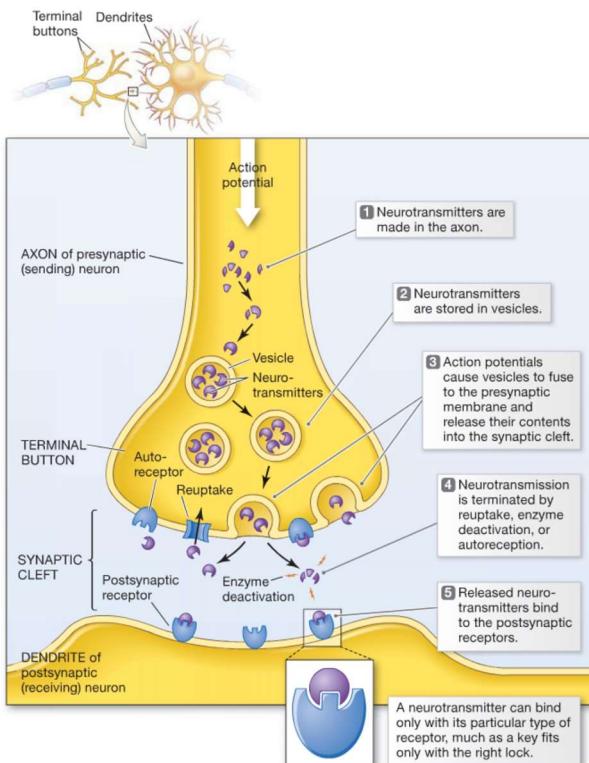


Alıntı Notu: GörSEL <https://qbi.uq.edu.au/brain/brain-anatomy/types-neurons> adresinden alınmıştır.

Multipolar nöronlar bir çekirdekten, nöral iletiyi alan dendrit'lерden ve nöral iletiyi ileten aksonlardan oluşurlar. Akson denilen uzun dalın üzerinde bazı nöronlarda miyelin kılıfı (myelin sheet) bulunmaktadır. Miyelin kılıfının en önemli işlevi nöral iletiyi hızlandırmasıdır. Miyelinli nöronların ileti hızı yaklaşık 80-120 m/sn, miyelinsiz nöronların ileti hızı ise yaklaşık 0.5-2 m/sn civarındadır. Beyinde hem miyelinli hem de miyelinsiz nöronlar bulunur. Miyelenli nöronlar beyaz renkte miyelinsiz olanlar ise gri renkte görüntü verdiklerinden beyinde miyelinli nöronların bulunduğu bölgelere "beyaz madde (white matter)", miyelinsiz nöronların bulunduğu bölgelere ise "gri madde (gray matter)" denilmektedir.

Pekiyi nöral ileti kimyasal düzeyde nasıl gerçekleşmektedir? İşte bir nöronun aksonunun ucunda düğmecikler (terminal buttons) bulunmaktadır. Nöron ateşlendiğinde bu düğmeciklerden "nörotransmitter" denilen moleküller zerk edilir. Nöral ileti büyük ölçüde bu nörotransmitterler yoluyla yapılmaktadır.

Akson uçları duruma göre yüzlerce ya da binlerce başka nörona bağlı olabilmektedir. İleti genellikle (ama her zaman değil) bir nöronun aksonu ile karşı taraftaki nöronun dendriti arasında gerçekleşir. Nöral ietinin gerçekleştiği bölgeye sinaps denilmektedir. Bir sinaps tipik olarak şöyledir:



Alıntı Notu: Görüel <https://in.pinterest.com/pin/722546333950674913/> adresinden alınmıştır.

Yukarıda de belirtildiği gibi aksonların ucunda düğmecikler (axon terminal buttons) bulunmaktadır. Dendritlerde de ismine reseptör denilen küçük yarıklar (synaptic cleft) vardır. Akson uçlarındaki düğmeciklerden ismine nörotransmitter denilen kimyasallar zerk edilir. Zerk edilen bu nörotransmitterler dendritlerdeki reseptörler tarafından alınmaktadır. Ancak her türlü reseptör her türlü nörotransmitter'i alamamaktadır. Her nörotransmitter'i alan farklı reseptörler vardır.

Nörotransmitterler çeşitli alt sınıflara ayrılmaktadır. En önemli nörotransmitterler şunlardır:

Amino Asit Grubundan olanlar: Glutamat, GABA, Glisin

Monoamin Grubundan Olanlar: Serotonin, Histamin, Dopamin, Adrenalin, Noradrenalin. Dopamin, Adrenalin ve Noradrenaline "katakolaminler" de denilmektedir.

Peptit Grubundan Olanlar: Endorfin

Diğerleri: Asetilkolin, ...

Örneğin psikoaktif maddeler ve psikotrop ilaçlar bu nörotransmitterleri artırıp azaltabilmektedir. Sinapslarda nörotransmitterleri artırma etkisi yaratan maddelere "agonist", azalma etkisi yaratan maddelere ise "antagonist" denilmektedir. Örneğin "dopamin antagonisti demek sinapslardaki dopamin seviyesini düşüren madde" demektir. Agonist ya da antagonist etki çeşitli biçimlerde oluşturulabilmektedir. Örneğin:

- Nörotransmitter üretimi artırılarak ya da azaltılarak
- Reseptörler bloke edilerek.
- Reseptörlerin etkinliği artırılarak (yani daha çok nörotransmitter olmasını sağlayarak).

Axon uçları nörotransmitterleri zerk ettikten sonra kullanılmayanları geri almaktadır. Buna "geri alım (reuptake)" işlemi denilmektedir. İşte bazı maddeler bu geri alımı engelleyerek agonist bir etki oluşturabilmektedir. Yukarıda da belirtildiği gibi psikiyatride kullanılan psikotrop ilaçlar büyük ölçüde sinapslardaki nörotransmitterler üzerinde etkili olmaktadır. Örneğin:

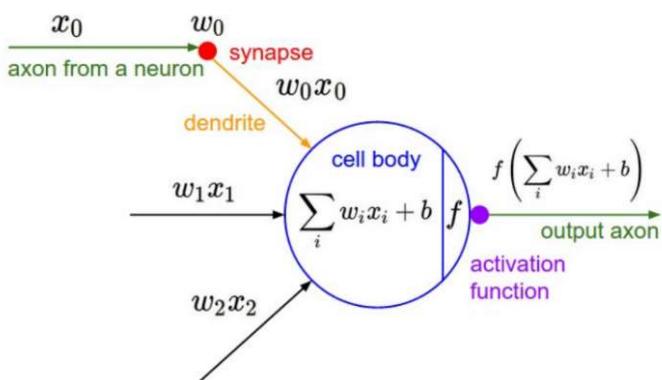
- SSRI türevi antidepresanlar serotonin geri alımını engelleyerek sinapslardaki serotonin miktarını artırmaktadır. Depresyonla sinapslardaki serotonin eksikliği arasında ilişki olduğu düşünülmektedir.

- Bazı trisklik antidepressanlar ve SNRI türevi antidepressanlar sinapslardaki noradrenalin miktarını artırmaktadır.
- Şizofreni ve diğer psikotik bozuklukların tedavilerinde kullanılan antipsikotikler genellikle dopaminerjik sistemle ilgili etki gösterirler. Bazı psikotik bozuklukların nedenleri beynin bazı bölgelerindeki dopaminerjik aktivitenin fazla olmasıyla açıklanmaya çalışılmaktadır.
- Sakinleştirici (sedatif) olarak kullanılan ilaçların çoğu (örneğin benzodiazepinler) GABA isimli nörotransmitteri artırmaktadır. GABA nöral ileti üzerinde inhibisyon etkisi yapan önemli bir nörotransmitter'dir.

Psikotrop ilaçlar deneyel yolla (hatta bazıları tesadüflerle) bulunmuştur. Maalesef neredeyse hiçbir psikotrop ilaçın çalışma mekanizması ayrıntılarıyla ve kesin düzeyde bilinmemektedir.

Yapay Nöron Modeli

Yapay sinir ağlarındaki nöral iletişim oldukça basitleştirilerek matematiksel bir algoritmik yönteme dönüştürülmüştür. İşte yapay sinir ağları temelde bu basitleştirilmiş matematiksel nöron modeline dayanmaktadır. Nasıl gerçek bir nörona birtakım girişler (dendritlerdeki reseptörler) ve çıkışlar (akon'dan zerk edilen nörotransmitterler) varsa matematiksel nöron modelinde de nörona çeşitli girişler ve bir çıkış söz konusudur. Örnek bir yapay nöron aşağıdaki şekilde temsil edilebilir. Bu tek nöronluk modele aynı zamanda "perceptron" da denilmektedir.



Bu örnek nöron modelinde x 'ler (x_0, x_1, x_2) gözlenen birtakım değişkenleri temsil etmektedir. Başka bir deyişle buradaki x 'ler tipik bir istatistik veri tablosundaki (yani data frame'lerdeki) sütunları temsil etmektedir. Örneğin bir apartman dairesinin o andaki piyasa değerini tahmin etmeye çalışan bir nöral ağ tasarlayacak olalım. Bu ağın girdileri (yani x değerleri) neler olabilir? Bir dairenin fiyatı neler bağlıdır? Burada kabaca şu unsurları sayabiliyoruz:

- Apartmanın yaşı
- Dairenin metrekare büyüklüğü
- Dairenin şehir merkezine uzaklığı
- Dairenin binanın kaçinci katında olduğu
- Dairenin oda sayısı

Ya da örneğin 10 tane biyomedikal tetkik sonucuna göre kişilerin belli bir hastalığa sahip olup olmadığını anlamaya çalışan bir yapay sinir ağ modeli kurmak isteyelim. Bu modeldeki x 'ler bu 10 biyomedikal tetkik değerleri olacaktır. Bu yapay sinir ağında çıktılar girdilere bağlandıktan sonra en hihayet tek bir çıkış elde edilecektir. Bu çıkış da kişinin o hastalığa sahip olduğunu verecektir.

Yukarıdaki nöron modelinde bu girdi değişkenlerinin birtakım w katsayılarıla çarpılıp toplandığı görülmektedir. Sonra da bu toplam bir fonksiyona sokulup bir çıktı değeri elde edilmektedir. Peki buradaki w değerleri ne anlam ifade etmektedir? w değerleri aslında eğitimli ağlarda nihai olarak elde edilmiş bir daha değiştirilmeyecek ağırlık değerleridir.

Zaten yapay sinir ağının eğitilmesi demek aslında girildilere göre uygun sonucu verebilecek w ağırlık değerlerinin tespit edilmesi sürecidir. Eğitimli yapay ağlarında daha önceki verilere dayanılarak girdilerle çıktılar arasında bir ilişki kurulmaya çalışılır. Bu ilişki aslında w katsayılarının uygun biçimde belirlenmesiyle kurulmaktadır. Örneğin ev fiyatını belirleme probleminde biz yapay sinir ağına evin özelliklerini girdi olarak veririz. Yapay sinir ağı da evin fiyatını bize çıktı olarak verir. Ancak yapay sinir ağının bunu yapabilmesi için gerçek birtakım ev özellikleri ve fiyatlarıyla eğitilmesi gerekmektedir. İşte bu eğitim sırasında aslında yapay sinir ağındaki w katsayıları uygun biçimde ayarlanmaktadır. Eğitim bittikten sonra artık biz ağa yeni bir ev özelliklerini verdığımızda bu w katsayıları ve aktivasyon fonksiyonları devreye girerek bize o evin fiyatını verecektir.

Biz yukarıda tek nöron üzerinde açıklamalar yaptık. Halbuki insanın sinir sisteminde olduğu gibi yapay sinir ağlarında da pek çok nöronun çıktıları başka nöronların girdilerine bağlanabilmektedir. İleride bu bağlantı modelleri hakkında bilgiler verilecektir.

Yapay Nöronun Python'da Bir Sınıfla Temsil Edilmesi

Yapay bir nöron basit bir sınıfta temsil edilebilir. Örneğin:

```
import numpy as np

class Neuron:
    def __init__(self, weights, activation, bias = 0):
        self.weights = weights
        self.activation = activation
        self.bias = bias

    def output(self, x):
        total = np.dot(self.weights, x)
        return self.activation(total + self.bias)

    def sigmoid(x):
        return 1 / (1 + np.exp(-x))

n = Neuron([0.2, 0.3, 0.4], sigmoid)
result = n.output([2, 1, 7])
print(result)
```

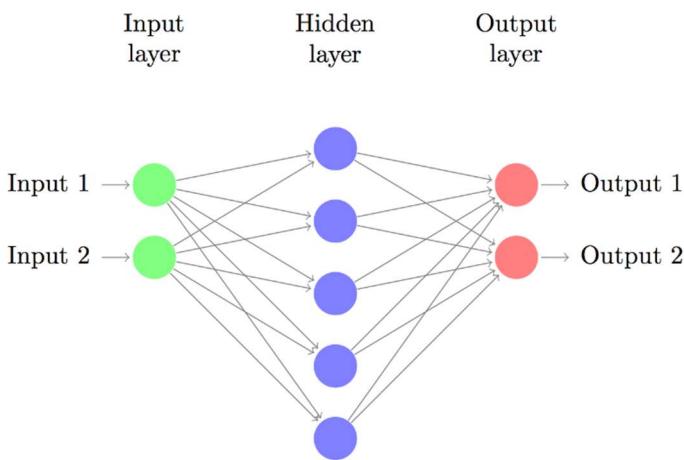
Burada Neuron sınıfı bizden nesne yaratıldığında ağırlık değerlerini, aktivasyon fonksiyonunu ve bias değerini istemiştir. Sınıfın output fonksiyonu da nöral işlemi yaparak bize çıktı değerini vermektedir.

Yapay Sinir Ağlarında Katmanlar

Bir yapay sinir ağında üç katman söz konusudur:

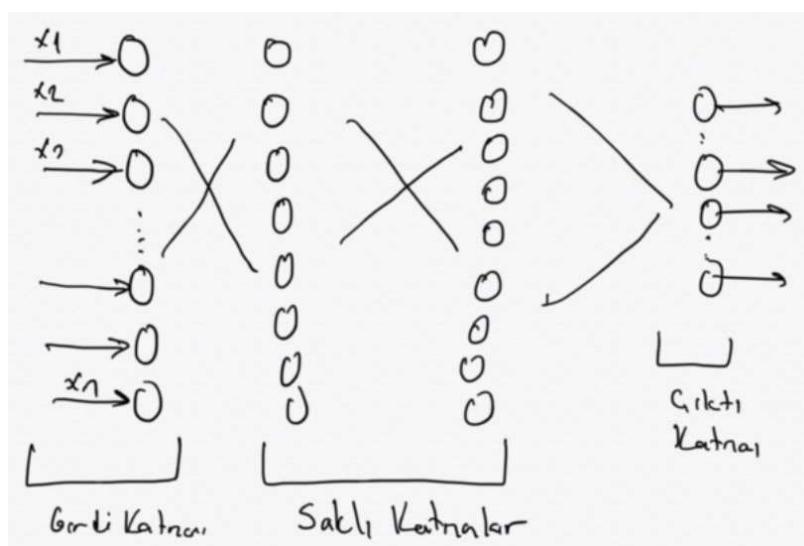
- 1) Girdi Katmanı (Input Layer / Visible Layer)
- 2) Saklı Katman (Hidden Layer)
- 3) Çıkış Katmanı (Output Layer)

Bir yapay sinir ağında katmanlar ve katmanlardaki nöronların oluşturduğu yapıya "ağ mimarisi" denilmektedir.



Girdi katmanı dış dünyadan girdi değerlerini (yani x değerlerini) alan katmandır. Girdi katmanındaki nöronlar içerisinde bir işlem yapılmamaktadır. Her ne kadar şekillerde girdi katmanı da her bir x giriş için bir nöronla temsil ediliyorsa da aslında gerçek bir nöron işlevi yapmamaktadır. Yani girdi katmanındaki nöronların bir giriş varıdır. Bu nöronların çıkış değeri de giriş değerinin aynısıdır. Örneğin bir kişinin diyabetli olup olmadığını anlayabilmek için 10 farklı biyomedikal tetkik sonuçlarını girdi olarak kullanmak isteyelim. Bu durumda ağın girdi katmanı 10 nöronundan oluşacaktır. Bu 10 nöronun girdisi tamamen çıktı ile aynı olacaktır.

Basit problemler yalnızca girdi ve çıktı katmanı olan bir ağla çözülebilir. Ancak model karmaşıklaştıkça ara katman görevini yapan saklı katmanların kullanılması gerekmektedir. Saklı katmanlar düzey olarak bir tane, iki tane ya da ikiden çok olabilir. Örneğin:



Burada iki saklı katman bulunmaktadır. Duruma göre veri bilimcisi tarafından saklı katmanlarının sayısı artırılıp azaltılabilmektedir. Saklı katmanlardaki nöron sayıları girdi katmanındaki nöron sayısı ile aynı olmak zorunda değildir. Genellikle eğer girdi katmanındaki nöronların sayıları görelî olarak azsa saklı katmanlardaki nöronların sayıları fazlalaştırılır. Eğer girdi katmanında görelî olarak çok fazla nöron varsa saklı katmanlardaki nöronların sayıları işlem yüklerini azaltmak için düşürülebilmektedir. Genel olarak saklı katmanların sayısı ikiden büyükse bu tür yapay sinir ağlarına derin yapay sinir ağları ya da özetle derin öğrenme (deep learning) ağları denilmektedir. Yani derin ağlarla normal sinir ağları arasındaki fark saklı katmanların sayısı ile ilgilidir.

Çıktı katmanı sinir ağından elde edilecek bilgiyi dış dünyaya veren katmandır. Tabii çıktı katmanındaki nöronlarda da saklı katmanlarda olduğu gibi işlemler yapılmaktadır. Ancak çıktı katmanı artık değerlerin elde edildiği son katmandır.

Yani biz girdileri girdi katmanına veririz sonucu çıktı katmanından alırız. Çıktı katmanın kaç nöron dan oluşacağı çıktıının ne olduğu ile ilgilidir.

Şimdi katmanlardaki olası nöron sayıları hakkında bazı şeyler söyleyelim. Girdi katmanındaki nöron sayısı kestirimde kullanılacak özelliklerin (features) sayısı kadar olmalıdır. Çıktı katmanındaki nöronların sayısı da elde edilecek kestirim sonucuyla ilgilidir. Örneğin ağımız var/yok gibi, olumlu/olumsuz gibi, hasta/sağlıklı ikili bir kestirimde bulunacaksa (bu tür modellere istatistikte lojistik regresyon denilmektedir) çıktı katmanı tek bir nöron dan oluşur. Örneğin ağımız bir resimdeki rakamın kaç olduğunu kestirmeye çalışıyorsa bu durumda çıktı katmanında 10 tane nöron bulunur. Yine örneğin ağımız bir evin fiyatını tahmin etmeye çalışıyorsa çıktı katmanı evin fiyatını veren tek bir nöron dan oluşacaktır. O halde bilmemişimiz iki durum vardır: Ağıımızda kaç tane saklı katman ve her saklı katmanda kaç tane nöron bulunmalıdır? İşte bu soruların yanıtları deneyimle ve üzerinde çalışılan problemin nitelegine göre veri bilimcisi tarafından doğru biçimde verilmeye çalışılmaktadır. Ancak yine de bu konuda önerilen birkaç genel yönerge vardır. Saklı katmanların sayısı için şu pratik tavsiyelerde bulunulabilir. (<https://www.heatonresearch.com/2017/06/01/hidden-layers.html>)

- Eğer problem doğrusal olarak ayrılabilir (linear separable) ise saklı katman kullanmaya gerek yoktur. Doğrusal olarak ayrılabilir problem hakkında kısa teorik bilgi ileride verilecektir.
- Bir saklı katman pek çok sorunun çözümü için yeterlidir. İki saklı katman çok büyük ölçüde pek çok problemin çözümü için yeterli olmaktadır.
- İkiden fazla saklı katmanı olan derin modeller karmaşık problemlerin çözümü için kullanılmaktadır. Bunlara örnekler ileride verilecektir.

Bir saklı katmandaki nöron sayıları için üstünüğü pratik tavsiyeler de şunlar olabilir:

- Saklı katmandaki nöronların sayısı girdi katmanındaki nöronların sayısının $2/3$ ile çıktı katmanındaki nöronların sayısının toplamı kadar olabilir. Örneğin girdi katmanındaki nöron sayısı 5, çıktı katmanındaki 1 olsun. Bu durumda saklı katmandaki nöron sayısı 5 olabilir.
- Saklı katmandaki nöronların sayısı girdi katmanındaki nöron sayısının iki katından fazla olmamalıdır.

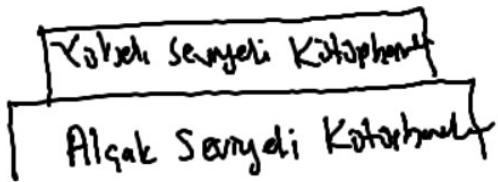
Tabii aslında saklı katmanlardaki nöron sayılarına ilişkin bu iki sezgisel yargıya bazı durumlarda uyulmamaktadır. Genellikle saklı katmanlardaki nöron sayıları deneyimle, deneme yanlışlığını yöntemleriyle belirlenmektedir. Şüphesiz bir fayda sağlanmadığı halde çok saklı katman ya da saklı katmanlarda çok nöron bulundurmasının bir anlamı yoktur. Ağ ne kadar sade olursa hem işlem yükü bakımından hem de verilerin saklanması ve iletilmesi bakımından o kadar avantaj sağlanır.

Yapay Sinir Ağlarında ve Derin Öğrenme Ağlarında Yaygın Kullanılan Kütüphaneler ve Framework'ler

Bir yapay sinir ağı mimarisinin sıfırdan programını yazmak çok zor olmasa da zaman alıcı bir süreçtir. Üstelik programcılar tarafından kendi gereksinimleri doğrultusunda yazılan bu tür kodların pek çoğu genel değildir ve genişletilememektedir. Bu konudaki diğer bir sorun ise yapay sinir ağlarındaki işlem yüküyle ilgilidir. Yapay sinir ağlarının eğitilmesi çok yoğun işlemleri gerektirebilmektedir. Bu işlemleri bir framework kullanmadan yapmak isteyen programcı kendisini paralel programlama gibi görelî karmaşık konular içerisinde bulabilmektedir. Bu da programcının toplam iş yükünü artırmaktadır.

İşte her konuda olduğu gibi yapay sinir ağlarında da işlemleri kolaylaştırmak için birtakım kütüphanelerden ve framework'lerden faydalılmaktadır. Bu tür kütüphanelerin ve framework'lerin sayısı oldukça fazla olmakla birlikte bazları çokça tercih edilmektedir. Bu tür kütüphaneleri ve framework'leri kabaca "aşağı seviyeli olanlar" ve yüksek

"yüksek seviyeli olanlar" biçiminde ikiye ayıralım. Tabii aslında yüksek seviyeli kütüphaneler ve framework'ler kendi içlerinde aşağı seviyeli kütüphanelerden faydalananabilmektedir.



Bazı kütüphaneler ve framework'ler yalnızca bazı dillerden kullanılabılırken bazıları farklı programlama dillerinden kullanılabilmektedir. Kütüphane ve framework tercihlerini bu durum da etkilemektedir.

scikit-learn isimli kütüphane Python'da yazılmıştır ve makine öğrenmesi için kullanılan aşağı seviyeli genel bir kütüphanedir. Pek çok kütüphane kendi içerisinde scikit-learn kütüphanesini de kullanmaktadır. scikit-learn açık kaynak kodlu bir kütüphanedir, numpy ve scipy kullanılarak yazılmıştır. Ancak scikit-learn yapay sinir ağlarına yönelik tasarlanmamıştır. Yani scikit-learn içerisinde yapay sinir ağlarını oluşturacak modüller yoktur.

PyTorch Python ve C++'ta yazılmış olan bir yapay sinir ağının derin öğrenme ağının kütüphanesidir. Proje Facebook tarafından yaşama geçirilmiştir. Bu kütüphane veri bilimcileri arasında belli yoğunluklarda kullanılmaktadır. PyTorch da numpy üzerine oturtulmuştur.

Theano aslında Python programları için yazılmış bir nümerik analiz kütüphanesidir. Ancak yapay sinir ağlarında ve derin öğrenmede aşağı seviyeli bir kütüphane olarak Theano'dan faydalılmaktadır.

TensorFlow Google tarafından geliştirilmiş olan aşağı seviyeli nümerik analiz kütüphanesidir. Makine öğrenmesi, yapay sinir ağları ve derin öğrenme ağları uygulamalarında en çok tercih edilen aşağı seviyeli kütüphane durumundadır. TensorFlow Python dahil olmak üzere pek çok programlama dilinden kullanılabilir. Bu kütüphane de Python ve C++ kullanılarak yazılmıştır.

MxNet ise başka bir aşağı seviyeli yapay sinir ağları ve derin öğrenme kütüphanesidir. Apache grubu tarafından açık kaynak kodlu olarak yazılmıştır.

Yüksek seviyeli yapay sinir ağları ve derin öğrenme kütüphanelerinin açık ara en başında Keras gelmektedir. Keras Python'da yazılmıştır ve oldukça yüksek seviyeli bir kütüphanedir. Ancak Keras arka planda birkaç alçak seviyeli kütüphanesi "backend" olarak kullanılabilmektedir. (Default durumda Keras Google'in TensorFlow kütüphanesini kullanmaktadır.) Kursumuzun bu bölümünde yapay sinir ağları ve derin öğrenme uygulamalarında basitliğinden dolayı Keras kullanılacaktır. Keras kütüphanesinin dokümantasyonu <https://keras.io/> sitesinde bulunmaktadır. Keras açık kaynak kodlu bir projedir ve Google Tensorflow ekibi tarafından da desteklenmektedir.

Veri Tablolarının Kullanıma Hazır Hale Getirilmesi

İstatistikte toplanan bilgiler ilk önce veri tabloları biçiminde düzenlenirler. Bir veri tablosu sütunlardan ve satırlardan oluşmaktadır. Bu bakımdan veri tabloları veritabanlarındaki tablolara da benzemektedir. Veri tablolarındaki sütunlara "özellikler (features)" denilmektedir. Biz kursumuzda veri tablolarındaki msütunlara bazen "özellik" bazen de "sütun" diyeceğiz. Örnek bir veri tablosu şöyle olabilir:

Medeni Durum	Yaş	Eğitim Durumu	Maaş
Evl.	37	Lise	3850
Bekar	40	Üniverzite	7360
Evi	28	Lise	2700
...

Burada Medeni Durum, Yaş, Eğitim Durumu ve Maaş tablonun özellikleridir. Satırlar da kayıtları oluşturmaktadır. Bir veri tablosunda sütunlar farklı ölçek türlerine ilişkin olabilmektedir. Örneğin yukarıdaki tabloda Medeni Durum Kategorik (Nominal), Eğitim Durumu Sırasal (Ordinal) ölçeklere ilişkindir. Ancak Yaş ve Maaş sütunları (özellikleri) Oransal (Ratio) ölçüye ilişkindir.

Veri tabloları çeşitli formatlarda bulunabilmektedir. Makine öğrenmesinde en çok karşılaşılan format CSV'dir. CSV (Comma Separated Values) dosyası satırlardan oluşmaktadır. Satırlardaki elemanlar (yani sütun bilgileri) tipik olarak ',' karakteri ile birbirlerinden ayrılmaktadır. (Aslında CSV dosyalarında ayıraçlar bazen ',' dışındaki karakterlerden de oluşturulabilmektedir.) CSV dosyalarında isteğe bağlı olarak sütunların ne anlam ifade ettiğini belirten başlık kısımları da bulunabilmektedir. Bu başlık kısımları bizim için bazen gerekli bazen de gereksiz olabilir. Örneğin bu başlıklar eğer CSV dosyası pandas.read_csv fonksiyonuyla okunuyorsa DataFrame içerisindeki sütun isimleri haline dönüştürülmektedir. Biz kursumuzda veri tablolarının genel olarak CSV formatında olduğunu varsayıcağız.

Alıntı Notu: Aşağıdaki gruplandırma "Data Preparation for Machine Learning - Jason Brownlee" isimli kitaptan alınmıştır ve bölüm içerisinde bu kitaptan faydalанılmıştır. Okuyucunun bu kitabı baştan sona gözden geçirmesini tavsiye ediyoruz.

Verilerin kullanıma hazır hale getirilmesi süreci aslında ayrıntılı bir konudur. Bu nedenle biz kursumuzda önce genel bir açıklama yapıp sonra gerekli yerlerde gerekli yöntemleri ele alacağız. Genel olarak bir veri tablosunun kullanıma hazır hale getirilmesi için beş yöntem grubu kullanılmaktadır. Şimdi bu yöntem grupları hakkında temel bilgileri edinelim.

1) Verilerin Temizlenmesi (Data Cleaning): Veriler içerisinde geçersiz olan değerler söz konusu olabilir. Geçersiz veriler bazen mevcut olmayan (nan/None) veriler biçiminde, Bazen bozulmuş veriler biçiminde, bazen de yinelenen (mükerrer) veriler biçiminde karşımıza çıkabilmektedir. Veri bilimcisinin bu verileri dikkate alıp bunları ortadan kaldırırmaya yönelik teknikleri uygulaması gereklidir. Örneğin bu tür durumlarda geçersiz verilerin bulunduğu satırlar ya da sütunlar tablodan tümden atılabilir ya da geçersiz değerler yerine başka değerler (tipik olarak ortalama değerler ya da mod değerleri) yerleştirilebilir. Bazen aşırı ucta bazı değerler bozucu etkiler de yaratılmaktadır. Bu aşırı uçtaki değerlerden de kurtulmak gerekebilir. Bazı veri tablolarında ise birbirile yüksek korelasyona sahip olan sütunlar (özellikler) da bulunabilmektedir. Bize ek bir bilgi vermeyen bu tür sütunların bazlarının atılması işlenecek verinin miktarını düşürmektedir.

2) Özellik Seçimi (Feature Selection): Özellik seçimi veri tablosundaki sütunların gerekli olanlarının alınıp gereksiz olanlarının atılması ile ilgili bir hazırlık etkinliğidir. Örneğin veri tablosunun bir sütununda kişinin ismi olabilir ve bu ismin kestirim sürecinde hiçbir etkisi olmayıabilir. Bu durumda bu sütunun atılması gerekebilir. Bazen çok sayıda sütunun nispeten önemsiz olanlarının atılması da uygun olabilmektedir. Uygun sütunların seçilmesi süreci sütunun türlerine ve hedefe bağlı olarak da değişebilmektedir.

3) Verilerin Dönüşürlmesi (Data Transformation): Verilerin dönüşürlmesi verilerin türlerini ya da dağılımını değiştirmek için kullanılan yöntemlerdir. Makine öğrenmesinde sıralı ve kategorik veriler doğrudan kullanılamamaktadır. Bunların algoritmala sokulmadan önce sayısal biçimde dönüştürülmesi gereklidir. İşte sıralı ve kategorik verilerin sayısal biçimde dönüştürülmesi en çok uygulanan veri dönüştürmesi yöntemlerindendir. Veri dönüştürmesi sırasında bazen bunun tersi de yapılmaktadır. Örneğin sayısal verilerin sıralı biçimde dönüştürülmesi de gerekebilmektedir. Bu işlem "ayırık hale getirme (discretization transform)" denilmektedir. Kategorik verilerin sayısal biçimde dönüştürülmesinde en sık uygulanan yöntemlerden biri "one hot encoding" denilen yöntemdir. Bu

yöntemde kategorik veri birden fazla sütunu olan ikili (binary) veri haline dönüştürülmektedir. Bazen veri tablosunun sütunları arasında skala farklılıklar olabilmektedir. Bu skala farklılıklar pek çok algoritmada bozucu etki yaratır. Bunları ortadan kaldırmak için "normalizasyon" ve "standardizasyon" işlemleri yapılabilmektedir.

4) Özellik Mühendisliği (Feature Engineering): Özellik mühendisliği var olan sütunlardan yeni sütunlar oluşturma sürecine ilişkin yöntemlere denilmektedir. Örneğin bazen sütunlara bool bir sütun eklenebilir. Bazen sütunlar üzerinde bazı işlemlerin sonucu olarak yeni bir sütun eklenebilir.

5) Boyut İndirgemesi (Dimensionality Reduction): Veri tablosundaki sütunlar aslında çok boyutlu uzaydaki boyutlar olarak düşünülebilirler. Örneğin iki sütunlu verilerdeki satırlar iki boyutlu bir düzlemede nokta belirtirler. Boyut indirgemesi de aslında çok sayıda sütunu temsil edebilen daha az sayıda sütun oluşturulması sürecidir. Boyut indirgemesi için "temel bileşenler analizi (principal components analysis)", "tekil değer ayrıştırması (singular value decomposition)", "doğrusal ayırm analizi (linear discriminant analysis)" gibi yöntemler kullanılmaktadır.

Veri Tablolarının Gereksiz Sütunlardan Arındırılması

Verilerin kullanıma hazır hale getirilmesi sürecinde veri tablolarının gereksiz sütunlardan arındırılması en çok kullanılan "özellik seçimi (feature selection)" yöntemlerinden biridir. Veri tablolarının gereksiz sütunlardan arındırılması birkaç biçimde yapılmaktadır. Eğer bir CSV dosyası söz konusu ise (makine öğrenmesi bağlamında genellikle veriler CSV dosyalarından okunmaktadır) gereksiz sütunlar numpy.loadtxt fonksiyonunda işin başında atılabilir. Örneğin aşağıdaki gibi bir CSV dosyası olsun:

```
Adı Soyadı,Boy,Kilo,Doğum Yeri  
Ali Bulut,182,89,Eskişehir  
Erol Öner,187,82,İzmir  
Ayşe Tan,172,58,Urfा  
Rasim Taşcan,168,92,Samsun
```

Dosyanın isminin "test.csv" olduğunu varsayıyalım. Şimdi bu veri tablosunun "Adı Soyadı" ve "Doğum Yeri" sütunlarını gereksiz olduğu gerekçesiyle atmak isteyebiliriz. numpy.loadtxt fonksiyonun usecols parameteresinin dosyadan elde edilecek satırları belirlemek için kullanıldığını anımsayınız. Bu tabloda Boy 1'inci sütunu, Kilo ise 2'inci sütunu belirtmektedir. Okuma işlemi şöyle yapılabilir:

```
import numpy as np  
  
data = np.loadtxt('test.csv', delimiter=',', encoding='utf-8', skiprows=1, usecols=(1, 2))  
print(data)
```

İşlemden şöyle bir çıktı elde ederiz:

```
[[182.  89.]  
 [187.  82.]  
 [172.  58.]  
 [168.  92.]]
```

Tabii tablonun hepsini okuyup söz konusu ikiş sütunu numpy.delete fonksiyonu ile axis=1 parametresini kullanarak da silebilirdik. Ancak bu işlem daha yorucu olurdu:

```
import numpy as np  
  
data = np.loadtxt('test.csv', delimiter=',', encoding='utf-8', skiprows=1, dtype='object')  
data = np.delete(data, [0, 3], axis=1).astype(np.float32)  
print(data)
```

Sayısal olmayan sütunları okurken dtype=object verildiğine dikkat ediniz. Bu durumda satırların hepsi string olarak elde edilmektedir. astype metodu string olan sütunların np.float32 türüne dönüştürülmesi için kullanılmıştır.

CSV dosyalarını okuyup gereksiz sütunları atmak için numpy.loadtxt fonksiyonu yerine pandas kütüphanesindeki read_csv fonksiyonu da kullanılabilir. pandas.read_csv fonksiyonunun numpy.loadtxt fonksiyonu ile benzer işlevselligi sahip olmakla birlikte daha yetenekli olduğunu söyleyebiliriz. Şimdi aynı işlemleri pandas.read_csv fonksiyonu ile yapalım:

```
import pandas as pd

data = pd.read_csv('test.csv', delimiter=',', encoding='utf-8', usecols=(1, 2))
print(data)
```

Kodun çalıştırılması sonucunda şöyle bir çıktı elde edilmiştir:

```
Boy Kilo
0 182 89
1 187 82
2 172 58
3 168 92
```

Burada CSV dosyasındaki başlık isimlerinin sütun isimleri haline getirildiğine dikkat ediniz. Tabii aslında pandas'ta biz yine tüm tabloyu okuduktan sonra da belli sütunları atabiliyoruz:

```
import pandas as pd

data = pd.read_csv('test.csv', delimiter=',', encoding='utf-8')
df = df[['Boy', 'Kilo']]
print(df)
```

Burada önce tüm tabloyu okuduktan sonra dilimleme ile belli sütunları aldığımıza dikkat ediniz. Elde edilen çıktı aynı olacaktır:

```
Boy Kilo
0 182 89
1 187 82
2 172 58
3 168 92
```

Şimdi de müşterilerin kredi kartı ödemelerini yapıp yapmadıkları ile ilgili bilgileri içeren "bank.csv" dosyasından bazı sütunları atalım. Bu dosyayı dosyayı <https://www.kaggle.com/janiobachmann/bank-marketing-dataset/data> adresinden indirebilirsiniz. Dosyanın çalışma dizininde bulunduğu varsayıcağız. Dosyanın genel görünümü aşağıdaki gibidir:

```
age,job,marital,education,default,balance,housing,loan,contact,day,month,duration,campaign,pdays,previous
,poutcome,deposit
59,admin.,married,secondary,no,2343,yes,no,unknown,5,may,1042,1,-1,0,unknown,yes
56,admin.,married,secondary,no,45,no,no,unknown,5,may,1467,1,-1,0,unknown,yes
41,technician,married,secondary,no,1270,yes,no,unknown,5,may,1389,1,-1,0,unknown,yes
55,services,married,secondary,no,2476,yes,no,unknown,5,may,579,1,-1,0,unknown,yes
54,admin.,married,tertiary,no,184,no,no,unknown,5,may,673,2,-1,0,unknown,yes
42,management,single,tertiary,no,0,yes,yes,unknown,5,may,562,2,-1,0,unknown,yes
56,management,married,tertiary,no
....
```

Burada örneğin 8'inci (contact) ve 15'inci (poutcome) sütunları atmak isteyelim. Bunun için pandas.read_csv numpy.loadtxt fonksiyonundan daha kolay bir seçenekir:

```
import pandas as pd

df = pd.read_csv('bank.csv')
df.drop(['contact', 'poutcome'], axis=1, inplace=True)
print(df)
```

Kodun çalıştırılması sonucunda oluşan çıktıyı inceleyiniz:

```
   age      job marital education ... campaign pdays previous deposit
0   59    admin. married secondary ...        1     -1       0     yes
1   56    admin. married secondary ...        1     -1       0     yes
2   41 technician married secondary ...        1     -1       0     yes
3   55   services married secondary ...        1     -1       0     yes
4   54    admin. married tertiary ...        2     -1       0     yes
...
11157  33 blue-collar single primary ...        1     -1       0      no
11158  39   services married secondary ...        4     -1       0      no
11159  32 technician single secondary ...        2     -1       0      no
11160  43 technician married secondary ...        2    172       5      no
11161  34 technician married secondary ...        1     -1       0      no
```

Kategorik (Nominal) ve Sıralı (Ordinal) Verilerin Sayısal Biçime Dönüşürtülmesi

Tipik bir veri tablosunda kişinin cinsiyeti, medeni durumu, yaşadığı ülke gibi kategorik (nominal) ve sıralı (ordinal) sütunlar bulunabilmektedir. Makine öğrenmesinde pek çok yöntem ve algoritma kategorik ve sıralı veriler üzerinde doğrudan işlem yapamamaktadır. Bu nedenle önce kategorik ve sıralı verilerin sayısal biçimde dönüştürülmeleri gereklidir. Ayrıca yapay sinir ağları bağlamında Keras kütüphanesi Pandas kütüphanesindeki DataFrame sınıfını kullanmaz. Yalnızca numpy kütüphanesinin ndarray veri yapısını kullanmaktadır. Yani başka bir deyişle bizim Keras'ı kullanabilelimiz için veri tablosundaki tüm sütunların sayısal biçimde ndarray haline dönüştürülmüş olması gereklidir. Bu nedenle kategorik ve sıralı verilerin sayısal biçimde dönüştürülmesi verileri hazır hale getirme sürecinde kullanılan en önemli veri dönüşümü (data transformation) yöntemlerinden biridir.

Kategorik (Nominal) Verilerin Sayısal Biçime Dönüşürtülmesi

Kategorik verilerde her bir kategori için 0'dan itibaren farklı bir tam sayı karşılık düşürülmesi yoluna gidilebilir. Tabii buradaki değerler arasında bir sıralama ilişkisi olmadığı için bu tam sayılar büyüklik küçüklük belirtmeyecek yalnızca farklı kategorileri belirtecektir. Bu tür kategorik alanların farklı tam sayılar biçiminde sayısal olarak ifade edilmesini kolaylaştırmak için numpy ya da pandas kütüphanelerinde hazır bir fonksiyon ya da metod bulunmaktadır. Bu işlem en kolay olarak sklearn (scikit-learn) kütüphanesindeki LabelEncoder sınıfı ile yapılabilir. Bu sınıf sklearn.preprocessing paketi içerisinde yer almaktadır. Dönüşüm için önce LabelEncoder sınıfı türünden bir nesne yaratılır. Örneğin:

```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
```

Sonra LabelEncoder sınıfının fit isimli metodu bizden kategorik ölçekteki değerleri alır. Sınıfın transform metodu da dönüştürülecek kategorik değerleri alarak 0'dan itibaren sayısal değerlere dönüştürmektedir. LabelEncoder sınıfının classes_ isimli örnek özniteliği ile fit işlemi sırasında verilen kategorik değerler elde edilebilir. Örneğin:

```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
```

```

le.fit(['mavi', 'kırmızı', 'yeşil', 'beyaz'])
data = ['mavi', 'kırmızı', 'mavi', 'yeşil', 'beyaz', 'yeşil', 'kırmızı']
result = le.transform(data)

print(data)
print(result)
print(le.classes_)

```

Kodun çıktısı şöyledir:

```

['mavi', 'kırmızı', 'mavi', 'yeşil', 'beyaz', 'yeşil', 'kırmızı']
[2 1 2 3 0 3 1]
['beyaz' 'kırmızı' 'mavi' 'yeşil']

```

Göründüğü gibi fit metodunu bizden kategorik ölçek değerlerini almış, transform ise bunları dönüştürmüştür. Biz bu örnekte metotlara Python listeleri verdik. Tabii uygulamada tipik olarak bu metotlara numpy dizileri verilmektedir. Aslında fit ve transform işlemi bir arada fit_transform metoduyla da yapılabilmektedir:

```

from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
data = ['mavi', 'kırmızı', 'mavi', 'yeşil', 'beyaz', 'yeşil', 'kırmızı']
result = le.fit_transform(data)

print(data)
print(result)
print(le.classes_)

```

Burada fit_transform doğrudan dönüştürülecek verileri bizden almıştır. Metot bu verileri önce fit metoduna sonra da transform metoduna sokmaktadır. Sınıfın fit metodunu aynı kategorik değerler birden fazla kez kullanılmışsa onları zaten dikkate almaz. Yukarıdaki koddan elde edilen çıktı öncekiyle aynı olacaktır:

```

['mavi', 'kırmızı', 'mavi', 'yeşil', 'beyaz', 'yeşil', 'kırmızı']
[2 1 2 3 0 3 1]
['beyaz' 'kırmızı' 'mavi' 'yeşil']

```

Burada önemli bir noktaya dikkatinizi çekmek istiyoruz. Bu sınıf ile yapılan dönüştürmede hangi kategorik değerin hangi sayıyla ifade edileceğini biz belirleyememekteyiz. fit metodunu verilen kategorik değerleri numpy.unique fonksiyonuna soktuktan sonra onlara 0'dan itibaren değerler karşılık düşürmektedir. numpy.unique fonksiyonun yinelenenleri attıktan sonra buna ek olarak sıraya dizme işlemi yaptığı da anımsayınız. Yani bu durumda aslında bizim fit ya da fit_transform metodlarına verdığımız kategoriler alfabetik olarak sıraya dizildikten sonra onlara numaralar atanmış olmaktadır. Gerçekten de yukarıdaki örnekte beyazın 0, kırmızının 1 olduğunu dikkat ediniz. Buradan LabelEncoder sınıfının eğer kategorik isimler sıralı değilse sıralı veriler için kullanılamayacağı sonucunu çıkarabiliriz.

LabelEncoder sınıfının fit ve fit_transform metodlarına aslında biz kategorik verileri yazışal biçimde vermek zorunda değiliz. Kategorik veriler zaten sayısal biçimde de bulunuyor olabilir. Örneğin kategorik verilerin şehirlerin plaka numaralarından oluştuğunu düşünelim:

```

from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
data = [34, 26, 37, 34, 6, 77, 10, 16, 35, 34, 6]
result = le.fit_transform(data)

print(data)
print(result)
print(le.classes_)

```

Kodun çıktısı şöyle olacaktır:

```
[34, 26, 37, 34, 6, 77, 10, 16, 35, 34, 6]
[4 3 6 4 0 7 1 2 5 4 0]
[ 6 10 16 26 34 35 37 77]
```

Şimdi aynı işlemi bir CSV dosyasından hareketle yapalım. Örneğimizdeki "test.csv" dosyası şöyle olsun:

```
Cinsiyet,Kilo,Boy,Şehir
Erkek,85,172,Eskişehir
Kadın,72,170,İzmir
Kadın,65,162,İstanbul
Erkek,92,183,İstanbul
Kadın,62,173,Ankara
Erkek,98,172,İzmir
```

Burada ilk ve son sütundaki kategorik verileri sayısal biçimde şöyle dönüştürebiliriz:

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder

df = pd.read_csv('test.csv')
print(df)

le = LabelEncoder()
df['Cinsiyet'] = le.fit_transform(df['Cinsiyet'])
print(le.classes_)
df['Şehir'] = le.fit_transform(df['Şehir'])
print(le.classes_)
print(df)
dataset = df.to_numpy()
print(dataset)
```

Kodun çalıştırılması sonucunda şöyle bir çıktı elde edilmiştir, inceleyiniz:

```
   Cinsiyet  Kilo  Boy      Şehir
0     Erkek    85  172  Eskişehir
1     Kadın    72  170      İzmir
2     Kadın    65  162  İstanbul
3     Erkek    92  183  İstanbul
4     Kadın    62  173    Ankara
5     Erkek    98  172  İzmir
['Erkek' 'Kadın']
['Ankara' 'Eskişehir' 'İstanbul' 'İzmir']
   Cinsiyet  Kilo  Boy  Şehir
0          0    85  172    1
1          1    72  170    3
2          1    65  162    2
3          0    92  183    2
4          1    62  173    0
5          0    98  172    3
[[ 0  85 172  1]
 [ 1  72 170  3]
 [ 1  65 162  2]
 [ 0  92 183  2]
 [ 1  62 173  0]]
```

```
[ 0  98 172   3]]
```

LabelEncoder sınıfının fit, transform ve fit_transform metotları hem Python listeleriyle, hem numpy dizileriyle hem de Pandas'in Series nesneleriyle çalışabilmektedir. transform ve fit_transform sonucu her zaman ndarray olarak vermektedir.

LabelEncoder sınıfının inverse_transform metodu tam tersi bir işlemi yapmaktadır. Yani bu metoda biz sayısal verileri verirsek o bize kategorik değerlerden oluşan bir numpy dizisi verir. Tabii inverse_transform metodunu kullanmadan önce bir fit işleminin yapılmış olması gereklidir.

sklearn.preprocessing modülünde LabelEncoder sınıfıyla benzer işlemi yapan OrdinalEncoder isimli bir sınıf da vardır. OrdinalEncoder aslında LabelEncoder sınıfının birden çok sütunu aynı anda alarak işlem yapan bir biçimde gibidir. OrdinalEncoder sınıfında oluşturulan kategoriler sınıfın categories_ örnek özniteliği ile bize bir numpy listesi olarak verilmektedir. Örneğin:

```
from sklearn.preprocessing import OrdinalEncoder

data = [['mavi', 'erkek'], ['kırmızı', 'kadın'], ['siyah', 'kadın'], ['mavi', 'kadın'],
        ['beyaz', 'kadın'], ['yeşil', 'kadın']]

oe = OrdinalEncoder()
result = oe.fit_transform(data)
print(result)
print(oe.categories_)
```

Şimdi önceki örnekteki "test.csv" dosyasını OrdinalEncoder sınıfıyla dönüştürelim:

```
import pandas as pd
from sklearn.preprocessing import OrdinalEncoder

df = pd.read_csv('test.csv')
print(df)

oe = OrdinalEncoder()
df[['Cinsiyet', 'Şehir']] = oe.fit_transform(df[['Cinsiyet', 'Şehir']])
print(df)
dataset = df.to_numpy()
print(dataset)
```

Kodun çıktısı şöyle olacaktır:

```
Cinsiyet Kilo Boy Şehir
0   Erkek    85  172 Esküsehir
1   Kadın    72  170 İzmır
2   Kadın    65  162 İstanbul
3   Erkek    92  183 İstanbul
4   Kadın    62  173 Ankara
5   Erkek    98  172 İzmır
Cinsiyet Kilo Boy Şehir
0      0.0   85  172   1.0
1      1.0   72  170   3.0
2      1.0   65  162   2.0
3      0.0   92  183   2.0
4      1.0   62  173   0.0
5      0.0   98  172   3.0
[[ 0.  85.  172.  1.]
 [ 1.  72.  170.  3.]]
```

```
[ 1. 65. 162. 2.]
[ 0. 92. 183. 2.]
[ 1. 62. 173. 0.]
[ 0. 98. 172. 3.]]
```

OrdinalEncoder sınıfının fit, transform ve fit_transform metotları iki boyutlu Python listeleriyle, numpy dizileriyle ya da Pandas'ın DataFrame nesneleriyle çalışabilmektedir. Bu metotlarla tek boyutlu bir liste, numpy dizisi ya da Series nesnesi ile kullanılamamaktadır. transform ve fit metotları bize her zaman sayısallaştırılmış verileri numpy dizisi olarak verdiği de dikkat ediniz.

OrdinalEncoder sınıfının da inverse_transform isimli bir metodu vardır. Bu metot çok boyutlu bir Python listesini, numpy dizisini ya da Pandas DataFrame nesnesini parametre olarak alıp kategorik değerlere ilişkin bir numpy dizisi vermektedir

OrdinalEncoder sınıfı da -ismi yanlış bir çağrıım uyandırsa da- aslında sıralı veriler için kullanılamaz. Çünkü burada yine LabelEncoder sınıfının yaptığı gibi numpy.unique işlemi uygulanmaktadır. Bu sınıf da alfabetik ya da sayısal olarak en küçük kategoriden başlayarak sayısal atamaları yapmaktadır.

Aslında kategorik veriler çoğu kez buradaki gibi sayısal biçimde dönüştürülmemektedir. Kategorik verilerin sayısal biçimde dönüştürülmesinde sıklıkla "one hot encoding" denilen teknik kullanılmaktadır. İzleyen bölümde "one hot encoding" denilen tekniği ele alacağız.

One Hot Encoding Dönüşürmesi

Kategorik verilerin 0'dan itibaren tam sayılar kullanılarak sayısal biçimde dönüştürülmesi pek çok algoritma için bir sorundur. Çünkü bu biçimdeki sayıların arasında büyükük küçükük ilişkisi vardır ve bu büyükük küçükük ilişkisi pek çok algoritmada yanlış öğrenmelere yol açabilmektedir. Örneğin biz LabelEncoder ya da OrdinalEncoder sınıflarıyla üç kategorik renk değerine Blue = 0, Green = 1, Yellow = 2 değerlerini atamış olalım. Buradaki Yellow > Green > Blue gibi bir ilişkinin ya da Blue + Green = Yellow gibi bir ilişkinin bizim için bir anlamı yoktur. Ancak pek çok algoritma bu üç renk böyle kodlandığında onların kategorik değil sıralı ölçüye sahip olduğunu sanmaktadır. İşte kategorik değerlerin sanki onlar sıralı ölçüye ilişkinmiş gibi değerlendirilmesini engellemek için "one hot encoding" denilen bir yöntem kullanılmaktadır. İngilizce "One hot" bir grupitten yalnızca birinin 1 olduğunu 0 olduğunu anlatmak için kullanılan bir deyimdir.

One hot encoding dönüşürmesinde kategorik veriler ikili sistemde kategori sayısı kadar sütunla ifade edilmektedir. Bu yöntemde n kategoriden birini belirten bir veri n tane sütunla ifade edilir. Verinin ilişkin olduğu kategoriye ilişkin sütun bilgisi 1 diğer sütun bilgileri 0 yapılır. Örneğin aşağıdaki gibi bir veri tablosu söz konusu olsun:

Tepki Süresi	Renk
1.2	Kırmızı
2.3	Mavi
0.7	Kırmızı
1.2	Yeşil
3.2	Mavi

Bu tabloda Renk "Kırmızı, Yeşil ve Mavi" olabilen üç kategoriden oluşan kategorik bir ölçüye ilişkindir. Bu tablodaki Renk sütununun "one hot encoding" tekniği ile sayısallaştırılması sonucunda şöyle bir tablo elde edilir:

Tepki Süresi	Kırmızı	Yeşil	Mavi
1.2	1	0	0
2.3	0	0	1

0.7	1	0	0
1.2	0	1	0
3.2	0	0	1

Burada Renk sütunun yerine onun kategori sayısı kadar sütun eklendiğine dikkat ediniz. Renk hangi renkse yalnızca o rengin sütun değeri 1, diğer renklerin sütun değerleri 0 yapılmıştır. Bu biçimdeki bir kodlamayla renkler arasında bir sıra ilişkisinin ortadan kaldırılmaya çalışıldığına dikkat ediniz. Eğer bizim kategorik verimiz 10 tane farklı kategoriden oluşmuş olsaydı biz de 10 kategori için 10 tane sütun oluşturacaktık.

Burada özel bir duruma dikkatinizi çekmek istiyoruz.. One hot encoding ikiden fazla sınıf belirten kategorik veriler için uygulanması gereken bir tekniktir. İki sınıflı kategorik veriler için one hot encoding uygulamaya hiç gerek yoktur. Örneğin veri tablosunun bir sütunu cinsiyet belirtiyor olsun. Cinsiyet ikili bir kategorik sınıf oluşturduğu için bu sütun için bizim one hot encoding uygulamamız gerekmeyecektir. Bu tür iki sınıflı kategorik alanlar tipik olarak 0 ve 1 biçiminde sırasallaştırılarak bırakılabilirler.

Şimdi de "one hot encoding" işleminin nasıl yapılacağını görelim. scikit-learn kütüphanesindeki sklearn.preprocessing modülünde "one hot encoding" işlemini yapan OneHotEncoder isimli hazır bir sınıf vardır. Ancak bu sınıf kategorik verilerin önce sayısal biçimde dönüştürülmüş olmasını ister. Bu nedenle biz bu sınıfı kullanmadan önce kategorik verileri LabelEncoder ya da OrdinalEncoder sınıfı ile sayısal biçimde dönüştürmemiz gerekmektedir. OneHotEncoder sınıfı şöyle kullanılmaktadır:

1) Önce one hot encoding dönüştürmesi yapılacak değerlerin iki boyutlu bir matris biçiminde ifade edilmeleri gereklidir. Bu matrisin elemanları doğrudan yazışal etiketler de olabilir, sayısal değerler de olabilir. Ancak genellikle one hot encoding işlemi numpy dizileri üzerinde yapıldığı için ve numpy dizileri de ortak bir türden olduğu için programcılar yazışal kategorik alanları önce LabelEncoder ya da OrdinalEncoder sınıfları kullanılarak sayısal biçimde dönüştürürler. Örneğin yukarıda verdığımız veri tablosunun aşağıdaki gibi bir "test.csv" dosyası içerisinde bulunduğu varsayıyalım:

```
Tepki Süresi,Renk
1.2,Kırmızı
2.3,Mavi
0.7,Kırmızı
1.2,Yeşil
3.2,Mavi
```

Şimdi bu dosyayı okuyup Renk sütununu LabelEncoder sınıfı ile sayısal biçimde dönüştürelim:

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder

df = pd.read_csv('test.csv')
print(df)

le = LabelEncoder()
df['Renk'] = le.fit_transform(df['Renk'])
print(df)
dataset = df.to_numpy()
print(dataset)
```

Buradan şöyle bir çıktı elde edilecektir:

	Tepki Süresi	Renk
0	1.2	Kırmızı
1	2.3	Mavi
2	0.7	Kırmızı

```

3      1.2  Yeşil
4      3.2  Mavi
Tepki Süresi Renk
0      1.2  0
1      2.3  1
2      0.7  0
3      1.2  2
4      3.2  1
[[1.2 0. ]
 [2.3 1. ]
 [0.7 0. ]
 [1.2 2. ]
 [3.2 1. ]]

```

2) sklearn.preprocessing modülü içerisindeki OneHotEncoder sınıfı türünden bir nesne yaratılır. Bu nesnenin fit_transform metoduna sayısallaştırılmış kategorik değerler parametre olarak verilir. Ancak fit_transform metodu parametre olarak iki boyutlu bir Python listesini, bir numpy dizisini ya da Pandas'ın bir DataFrame nesnesini almaktadır. Eğer elinizde tek boyutlu bir vektör varsa onu tek bir sütundan oluşan bir matris biçimine dönüştürmeniz gereklidir. Bunun için ndarray sınıfının reshape metodundan faydalanabilirsiniz.

```

from sklearn.preprocessing import OneHotEncoder

ohe = OneHotEncoder(sparse=False)
ohe_data = ohe.fit_transform(dataset[:, 1].reshape(-1, 1))
print(ohe_data)

```

Burada fit_transform işleminde ilgili sayısallaştırılmış kategorik verilerin sütun vektörüne dönüştürüldüğüne dikkat ediniz. Bu işleminden şöyle bir çıktı elde edilmiştir:

```

[[1. 0. 0.]
 [0. 1. 0.]
 [1. 0. 0.]
 [0. 0. 1.]
 [0. 1. 0.]]

```

Aslında fit_transform ile iki farklı sütun da tek hamlede one hot encoding dönüştürmesine sokulabilmektedir. fit_transform metoduna genel olarak iki boyutlu bir dizinin geçirilebildiğine dikkat ediniz. Biz bu örnekte fit_transform metoduna LabelEncoder sınıfı ile 0'dan itibaren sayısallaştırılmış sütunu parametre olarak verdik. Aslında sütunun 0'dan itibaren ardışılı numaralarla sayısallaştırılmış olması gerekmektedir. Buradaki sayılar kategori belirttiği için herhangi değerlerde olabilmektedir. fit_transform metodunun sayısal kategorik veriler yerine yazışal kategorik veriler üzerinde de işlem yapabildiğini anımsatmak istiyoruz.

Biz aslında fit ve transform işlemlerini birlikte değil ayrı ayrı da yapabilirdik:

```

from sklearn.preprocessing import OneHotEncoder

ohe = OneHotEncoder(sparse=False)
ohe.fit(dataset[:, 1].reshape(-1, 1))
ohe_data = ohe.transform(dataset[:, 1].reshape(-1, 1))
print(ohe_data)

```

Bu işlemlerin ayrı ayrı yapılması size anlamsız gelebilir. Ancak aslında fit işleminde biz "one hot encoding" işlemi için gereken kategorileri belirtmekteyiz. transform ise gerçek dönüştürmeyi yapmaktadır. Yani örneğin biz bir kere fit işlemi yapıp çok defa transform işlemi yapabiliriz. fit_transform metodu aslında önce fit sonra transform işlemini yapmaktadır. Hem fit metodunun hem de transform metodunun iki boyutlu Python listeleriyle, numpy dizileriyle ya da Pandas'ın DataFrame nesneleriyle çalışmasına dikkat ediniz.

3) Nihayet one hot encoding işlemi sonucunda elde edilen matrisin gerçek veri matrisine yerleştirilmesi gerekmektedir. Bunun için tabii önce gerçek veri matrisindeki sayısallaştırılmış kategorik sütun silinmelidir. İçin bu kısmını Pandas'ın DataFrame üzerinde yapabiliyoruz:

```
df.drop(['Renk'], axis=1, inplace=True)
df[['Kırmızı', 'Yeşil', 'Mavi']] = ohe_data
print(df)
```

Bu işlemden Şöyleden bir çıktı elde edilecektir

	Tepki Süresi	Kırmızı	Yeşil	Mavi
0	1.2	1.0	0.0	0.0
1	2.3	0.0	1.0	0.0
2	0.7	1.0	0.0	0.0
3	1.2	0.0	0.0	1.0
4	3.2	0.0	1.0	0.0

Ya da işin bu kısmını hiç Pandas kullanmadan numpy dizisi üzerinde de yapabiliyoruz:

```
dataset = np.delete(dataset, 1, axis=1)
dataset = np.append(dataset, ohe_data, axis=1)
print(dataset)
```

Şimdi bu adımları tek bir kodla birlestirelim:

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder

df = pd.read_csv('test.csv')
print(df)

le = LabelEncoder()
df['Renk'] = le.fit_transform(df['Renk'])

from sklearn.preprocessing import OneHotEncoder

ohe = OneHotEncoder(sparse=False)
ohe_data = ohe.fit_transform(df['Renk'].to_numpy().reshape(-1, 1))
df.drop(['Renk'], axis=1, inplace=True)
df[['Kırmızı', 'Yeşil', 'Mavi']] = ohe_data
print(df)
```

Kod çalıştırıldığında şöyleden bir çıktı elde edilecektir:

	Tepki Süresi	Renk
0	1.2	Kırmızı
1	2.3	Mavi
2	0.7	Kırmızı
3	1.2	Yeşil
4	3.2	Mavi

	Tepki Süresi	Kırmızı	Yeşil	Mavi
0	1.2	1.0	0.0	0.0
1	2.3	0.0	1.0	0.0
2	0.7	1.0	0.0	0.0
3	1.2	0.0	0.0	1.0
4	3.2	0.0	1.0	0.0

"One hot encoding" işlemi uygulamanın diğer bir yolu da tensorflow.keras.utils modülündeki to_categorical fonksiyonunu kullanmaktır. Bu fonksiyon 0'dan başlayan ardışılı tamsayıların bulunduğu kategorik ndarray nesnesini parametre olarak alır ve geri dönüş değeri olarak da "one hot encoding" biçiminde dönüştürülmüş ndarray matrisi verir. Biz de bu matrisi uygun sütunlarla yer değiştirebiliriz. Örneğin:

```
data = [0, 0, 2, 2, 3, 1, 2, 1, 1]

from tensorflow.keras.utils import to_categorical

ohe_data = to_categorical(data)
print(ohe_data)
```

Kodun çalıştırılması sonucunda şöyle bir çıktı elde edilmiştir:

```
[[1. 0. 0. 0.]
 [1. 0. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 1. 0. 0.]
 [0. 1. 0. 0.]]
```

Biz de bu işlemden elde ettiğimiz "one hot encoding" ndarray nesnesini orijinal nesnenin sütunlarıyla değiştirebiliriz. Şimdi de yukarıdaki renk içeren "test.csv" örneğini to_categorical fonksiyonuyla one hot encoding yapalım:

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.utils import to_categorical

df = pd.read_csv('test.csv')

le = LabelEncoder()
df['Renk'] = le.fit_transform(df['Renk'])

df[['Renk-0', 'Renk-1', 'Renk2']] = to_categorical(df['Renk'])
df.drop(['Renk'], axis=1, inplace=True)
print(df)
```

Kod çalıştırıldığında şöyle bir çıktı elde edilmiştir:

	Tepki Süresi	Renk-0	Renk-1	Renk2
0	1.2	1.0	0.0	0.0
1	2.3	0.0	1.0	0.0
2	0.7	1.0	0.0	0.0
3	1.2	0.0	0.0	1.0
4	3.2	0.0	1.0	0.0

Tabii aslında one hot encoding yapan bir fonksiyon yazmak da oldukça kolay. Örneğin:

```
def one_hot_encoder(dataset, column):
    ncategory = np.max(dataset[:, column]) + 1
    ohe = np.zeros((dataset.shape[0], int(ncategory)), dtype=np.float32)
    for index, category in enumerate(dataset[:, column]):
        ohe[index, int(category)] = 1

    dataset = np.delete(dataset, column, axis=1)
```

```
dataset = np.hstack((dataset, ohe))
```

```
return dataset
```

Bu fonksiyon bizden kategori sütunu sayısal hale dönüştürülmüş olan numpy dizisini ve one hot encoding yapılacak sütun numarasını alır. Bize one hot encoding yapılmış yeni matrisi geri dönüş değeri olarak verir. Örneğin:

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder

def one_hot_encoder(dataset, column):
    ncategory = np.max(dataset[:, column]) + 1
    ohe = np.zeros((dataset.shape[0], int(ncategory)), dtype=np.float32)
    for index, category in enumerate(dataset[:, column]):
        ohe[index, int(category)] = 1

    dataset = np.delete(dataset, column, axis=1)
    dataset = np.hstack((dataset, ohe))

return dataset

df = pd.read_csv('test.csv')
print(df)

le = LabelEncoder()
df['Renk'] = le.fit_transform(df['Renk'])
print(df)

dataset = one_hot_encoder(df.to_numpy(), 1)
print(dataset)
```

Kod çalıştırıldığında şöyle bir çıktı elde edilecektir:

```
Tepki Süresi      Renk
0            1.2  Kırmızı
1            2.3    Mavi
2            0.7  Kırmızı
3            1.2    Yeşil
4            3.2    Mavi
Tepki Süresi  Renk
0            1.2    0
1            2.3    1
2            0.7    0
3            1.2    2
4            3.2    1
[[1.2 1.  0.  0. ]
 [2.3 0.  1.  0. ]
 [0.7 1.  0.  0. ]
 [1.2 0.  0.  1. ]
 [3.2 0.  1.  0. ]]
```

One hot encoding dönüştürmesi kategorik veriler için iyi sonuçlar doğursa da bazı dezavantajlara da sahiptir. Örneğin veri tablosundaki bir sütunun kategori sayısı çok fazla olabilir. Bu durumda one hot encoding çok fazla sütunun tabloya eklenmesine yol açar. Bu tür durumlarda one hot encoding yerine başka bir yöntem (örneğin "binary encoding") tercih edilebilir ya da one hot encoding sonrasında "boyutsal özellik indirgemesi (dimensionality feature reduction)" uygulanabilir. (Boyutsal özellik indirgemesi kursumuzda ileride ele alınmaktadır.) One hot encoding işlemi için tüm kategorilerin neler olduğunu işin başında bilinmeyip olması gerekmektedir. Halbuki pek çok durumda toplam kategorilerin sayısı işin başında bilinmez. Bu tür durumlarda hash tabloları oluşturulabilir ve bu tablolara göre kodlama

yapılabilir. Kategori sayısının çok fazla olduğu durumlarda sıkılıkla uygulanan yöntemlerden bir diğeri de "binary encoding" yöntemidir.

Binary Encoding Yöntemi

Binary Encoding yöntemi kategorik değerlerin sayısı n olmak üzere $\log_2 n$ tane sütunla ifade edilmesini sağlamaktadır. Bu yöntemde sanki toplamda n tane farklı kategorik değer k tane ($k = \log_2 n$) sütun ile ikilik sistemde kodlanmaktadır. Örneğin toplamda 64 farklı kategorik değer söz konusu olsun. Bu durumda bu kategorik değerler $\log_2 64 = 6$ tane sütun ile ikilik sistemde kodlanabilir. Örneğin:

Sütun-1	Sütun-2	Sütun-3	Sütun-4	Sütun-5	Sütun-6	Kategori
0	0	0	0	0	0	C1
0	0	0	0	0	1	C2
...
1	1	1	1	1	0	C63
1	1	1	1	1	1	C64

Binary Encoding işlemi scikit-learn kütüphanesindeki `category_encoders` modülündeki `BinaryEncoder` sınıfı ile gerçekleştirilebilmektedir. Ancak `category_encoders` modülü `sklearn` paketleri içerisinde değildir. Onun ayrıca install edilmesi gerekmektedir:

```
pip install category_encoders
```

`BinaryEncoder` sınıfı `OneHotEncoder` sınıfında olduğu gibi kategori sayısı kadar sütun oluşturmaktadır. Bu nedenle elde edilen pek çok sütun tamamen sıfırlardan oluşabilir. `BinaryEncoder` sınıfı bizden `pandas DataFrame` nesnesini alıp bize yine `DataFrame` de verebilmektedir. Örneğin:

```
import pandas as pd
from category_encoders.binary import BinaryEncoder

df = pd.read_csv('test.csv')
print(df)

be = BinaryEncoder()
be_df = be.fit_transform(df[ 'Renk' ])
print(be_df)
```

Kodun çalıştırılması sonucunda şöyle bir çıktı elde edilecektir:

```
Tepki Süresi      Renk
0      1.2  Kırmızı
1      2.3    Mavi
2      0.7  Kırmızı
3      1.2    Yeşil
4      3.2    Mavi
Renk_0  Renk_1  Renk_2
0      0      0      1
1      0      1      0
2      0      0      1
3      0      1      1
4      0      1      0
```

Şimdi `DataFrame` içerisindeki 'Renk' sütununu silelim, elde edilen sütunlardan sıfır olanları atalım ve iki `DataFrame` nesnesini birlestirelim:

```

df.drop(['Renk'], axis=1, inplace=True)
be_df = be_df.loc[:, (be_df != 0).any(axis=0)]

result_df = pd.concat([df, be_df], axis=1)
print(result_df)

```

Şöyledir bir çıktı elde edilecektir:

	Tepki Süresi	Renk_1	Renk_2
0	1.2	0	1
1	2.3	1	0
2	0.7	0	1
3	1.2	1	1
4	3.2	1	0

Sıralı (Ordinal) Verilerin Sayısal Biçime Dönüşürtlmesi

Sıralı verilerin sayısal biçimde dönüştürülmesi işlemi için yüksek seviyeli kütüphanelerde hazır bir fonksiyon ya da metod bulunmaktadır. Bu işlem en pratik biçimde pandas kütüphanesindeki Series ya da DataFrame sınıflarının replace metotlarıyla gerçekleştirilebilir. Örneğin aşağıdaki gibi bir "test.csv" dosyamız olsun:

```

Cinsiyet,Kilo,Boy,Eğitim
Erkek,85,172,İlkokul
Kadın,72,170,Üniversite
Kadın,65,162,Lise
Erkek,92,183,Lise
Kadın,62,173,İlkokul
Erkek,98,172,Ortaokul

```

Burada "Eğitim" sütunu sıralı ölçüde ilişkindir. Şimdi biz DataFrame sınıfının replace metodunu kullanarak eğitim bilgilerini 0'dan itibaren artan bir sayıyla ifade etmeye çalışalım. Önce dosyayı pandas.read_csv fonksiyonu ile ilk sütunu atarak okuyalım:

```

import pandas as pd

df = pd.read_csv('test.csv', encoding='utf-8', usecols=[1, 2, 3])
print(df)

```

Elde edilen DataFrame aşağıdaki gibidir:

	Kilo	Boy	Eğitim
0	85	172	İlkokul
1	72	170	Üniversite
2	65	162	Lise
3	92	183	Lise
4	62	173	İlkokul
5	98	172	Ortaokul

Şimdi DataFrame sınıfının replace metodunu kullanarak dönüştürmemizi yapalım:

```

replace_dict = {'Eğitim': {'İlkokul': 0, 'Ortaokul': 1, 'Lise': 2, 'Üniversite': 3}}
df.replace(replace_dict, inplace=True)
print(df)

```

Kodun çalıştırılması sonucunda öyle bir çıktı elde edilecektir:

	Kilo	Boy	Eğitim
0	85	172	0

```

1    72  170      3
2    65  162      2
3    92  183      2
4    62  173      0
5    98  172      1

```

Artık bu DataFrame nesnesini to_numpy metodu ile ndarray nesnesine dönüştürebiliriz:

```
dataset = df.to_numpy()
print(dataset)
```

```
[[ 85 172  0]
 [ 72 170  3]
 [ 65 162  2]
 [ 92 183  2]
 [ 62 173  0]
 [ 98 172  1]]
```

Aynı işlemi Series nesnesi üzerinde de -biraz daha dolaylı olarak- şöyle yapabilirdik:

```
replace_dict = {'İlkokul': 0, 'Ortaokul': 1, 'Lise': 2, 'Üniversite': 3}
df['Eğitim'] = df['Eğitim'].replace(replace_dict)
```

Bu işlemi yalnızca numpy kullanarak yapmak daha zahmetlidir. Çünkü numpy kütüphanesinin Pandas kütüphanesindeki gibi bir replace fonksiyonu yoktur.

```
import numpy as np

data = np.loadtxt('test.csv', delimiter=',', skiprows=1, usecols=[1, 2, 3], encoding='utf-8',
dtype=np.object)

dataset = data[:, [0, 1]].astype(np.float32)
edu = data[:, 2]

replace_dict = {'İlkokul': 0, 'Ortaokul': 1, 'Lise': 2, 'Üniversite': 3}
v = np.vectorize(replace_dict.get)
dataset = np.insert(dataset, 2, v(edu), axis=1)
print(dataset)
```

Burada önce "test.csv" dosyasını numpy.loadtxt fonksiyonu ile dtype=np.object parametresi ile okuduk. numpy dizilerinin tek bir dtype türü olduğu için fonksiyon tüm değerleri string olarak okudu. Sonra "Kilo" ve "Boy" sütunlarını np.float32 türüne dönüştürerek dataset isimli yeni bir numpy dizisi içerisinde yerleştirdik. Nihayet "Eğitim" sütununu da numpy.vectorize fonksiyonu ile sıralı sayısal bir sütuna dönüştürerek bunu yeni bir sütun biçiminde ekledik. Yukarıdaki kod çalıştırıldığında aşağıdaki gibi bir çıktı elde edilecektir:

```
[[ 85. 172.  0.]
 [ 72. 170.  3.]
 [ 65. 162.  2.]
 [ 92. 183.  2.]
 [ 62. 173.  0.]
 [ 98. 172.  1.]]
```

Tabii numpy içerisinde bu işlemi yapmanın başka çok çeşitli yolları da vardır. Örneğin bu işlem aslında loadtxt fonksiyonu sırasında fonksiyonun converters parametresi kullanılarak da yapılabilir:

```
import numpy as np
```

```

replace_dict = {'İlkokul': 0, 'Ortaokul': 1, 'Lise': 2, 'Üniversite': 3}
dataset = np.loadtxt('test.csv', delimiter=',', skiprows=1, usecols=[1, 2, 3], encoding='utf-8',
converters={1: lambda x: float(x), 2: lambda x: float(x), 3: lambda x: replace_dict[x]},
dtype=np.object)
print(dataset)

```

Tabii sizin de gördüğünüz gibi bu tür işlemleri yalnızca numpy kütüphanesi ile yapmak yerine pandas kütüphanesini kullanarak yapmak ve en sonunda DataFrame nesnesini ndarray nesnesine dönüştürmek daha pratik ve kolay bir yöntemdir. Biz kursumuzda bazen doğrudan numpy kütüphanesini kullanırken bazen de Pandas kütüphanesinin sağladığı kolaylıklarını kullanacağız.

Denetimli (Supervised) Yapay Sinir Ağları ve Veri Kümeleri

Daha önceden makine öğrenmesini ve yapay sinir ağlarını denetimli (supervised), denetimsiz (unsupervised) ve pekiştirmeli (reinforcement) olmak üzere üçe ayırmıştık. Denetimli (supervised) yapay sinir ağlarında ağıın mimarisini oluşturulduktan sonra gerçek verilerle ağıın etgitilmesi (training) gerekmektedir. Bu eğitim sırasında nöronların w katsayıları ayarlanmaktadır. İşte bu eğitim işleminde kullanılmak üzere elimizde gerçek örneklerin bulunması gereklidir. Eğitimde kullanılan gerçek örneklem kümesine "eğitim veri kümesi (traning data set)" denilmektedir. Ağ eğitildikten sonra ağıın test edilmesi için de gerçek verilerden oluşan bir veri kümesine gereksinim duyulmaktadır. Buna da "test veri kümesi (test data set)" denilmektedir. O halde bizim gerçek verileri toplayıp bu verileri "eğitim veri kümesi" ve "test veri kümesi" olmak üzere ikiye ayırmamız gereklidir. Eğitimde kullanılan veri kümesinin aynı zamanda test işleminde kullanılması yanlış bir tekniktir. Çünkü ağı eğitilirken öğrenilen şeyler test edilirken kullanılmamalıdır. (Bu durumu şuna benzetebiliriz: Öğretmen derste birtakım sorular çözüp sınavda da aynısını sorarsa bu sınav gerçek durumu kestirmekte başarısız olabilir.)

Pekiyi eğitim veri kümesi ve test veri kümesi arasındaki oran nasıl olmalıdır? Yani topladığımız verilerin yüzde kaç eğitim veri kümesi için yüzde kaç test veri kümesi için kullanılmalıdır? İşte yapay sinir ağlarında ve derin öğrenme ağlarında aslında kesin kurallar yoktur. Pek çok seçim istege bağlıdır. Örneğin uygulamalarda tipik olarak verilerin %80'i eğitim veri kümesi için %20'si ise test veri kümesi için ayrılmaktadır. Ancak veri kümesi çok büyükse test kümesinin oranı düşürülebilir.

Yukarıda ağıın eğitilmesi ve test edilmesi için verilerin bir biçimde elde edilmiş olması gerektiğini söyledik. Verilerin elde edilmesi ayrı bir süreci oluşturmaktadır. Kursumuz bu verilerin elde edilmesi süreci ile ilgili değildir. Bu nedenle biz bu kursta vereceğimiz örneklerde zaten elde edilmiş olan hazır veriler üzerinde işlemlerimizi yapacağız.

Elde edilmiş veriler "eğitim veri kümesi" ve "test veri kümesi" biçiminde ikiye ayrıldıktan sonra ayrıca bunların da "girdi veri kümesi" ve "çıkı veri kümesi" biçiminde ayrıştırılması gereklidir. Örneğin bir banka müşterisinin 13 özelliğine bakarak o müşterinin 3 ay içerisinde bankayı terk edip etmeyeceğinin kestirilmeye çalışıldığını düşünelim. İşte bizim bu kestirimini yapabilmemiz için 13 özelliği olan müşterilerin bankayı 3 ay içerisinde terk edip etmediğine yönelik -bir biçimde elde edilmiş olan- bir veri tablomuzun olması gereklidir. Bu veri tablosunda girdi kümesi 13 özellikten, çıktı kümesi ise bankayı tek edip etmediğine ilişkin (örneğim 0 ya da 1) bir özellikten oluşmalıdır.

Şimdi verilerin bu biçimde ayrıştırılmasına örnek verelim. Örneğimizde "pima-indians-diabetes" isimli bir veri tablosunu kullanacağız. "pima-indians-diabetes" veri tablosu bir kişinin 8 özelliğine bakarak onun şeker hastası olup olmadığıının kestirilmesi için hazırlanmış gerçek bir veri tablosudur. Tablodaki 8 sütunun anlamları şöyledir:

```

# 1. Number of times pregnant
# 2. Plasma glucose concentration a 2 hours in an oral glucose tolerance test
# 3. Diastolic blood pressure (mm Hg)
# 4. Triceps skin fold thickness (mm)
# 5. 2-Hour serum insulin (mu U/ml)
# 6. Body mass index (weight in kg/(height in m)^2)
# 7. Diabetes pedigree function

```

```
# 8. Age (years)
# 9. Class variable (0 or 1)
```

Bu veri tablosu <https://www.kaggle.com/uciml/pima-indians-diabetes-database> adresinden .csv dosyası biçiminde indirilebilir. Dosyanın görüntüsü şöyledir:

```
Pregnancies,Glucose,BloodPressure,SkinThickness,Insulin,BMI,DiabetesPedigreeFunction,Age,Outcome
6,148,72,35,0,33.6,0.627,50,1
1,85,66,29,0,26.6,0.351,31,0
8,183,64,0,0,23.3,0.672,32,1
1,89,66,23,94,28.1,0.167,21,0
0,137,40,35,168,43.1,2.288,33,1
5,116,74,0,0,25.6,0.201,30,0
3,78,50,32,88,31,0.248,26,1
10,115,0,0,0,35.3,0.134,29,0
2,197,70,45,543,30.5,0.158,53,1
8,125,96,0,0,0,0.232,54,1
4,110,92,0,0,37.6,0.191,30,0
10,168,74,0,0,38,0.537,34,1
10,139,80,0,0,27.1,1.441,57,0
1,189,60,23,846,30.1,0.398,59,1
...
...
```

Bu tablo gerçek veriler kullanılarak oluşturulmuştur. Tablodaki her satır farklı bir kişinin bilgisini içermektedir. Her satırın ilk 8 sütunu kişinin çeşitli özelliklerini 9'uncu son sütun ise o kişinin şeker hasatası olup olmadığını belirtmektedir. Bu son sütundaki değerin 1 olması kişinin şeker hastası olduğunu, 0 olması ise kişinin şeker hastası olmadığını göstermektedir. Tabii bizim aslında bu bölümdeki amacımız yapay sinir ağlarını kullanarak yeni bir kişinin bu sekiz özelliğine bakarak onun belli güvenilirlikle şeker hastası olup olmadığını anlamaktır.

Bu veri tablosunun ayrıştırılmasını numpy kullanarak şöyle yapılabiliriz:

```
import numpy as np

dataset = np.loadtxt('diabetes.csv', dtype='float32', skiprows=1, delimiter=',')
dataset_x = dataset[:, :8]
dataset_y = dataset[:, 8]

tzone = int(len(dataset) * 0.80)

training_dataset_x = dataset_x[:tzone, :]
training_dataset_y = dataset_y[:tzone]

test_dataset_x = dataset_x[tzone:, :]
test_dataset_y = dataset_y[tzone:]
```

Burada önce dosya okunmuştur. Veri kümesi içerisinde kategorik bir sütunun olmadığına dikkat ediniz. Sonra veri kümesi girdi ve çıktı biçiminde (dataset_x, dataset_y) ayrıştırılmış sonra bu kümelerde de kendi aralarında %80 noktasından eğitim ve test veri kümesi biçiminde (training_dataset_x, training_datset_y) yeniden ayrıstırılmıştır.

Aslında yukarıdaki manuel işlem scikit-learn kütüphanesindeki tek bir fonksiyonla da yapılabilmektedir. Bu işlemi yapan `sklearn.model_selection` modülündeki `train_test_split` isimli fonksiyonun parametrik yapısı şöyledir:

```
sklearn.model_selection.train_test_split(*arrays, test_size=None, train_size=None,
random_state=None, shuffle=True, stratify=None)
```

Fonksiyona biz dataset_x ve dataset_y vektörlerini ayrı parametreler biçiminde veririz. Fonksiyon da bize training_dataset_x, test_dataset_x, training_dataset_y ve test_datatset_y biçiminde dörtlü bir demet verir. Fonksiyonun test_size isimli parametresi test kümесinin yüzde kaç olacağını bizden istemektedir. Bu parametre 0 ile 1 arasında float bir değer biçiminde girilmelidir. Örneğin:

```
import numpy as np

dataset = np.loadtxt('diabetes.csv', dtype='float32', skiprows=1, delimiter=',')
dataset_x = dataset[:, :8]
dataset_y = dataset[:, 8]

from sklearn.model_selection import train_test_split

training_dataset_x, test_dataset_x, training_dataset_y, test_datatset_y =
train_test_split(dataset_x, dataset_y, test_size=0.2)
```

Göründüğü gibi train_test_split isimli fonksiyon bizden girdi ve çıktı vektörünü alıp onu istediğimiz oranda parçalara ayırmaktadır. Yukarıda da belirttiğimiz gibi buradaki test_size isimli argümanı test verilerinin yüzdesini belirtmektedir. Bu yüzde 0 ile 1 arasında float bir sayı biçiminde girilmelidir. Yukarıdaki örnekte test_size=0.2 girişi eğitim veri kümесinin %80, test veri kümесinin %20 olacağı anlamına gelmektedir. Eğer fonksiyonda test_size parametresi belirtilmezse default olarak 0.25 alınmaktadır.

train_test_split fonksiyonu default durumda verilen x ve y vektörlerini karıştırıp ondan sonra eğitim ve test kümescini ayırtmaktadır. Eğer bunun yapılmasını istenmiyorsanız (bazen zamansal verilerde bunun yapılmasını istemeyebilirsiniz) fonksiyonun shuffle parametresini False geçmelisiniz.

Keras Kütüphanesinin Kurulması

Önceden de belirttiğimiz gibi Keras yüksek seviyeli bir yapay sinir ağı kütüphanesidir. Keras 2.3 sürümüne kadar birden fazla arkayı (backend) destekliyordu (TensorFlow, Microsoft Cognitive Toolkit, Theano, PlaidML). Ancak 2.4 sürümüyle birlikte artık Keras arkayı olarak yalnızca TensorFlow kütüphanesini kullanmaktadır. Bu nedenle Keras kütüphanesi yüklenmeden önce TensorFlow yüklenmiş olmalıdır.



Keras yalnızca Python kullanılarak, TensorFlow ise C++ ve Python kullanılarak yazılmıştır. Bu iki kütüphanenin yüklenmesi pip komutlarıyla şöyle yapılabilir:

```
pip install tensorflow
pip install keras
```

Tabii yükleme işlemini Anaconda Navigator sekmelerinden ya da PyCharm menülerinden görsel olarak da yapabilirsiniz. Kurasumuzun verildiği sırada Keras'in son sürümü 2.4.0, TensorFlow'un ise 2.5.0'dır. Ancak burada dikkat edilmesi gereken bir nokta şudur: Maalesef Tensorflow kütüphanesini ile kullanabilecek Python sürümleri farklıdır. Eğer Python sürümünüz 3.8 ise sizin TensorFlow 2.2 ya da daha sonraki bir versiyonu yüklemeniz gereklidir. Eğer Python versiyonunuz 3.9 ise sizin Tensorflow 2.5'i yüklemeniz gereklidir. Daha eski Python sürümleri için istediğiniz TensorFlow sürümünü yükleyebilirsiniz.

Tensorflow 2'li versiyonlarla birlikte Keras kütüphanesini bünyesine dahil etmiştir. Yani artık Keras kütüphanesi Tensorflow içerisinde tensorflow.keras paketi biçiminde bulunmaktadır. Ancak Keras ayrı bir kütüphane gibi de şimdilik

kendini devam ettirmektedir. Fakat maalesef Keras'in ayrı bir kütüphane biçiminde kurulumu sırasında TensorFlow kütüphanesi ile versiyon uyşumazlığı yaşanabiliyor. Bu nedenle Keras'ı eğer ayrı bir kütüphane olarak kuracaksanız - kursun yapıldığı tarihte- TensorFlow kütüphanesinin 2.2 versiyonunu yüklemelisiniz. Ayrıca artık bu kursla birlikte Keras kütüphanesinin Tensorflow bünyesine katılmış biçimini kullanacağınız. Halbuki önceki kurslarda ayrı bir biçimde kurulan Keras kütüphanesini kullanıyorduk.

Pekiyi bağımsız Keras kurulumu ile Tensorflow içerisindeki Keras kurulumu arasında bir fark var mıdır? Aslında Keras'in ileri sürümleri de zaten TensorFlow içerisindeki Keras ile uyumlu hale getirilmektedir. Ancak TensorFlow içerisindeki Keras, TensorFlow kütüphanesine yönelik bazı aşağı seviyeli işlemlerinin yapılmasına da izin verdiği için bu durum Keras ile TensorFlow kütüphanelerini beraber kullanmak isteyen programcılar için bir avantaj oluşturabiliyor. Yukarıda da belirttiğimiz gibi biz kursumuzla birlikte artık TensorFlow içerisindeki Keras'ı kullanacağız.

Keras'ta Yapay Sinir Ağlarının Oluşturulması

Keras yüksek seviyeli bir kütüphanedir. Dolayısıyla işlemler yüksek seviyeli biçimde yürütülmektedir. Keras modelinde programcının kabaca ağaç için şu belirlemeleri yapmış olması gereklidir:

- Ağdaki katmanlar
- Katmanlardaki nöronların sayısı
- Nöronların kullanacağı aktivasyon (transfer) fonksyonları
- Eğitimde ve teste kullanılacak girdi ve çıktı veri kümeleri
- Sistemin amaçlarına yaklaşlığını ölçmek için kullanılacak "amaç fonksiyonu (loss function)"
- w değerlerinin iyileştirme yöntemini belirleyen optimizasyon algoritması

Biz burada önce Keras'ta işlem adımlarını kabaca ele alacağımız sonra ayrıntılara gireceğiz. Keras'ta işlemler tipik olarak (her zaman değil) aşağıdaki beş aşamadan geçilerek gerçekleştirilmektedir:

1) Modelin Oluşturulması: Keras'ta yapay sinir ağı bir sınıf nesnesi ile temsil edilmektedir. Bu amaçla en yaygın kullanılan sınıf Sequential isimli sınıftır. Programcı bu Sequential sınıfı türünden bir nesne yaratır. Yaratılan bu nesneye genellikle "model" denilmektedir. Örneğin:

```
from tensorflow.keras.models import Sequential  
model = Sequential()
```

Keras'ta Sequential dışında başka model sınıfları da vardır. Bu sınıflar ileride ele alınacaktır.

2) Modele Katmanların Eklenmesi: Bu aşamada artık katmanları modele eklememiz gereklidir. Katman ekleme işlemi için Sequential sınıfının add metodunu kullanılmaktadır. add metodunu bizden argüman olarak katman istemektedir. Keras'ta katmayı temsil eden çeşitli sınıflar vardır. Bunların en çok kullanılanı Dense isimli sınıftır. Dense katmanı (dense yoğun anlamına geliyor) bu katmandan önceki katmanın her nöronunun bu katmanın her nöronu ile bağlanacağı anlamına gelmektedir. Dense sınıfının __init__ metodu aşağıdaki gibidir:

```
tensorflow.keras.layers.Dense(units,  
activation=None,  
use_bias=True,  
kernel_initializer='glorot_uniform',  
bias_initializer='zeros',  
kernel_regularizer=None,  
bias_regularizer=None,  
activity_regularizer=None,  
kernel_constraint=None,  
bias_constraint=None,
```

```
    **kwargs  
)
```

Metodun units parametresi ilgili katmandaki nöron sayısını, activation parametresi ise o katmandaki tüm nöronların aktivasyon (transfer) fonksiyonunu belirtmektedir. Burada fonksiyon isim olarak ya da fonksiyon nesnesi olarak girilebilmektedir. Eğer buradaki fonksiyon bir fonksiyon nesnesi olarak girilecekse tensorflow.keras.Activation modülündeki fonksiyon nesneleri kullanılmalıdır. use_bias parametresi eğer True geçilirse (default durum) bu durumda bias_initializer parametresi nörondaki "bias" değeri olarak kullanılmaktadır. bias değerinin anlamı ileride ele alınacaktır. kernel_initializer parametresi başlangıçtaki 'w' değerlerinin dağılımını belirtmektedir. Yani biz işin başında bu 'w' değerlerini rastgele biçimde belli bir dağılıma göre oluşturabiliriz. Bu parametre belirtilmeyebilir. Bu durumda w değerlerinin başlangıçtaki rastgele dağılımı "glorot_uniform" biçimde olacaktır. "glorot_uniform" dağılımı (buna "Xavier initialization" da denilmektedir) bir çeşit sürekli düzgün dağılımdır.

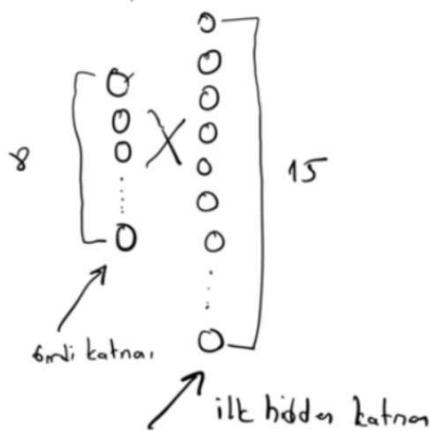
Biz Dense fonksiyonunda oluşturduğumuz katmanlara name parametresi yoluyla isimler de verebiliriz. Katmanlara isimler vermek bazen gereklidir. Eğer biz katmanlara name parametresi yoluyla isim vermemek bu durumda Dense fonksiyonu katmanlara kendisi otomatik isimler vermektedir. Biz genel olarak örneklerimizde katmanlara isimler vereceğiz.

fit metodun diğer parametreleri şimdilik ele alınmayacaktır. Ancak bunların da çeşitli default değerler aldığına görürorsunuz.

Dense fonksiyonunda input_dim isimli özel parametre (bu parametre ** parametresi yoluyla alınmaktadır) girdi katmanındaki nöron sayısını belirtmek için kullanılmaktadır. Zaten girdi katmanı normal bir katman gibi olmadığı için girdi katmanı için ayrı bir katman oluşturulmamaktadır. Yani ağa eklenen ilk katmanda input_dim girdi katmanındaki nöron sayısını (yani girdi değişkenlerinin sayısını) belirtir. Örneğin:

```
from tensorflow.keras.models import Sequential  
from tensorflow.keras.layers import Dense  
  
model = Sequential()  
  
model.add(Dense(15, input_dim=8, activation='relu', name='Hidden-1'))
```

Burada biz 15 nöronlu bir saklı (hidden) katman oluşturduk. Bu modelimizin ilk saklı katmanı olduğu için bu katmanda input_dim ile girdi katmanındaki nöronların sayısını da belirttik.



Aslında girdi katmanındaki nöronların sayısı input_dim parametresi yerine input_shape parametresiyle de girilebilmektedir. Bu iki parametrenin tek farkı input_dim parametresinin int bir değer alması, input_shape parametresinin ise int elemanlardan oluşan bir demet parametresi almasıdır. Bazı modellerde girdiler tek boyutlu değil

çok boyutlu da olabilmektedir. Bu durumda input_dim yerine input_shape parametresinin kullanılması gerekmektedir. Bu durumda yukarıdaki Dense katmanı eşdeğer biçimde şöyle de oluşturulabilirdik:

```
model.add(Dense(15, input_shape=(8, ), activation='relu', name='Hidden-1'))
```

Burada saklı katmanın transfer fonksiyonu "relu (rectifier linear unit)" olarak alınmıştır. Pek çok problem türü için saklı katmanlarda en fazla kullanılan transfer (aktivasyon) fonksiyonu budur. Bu fonksiyon negatif x değerleri için 0, pozitif x değerleri için x değerini veren basit bir fonksiyondur:

$$f(x) = \max(0, x)$$

relu fonksiyonu aşağıdaki gibi basit bir biçimde yazılabilir:

```
def myrelu(x):
    return np.maximum(0, x)
```

Ya da örneğin şöyle de yazılabilir:

```
def myrelu(x):
    return x * (x > 0)
```

relu fonksiyonun grafiğini de şöyle çizdirebiliiz:

```
import numpy as np
import matplotlib.pyplot as plt

def relu(x):
    return np.maximum(0, x)

x = np.linspace(-10, 10, 100)
y = relu(x)

plt.title('RELU Fonksiyonunun Grafiği')
plt.plot(x, y)
plt.show()
```

Keras'ta yaygın biçimde kullanılan aktivasyon fonksiyonları tensorflow.keras.activations modülünün içerisinde hazır biçimde bulunmaktadır. Ancak bu fonksiyonlar Tensorflow kütüphanesi içerisinde kullanılacak biçimde tasarlanılmışlardır. Dense sınıfının add metodunda aktivasyon fonksiyonları isim olarak değil de fonksiyon nesnesi olarak de belirtilebilmektedir:

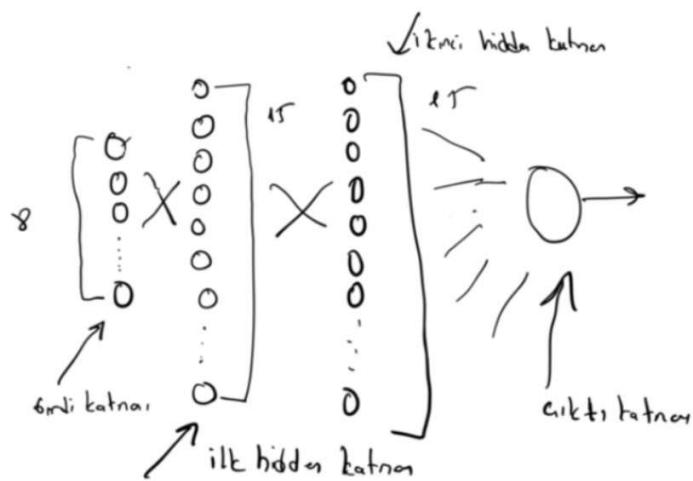
```
from tensorflow.keras.activations import relu

model.add(Dense(15, input_dim=8, activation=relu, name='Hidden-1'))
```

Şimdi ikinci saklı katmanı ve çıktı katmanını da ağa ekleyelim:

```
model.add(Dense(15, activation='relu', name='Hidden-2'))
model.add(Dense(1, activation='sigmoid', name='Output'))
```

Böylece ağımızda 4 katman oluşturmuş olduk: 1 tane girdi katmanı + 2 tane saklı katman + 1 tane çıktı katmanı.



Ağın çıktı katmanı için özel bir belirleme yapılmamaktadır. Her zaman son katman zaten çıktı katmanı olmaktadır. Dense katmanın "önceki katmandaki nöronların hepsinin bu katmandaki nöronların hepsiyle bağlantılı olacağı" anlamına geldiğine bir kez daha dikkat ediniz. Bu modelde çıktı katmanından aktivasyon fonksiyonu olarak "sigmoid" fonksiyonu kullanılmıştır. Sigmoid fonksiyonu iki sınıflı lojistik regresyon modellerinde çıktı katmanında en fazla tercih edilen aktivasyon fonksiyonudur:

$$\text{Sigmoid}(x) = \frac{1}{1 + e^x}$$

Sigmoid fonksiyonun grafiğini çizelim:

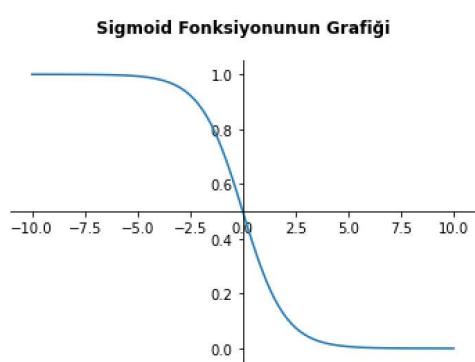
```
import numpy as np
import matplotlib.pyplot as plt

def sigmoid(x):
    return 1 / (1 + np.e ** x)

x = np.linspace(-10, 10, 100)
y = sigmoid(x)

plt.title('Sigmoid Fonksiyonunun Grafiği', weight='bold', pad=20)
axis = plt.gca()
axis.spines['left'].set_position('center')
axis.spines['bottom'].set_position('center')
axis.spines['top'].set_color(None)
axis.spines['right'].set_color(None)

plt.plot(x, y)
plt.show()
```



Şimdi "diabetes.csv" dosyasını okuyarak girdileri bu modele uygulayalım:

```

import numpy as np
from sklearn.model_selection import train_test_split

dataset = np.loadtxt('diabetes.csv', delimiter=',', skiprows=1)
dataset_x = dataset[:, :8]
dataset_y = dataset[:, 8]
training_dataset_x, test_dtaset_x, training_dataset_y, test_dataset_y =
train_test_split(dataset_x, dataset_y, test_size=0.2)

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential()

model.add(Dense(100, input_dim=8, activation='relu', name='Hidden-1'))
model.add(Dense(100, activation='relu', name='Hidden-2'))
model.add(Dense(1, activation='sigmoid', name='Output'))

```

Yukarıda da belirttiğimiz gibi genellikle ikili sınıflandırma (binary classification) tarzı problemlerde saklı katmanların transfer fonksiyonları "relu", çıktı katmanının transfer fonksiyonu ise "sigmoid" olarak alınmaktadır.

3) Modelin Derlenmesi: Modele katmanlar eklendikten sonra artık modelin derlenmesi gereklidir. Buradaki "derleme" terimi programlamadaki derleme anlamında kullanılmamaktadır. Bir çeşit konfigüre etmek anlamında kullanılmaktadır. Konfigüre etme işlemi Sequential sınıfının compile metodu ile yapılmaktadır. Metodun parametrik yapısı şöyledir:

```

Dense.compile(optimizer='rmsprop',
  loss=None,
  metrics=None,
  loss_weights=None,
  weighted_metrics=None,
  run_eagerly=None,
  steps_per_execution=None,
  **kwargs
)

```

compile metodundaki üç önemli parametre optimizer, loss ve metrics parametreleridir. Diğer parametrelerin daha az önemi vardır. Yapay sinir ağlarını eğitirken her eğitim işleminden (epoch) sonra gerçek değerlerle elde edilen değerler arasındaki hata farkı bir biçimde hesaplanmaktadır. İşte bu hesaplanma yöntemine "loss" fonksiyonu denilmektedir. Örneğin bir regresyon modelinde gerçekte olması gereken çıktı 172 iken ağımızın o anda bunu 182 olarak bulduğunu düşünelim. Burada bir fark söz konusudur. Normal olarak bu farkın minimize edilmesi gereklidir. İşte loss fonksiyonu bu farkı ölçmek için kullanılan fonksiyondur. İşlevinden dolayı bu fonksiyona "amaç fonksiyonu" da denilmektedir. Loss fonksiyonları problemin türüne göre seçilmektedir. Lojistik olmayan regresyon tarzı problemlerle lojistik regresyon problemlerinde kullanılabilen loss fonksiyonları farklıdır. Yapay sinir ağlarında çok kullanılan loss fonksiyonları şunlardır:

- **Mean Squared Error:** Lojistik olmayan regresyon modellerinde kullanılır. Bu tür problemlerde en fazla tercih edilen loss fonksiyonudur. Bu yöntemde olması gereken değerlerle olan değerler arasındaki farkın ortalaması bulunmaya çalışılmaktadır. Bu fark ne kadar küçükse hedefe o kadar yaklaşıldığı anlamı çıkar.

$$MSE = \frac{1}{n} \sum_{i=1}^n (Y_i - \hat{Y}_i)^2$$

Burada Y değeri tahmin edilen değeri \hat{Y} ise gerçek değeri belirtmektedir.

- **Mean Absolute Error:** Bu da lojistik olmayan regresyon modellerinde kullanılmaktadır. Fikir MSE ile aynıdır. Yalnızca kare yerine mutlak değer işlemi uygulanmaktadır:

$$MAE = \frac{1}{n} \sum_{i=1}^n |Y_i - \hat{Y}_i|$$

Burada da yine Y değeri tahmin edilen değeri \hat{Y} ise gerçek değeri belirtmektedir.

- **Mean Absolute Percentage Error:** Lojistik olmayan regresyon modellerinde kullanılmaktadır. Aşağıdaki gibi hesaplanır:

$$MAPE = \frac{100}{n} \sum_{i=1}^n \left| \frac{Y_i - \hat{Y}_i}{\hat{Y}_i} \right|$$

- **Mean Square Logarithmic Error:** Lojistik olmayan regresyon modellerinde kullanılmaktadır.

- **Binary Cross-Entropy:** Bu "loss" fonksiyonu ikili sınıfı lojistik regresyon modellerinde en çok tercih edilen loss fonksiyonudur. Örneğin biyomedikal verilerden hareketle kişinin diyabetli olup olmadığını belirlemek için oluşturulan model iki sınıfı lojistik regresyon modelidir. Bu modelde loss fonksiyonu "binary cross-entropy" seçilebilir.

- **Categorical Cross-Entropy:** Bu loss fonksiyonu ikiden fazla sınıfı lojistik regresyon modellerinde en çok tercih edilen loss fonksiyonudur. Örneğin bir resmin üzerinde yazan rakamın belirlenmesine yönelik bir model çok sınıfı lojistik regresyon problemidir. Böyle bir modelde loss fonksiyonu "categorical cross-entropy" seçilebilir.

Yukarıdaki loss fonksiyonları compile metoduna isimsel olarak girebiliriz. Bu fonksiyonlar için kullanılabilecek isimler şunlardır:

```
'mean_squared_error' ya da 'mse'  
'mean_absolute_error' ya da 'mae'  
'mean_absolute_percentage_error' ya da 'mape'  
'mean_squared_logarithmic_error' ya da 'msle'  
'categorical_crossentropy'  
'binary_crossentropy'
```

loss fonksiyonları isimsel fonksiyon nesnesi olarak da belirtilebilir. Keras'taki tüm loss fonksiyonları tensorflow.keras.losses modülünde bulunmaktadır. Buradaki önemli fonksiyon isimleri şöyledir:

```
tensorflow.keras.losses.mean_squared_error ya da tensorflow.keras.losses.mse  
tensorflow.keras.losses.mean_absolute_error ya da tensorflow.keras.losses.mae  
tensorflow.keras.losses.mean_absolute_percentage_error ya da tensorflow.keras.losses.mape  
tensorflow.keras.losses.mean_squared_logarithmic_error ya da tensorflow.keras.losses.msle  
tensorflow.keras.losses.categorical_crossentropy  
tensorflow.keras.losses.binary_crossentropy
```

Aslında programcının kendisi de bir loss fonksiyonu ya da sınıfı yazıp onu kullanabilmektedir. Kursumuzda bu durum ele alınmayacağından.

compile metodunun optimizer parametresi 'w' değerlerinin iyileştirilmesi için kullanılacak algoritmayı belirtmektedir. Bu algoritma loss fonksiyonuyla karıştırılmamalıdır. Loss fonksiyonu yalnızca bizim hedefe yaklaşma miktarımızı hesaplamakta kullanılmaktadır. Bu hedefe daha iyi yaklaşmak için 'w' değerlerinin nasıl güncellenmesi gerektiği optimizasyon algoritmasıyla ilgilidir. Çok kullanılan üç optimizasyon algoritması "rmsprop (root mean square propagation)", "adam (adaptive moment estimation)" ve "sgd (stochastic gradient descent)" algoritmalarıdır. Optimizasyon algoritmaları compile metodunun optimizer parametresine aşağıdaki isimlerle yazı olarak girilebilir:

```
'rmsprop'
```

```
'sgd'  
'adam'
```

compile metodunun optimizer parametresinin 'rmsprop' default değerini aldığına dikkat ediniz.

Optimizer algoritmaları aslında tensorflow.keras.optimizers modülündeki sınıflar tarafından gerçekleştirilmektedir. Örneğin rmsprop algoritması için RMSProp isimli sınıf, "adam" algoritması için Adam isimli sınıf, "sgd" algoritması için SGD isimli sınıf bulunmaktadır. Böylece biz optimizer parametresine algoritmanın ismini vermek yerine ilgili sınıflar türünden nesne yaratarak bu nesneleri de verebiliriz. Örneğin:

```
from tensorflow.keras.losses import msle  
from tensorflow.keras.optimizers import Adam  
  
model.compile(optimizer=Adam(), loss=msle)
```

Aslında programcının kendisi de optimzier algoritmalarına ilişkin fonksiyon ya da sınıf yazabilmektedir. Ancak kursumuzda bu konuyu ayrıntı olduğu gereklüğüyle ele almayacağız. Biz kursumuzda şimdilik optimizasyon algoritmalarının nasıl işlev gördüğü üzerinde durmayacağız. Bu algoritmaların nasıl çalıştığı kursumuzun ilerleyen bölümlerinde ele alınmaktadır.

compile metodundaki metrics parametresi bir liste biçiminde "metrics" denilen fonksiyonları almaktadır. "metrics" fonksiyonları da "loss" fonksiyonları gibi kestirilen değerlerle gerçek değerler arasındaki farklılığı ifade etmekte kullanılmaktadır. Ancak "loss" fonksiyonlarından eğitim sürecinde ve her batch işlemi bittiğinde (batch işleminin ne anlam ifade ettiği ilerde ele alınmaktadır) faydalananırken "metrics" fonksiyonlarından eğitim ve sınama (validation) sürecinde her epoch bittiğinde (epoch kavramından ilerde bahsedilecektir) faydalanyılmaktadır. Metrics fonksiyonları veri bilimcисine eğitim ve sınama süreçlerinde "eğitimin performansı hakkında bilgi" vermektedir. Loss fonksiyonlarının algoritmanın işleyiş konusunda etkili olduğuna, metrics fonksiyonlarının ise yalnızca bilgi verdiğine dikkat ediniz.

Metrics fonksiyonlarını ele almadan önce eğitim sırasında sınama (validation) işlemleri üzerinde durmak istiyoruz. Modelin sınanması eğitim sırasında modelin başarısını ölçmek için yapılan bir işlemidir. Veri bilimcisi isterse eğitim veri kümesinin bir bölümünün sınama amacıyla kullanılmasını sağlayabilir. Bu durumda eğitim devam ederken aynı zamanda her epoch'tan sonra sınama veri kümesi ile sınama işlemi de yapılır. Sınama işlemini test işlemi ile karıştırmayınız. Sınama işlemi eğitim sırasında test işlemi ise eğitim bittikten sonra yapılmaktadır.

Eğitim ve sınama işlemi sırasında kullanılabilecek çeşitli metrics fonksiyonları bulunmaktadır. Bu fonksiyonlar yukarıda da belirttiğimiz gibi compile metodunun metrics parametresine bir liste biçiminde girilir. Aşağıda önemli metrics fonksiyonları hakkında kısa açıklamalar veriyoruz:

- **accuracy**: Bu metrics fonksiyonu ağıın verdiği değerin yüzde kaçının gerçek değerle aynı olduğu hesabını yapmaktadır. Örneğin eğitim için 1000 adet veri kullanılıyor olsun. Bu 1000 veri ağa sokulduğunda eğer bunların 800 tanesi gerçek değerle aynı değerde ise (tabii belli bir delta aralığı içerisinde) bu durumda accuracy değeri $800 / 1000 = 0.8$ olacaktır. accuracy fonksiyonu lojistik olmayan regresyon modellerinde sıklıkla kullanılmaktadır.

- **binary_accuracy**: accuracy fonksiyonu gibidir. Ancak burada 0 ve 1 biçiminde ikili değerlerin belli bir delta aralığında uyuşmasına bakılmaktadır.

- **binary_crossentropy**: binary_crossentropy loss fonksiyonun metrics biçimidir. Lojistik olmayan iki sınıfı regresyon problemlerinde tercih edilmektedir.

- **categorical_accuracy**: Bu fonksiyon tipik olarak iki sınıfı lojistik regresyon modellerinde kullanılmaktadır. Belli bir delta aralığı içerisinde aynı olduğuna bakmaktadır. Ancak bu fonksiyon ikiden fazla sınıfı lojistik regresyon modellerinde kullanılmaktadır.

- **mean_squared_error**: Lojistik olmayan regresyon problemlerinde metrics değeri olarak sık kullanılmaktadır. mean_squared_error loss fonksiyonu ile aynı işlemleri yapmaktadır. Lojistik olmayan regresyon modellerinde tercih edilmektedir.

- **mean_absolute_error**: Lojistik olmayan regresyon problemlerinde metrics değeri olarak sık kullanılmaktadır. mean_absolute_error loss fonksiyonu ile aynı işlemleri yapmaktadır. Lojistik olmayan regresyon modellerinde tercih edilmektedir.

- **mean_absolute_percentage_error**: mean_absolute_percentage_error loss fonksiyonu ile aynı işlemleri yapmaktadır. Lojistik olmayan regresyon modellerinde tercih edilmektedir.

metrics parametresi için kullanılacak fonksiyon isimleri de şöyledir:

```
'accuracy' ya da 'acc'  
'binary_accuracy'  
'categorical_accuracy'  
'mean_absolute_error' ya da 'mae'  
'mean_absolute_percentage_error' ya da 'mape'  
'mean_squared_error' ya da 'mse'
```

Bu durumda compile metodunun örnek bir çağrı şöyledir:

```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['binary_accuracy'])
```

Aslında metrics fonksiyonları da tensorflow.keras.metrics modülündeki sınıflarla temsil edilmiştir.. Örneğin biz compile metodunda metrics fonksiyonlarını şöyledir:

```
from tensorflow.keras.metrics import BinaryAccuracy  
  
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=[BinaryAccuracy()])
```

Aslında buradaki sınıflar tensorflow.keras.losses modülündeki sınıflarla aynıdır. Dolayısıyla metrics fonksiyonları ve sınıfları yerine loss fonksiyonları ve sınıfları da kullanılabilmektedir. Ancak her loss fonksiyonu ve sınıfı için bir metrics fonksiyonu ve sınıfı, her metrics fonksiyonu ve sınıfı için de bir loss fonksiyonu ve sınıfı yoktur. Örneğin binary_cross_entropy hem bir loss fonksiyonu hem de bir metrics fonksiyonudur. Ancak binary_accuracy bir metrics fonksiyonudur fakat loss fonksiyonu değildir. Bu durumda örneğin biz metrics olarak tensorflow.keras.metrics modülündeki binary_crossentropy fonksiyonu yerine tensorflow.keras.losses modülündeki binary_crossentropy fonksiyonunu kullanabilirdik. Örneğin:

```
from tensorflow.keras.losses import binary_crossentropy  
  
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=[binary_crossentropy])
```

Tabii bu durumun tersi de geçerlidir. Yani loss fonksiyonu ya da sınıfı için biz tensorflow.keras.metrics modülündeki uygun fonksiyonları ve sınıfları da kullanabiliriz.

4) Modelin Eğitilmesi: Sequential.compile metodu ile modelin konfigüre edilmesinden sonra sıra eğitim aşamasına gelmiştir. Yukarıda da belirtildiği gibi eğitim modeldeki nöronların 'w' katsayılarının uygun biçimde ayarlanması sağlayan bir süreçtir. Şüphesiz eğitim veri kümesi ne kadar büyük ve anlamlı ise, kullanılan optimizasyon algoritması ve loss fonksiyonu ne kadar uygun seçilmişse eğitimin sonucunda gerçekleşen öğrenme de o kadar tatmin edici olacaktır. Eğitim işlemi Sequential sınıfının fit isimli metoduyla yapılmaktadır. Metodun parametrik yapısı şöyledir:

```
Sequential.fit(x=None,  
                y=None,  
                batch_size=None,  
                epochs=1,  
                verbose='auto',  
                callbacks=None,
```

```

validation_split=0.0,
validation_data=None,
shuffle=True,
class_weight=None,
sample_weight=None,
initial_epoch=0,
steps_per_epoch=None,
validation_steps=None,
validation_batch_size=None,
validation_freq=1,
max_queue_size=10,
workers=1,
use_multiprocessing=False
)

```

Göründüğü gibi metodun çok sayıda parametresi vardır. Ancak en önemli olan parametreler ilk dört parametre olan x, y, batch_size ve epochs parametreleridir. x ve y gerçek gözlemlerdeki girdi ve çıktı değerlerine ilişkin ndarray nesneleridir.

Eğitim sırasında gözlemler teker teker iterasyona sokulup loss değeri elde edilerek optimizasyon algoritması ile 'w' değerleri güncellenebilir. Ancak bazen on binlerce hatta milyonlarca eğitim verisini böyle tek tek işleme sokmak çok zaman alıbmaktedir ve üstelik de bazı nedenlerden dolayı eğitimin başarısını da düşürebilmektedir. İşte bunun için bir grup gözlem verisini (yani satırı) birlikte işleme sokup bunlardan elde edilen sonuçlardan hareketle 'w' değerlerinin güncellenmesi yoluna gidilmektedir. 'w' değerlerinin her bir gözlem verisinden (satırдан) sonra değil de bir grup gözlem verisi işleme sokulduktan sonra güncellenmesi işlemeye "batch işlemi" buradaki bir grup gözlem verisine de "batch" denilmektedir. İşte fit metodunun batch_size parametresi batch büyüğünün kaç gözlem verisinden (satırдан) oluşacağını belirtmektedir. Her batch işleminden sonra loss fonksiyonu kullanılarak tahmin edilen değerle gerçek değer arasındaki fark hesaplanır ve optimizasyon algoritması çalıştırılarak 'w' değerleri güncellenir. Bu işleme "iterasyon" denilmektedir. Yani "iterasyon" batch_size kadar gözlemin işleme sokup bundan loss değerinin elde edilmesi ve daha sonra optimizasyon algoritması ile w değerlerinin güncellenmesi sürecini belirtmektedir.

fit metodunun epochs parametresi eğitim veri kümесinin baştan aşağı toplamda kaç kez kullanılacağını belirtmektedir. Tabii aynı veri kümесinin aynı model için birden fazla kez kullanılması sizlere tuhaf gelebilir. Ancak ne olursa olsun bu yöntemin 'w' katsayılarını iyileştirmede belli bir faydası vardır. (Aynı dersi birden fazla kez dinlerseniz daha iyi öğrenirsiniz değil mi?) Çoğu zaman epoch değerini yükselttiğimizde öğrenmenin daha başarılı olduğunu görürüz. Ancak belli bir epoch değerinden sonra artık öğrenme başarısı pek artmamaktadır. Ayrıca epoch sayısının yüksek tutulması eğitim süresinin uzamasına ve "overfit" sorununun ortayamasına da yol açabilmektedir.

Yukarıda da belirttiğimiz gibi epoch eğitim veri kümесinin toplamda baştan sona kaç kere eğitim sürecinde kullanılacağını belirtmektedir. Yine yukarıda belirttiğimiz gibi her epoch işleminden sonra sınama (validation) işlemi yapılmaktadır. Sınama işleminde kullanılan veriler fit metod tarafından bizim ona verdığımız training_dataset_x ve training_dataset_y içerisinde alınmaktadır. İşte fit metodundaki validation_split isimli parametre bizim verdığımız verilerin yüzde kaçının sınama için kullanılacağını belirtmektedir. Metot kendi içerisinde bizim ona verdığımız training_dataset_x ve training_dataset_y vektörlerini bu oranda bölüp onun belli bir kısmını epoch işleminden sonra sınama işleminde kullanmaktadır. Programcı isterse sınama işi için veri kümесini kendisi de oluşturabilir. Yani sınama verisinin eğitim verilerinden alınması zorunlu değildir. Bunun için fit metodunun validation_data parametresi kullanılmaktadır. fit metodunda validation_split parametresinin default olarak 0 olduğunu dikkat ediniz. Bu durum eğitim sırasında epoch'lardan sonra hiç sınama yapılmayacağı anlamına gelmektedir. validation_split parametresinin girilmesi durumunda sınama verilerinin seçilmesi işin başında yapılmakta ve her epoch'ta aynı sınama verileri kullanılmaktadır. fit metodunun shuffle parametresinin default olarak True olduğunu dikkat ediniz. Bu durumda her epoch işleminde işin başında ayrılmış olan eğitim verileri kendi aralarında yeniden karıştırılmaktadır.

fit metodunun verbose parametresinin default durumda 'auto' olduğunu görüyorsunuz. Bu parametre eğitim sırasında bilgilendirme yazılarının hangi düzeyde ekrana yazdırılacağını belirtmektedir. Bu parametre aslında 0, 1 ya da 2 değerini almaktadır. 0 değeri fit işlemi sırasında ekrana hiçbir şeyin yazdırılmayacağı anlamına gelir. 1 değeri ekrana bir progress bar çıkartarak ilerlemeyi gösterir. 2 ise her epoch'ta loss ve metrik değerleri ekrana

yazdırmaktadır. Parametrenin default değeri olan 'auto' duruma göre 2 değerine ya da 1 değerine karşılık gelmektedir.

fit metodu geri dönüş değeri olarak History nesnesi vermektedir. Bu nesne içerisinde biz her epoch'taki sonuç bilgilerini alabiliriz. History nesneleri izleyen bölümlerde ele alınmaktadır.

Şimdiye kadar geçtiğimiz aşamalara ilişkin örnek bir model şöyle olabilir:

```
import numpy as np
from sklearn.model_selection import train_test_split

dataset = np.loadtxt('diabetes.csv', delimiter=',', skiprows=1)
training_dataset_x, test_dataset_x, training_dataset_y, test_dataset_y =
train_test_split(dataset[:, :8], dataset[:, 8], test_size=0.2)

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential()

model.add(Dense(100, input_dim=8, activation='relu', name='Hidden-1'))
model.add(Dense(100, activation='relu', name='Hidden-2'))
model.add(Dense(1, activation='sigmoid', name='Output'))

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['binary_accuracy'])
model.fit(training_dataset_x, training_dataset_y, batch_size=32, epochs=100,
validation_split=0.2)
```

Elde edilen sonuç şöyledir:

```
Epoch 1/100
16/16 [=====] - 1s 9ms/step - loss: 1.0918 - binary_accuracy: 0.5927 - val_loss:
0.7977 - val_binary_accuracy: 0.5772
Epoch 2/100
16/16 [=====] - 0s 2ms/step - loss: 0.7048 - binary_accuracy: 0.6477 - val_loss:
0.7097 - val_binary_accuracy: 0.6829
...
Epoch 99/100
16/16 [=====] - 0s 2ms/step - loss: 0.5110 - binary_accuracy: 0.7658 - val_loss:
0.6706 - val_binary_accuracy: 0.6829
Epoch 100/100
16/16 [=====] - 0s 2ms/step - loss: 0.4458 - binary_accuracy: 0.7902 - val_loss:
0.5761 - val_binary_accuracy: 0.7317
```

fit metodu her epoch'tan sonra sınıma verisi üzerinde elde ettiği başarı değerini ekrana yazdırmaktadır. Bu başarı değeri bizim metrics ile verdigimiz algoritmadan elde edilen değerdir. Programcı bu değerlere bakarak modelin başarısı hakkında bir fikir edinebilir. Duruma göre modelini değiştirmek düzeltmeye çalışabilir.

Şimdi modelimize bir tane daha saklı katman ekleyelim. Acaba modelimiz daha başarılı hale gelecek mi? Örneğin:

```
import numpy as np
from sklearn.model_selection import train_test_split

dataset = np.loadtxt('diabetes.csv', delimiter=',', skiprows=1)
training_dataset_x, test_dataset_x, training_dataset_y, test_dataset_y =
train_test_split(dataset[:, :8], dataset[:, 8], test_size=0.2)

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential()
```

```

model.add(Dense(100, input_dim=8, activation='relu', name='Hidden-1'))
model.add(Dense(100, activation='relu', name='Hidden-2'))
model.add(Dense(100, activation='relu', name='Hidden-3'))
model.add(Dense(1, activation='sigmoid', name='Output'))

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['binary_accuracy'])
model.fit(training_dataset_x, training_dataset_y, batch_size=32, epochs=100,
validation_split=0.2)

```

Elde edilen sonuçlar şöyledir:

```

Epoch 1/100
16/16 [=====] - 1s 9ms/step - loss: 2.1612 - binary_accuracy: 0.5316 - val_loss:
0.7898 - val_binary_accuracy: 0.6179
Epoch 2/100
16/16 [=====] - 0s 2ms/step - loss: 1.1615 - binary_accuracy: 0.5866 - val_loss:
0.8797 - val_binary_accuracy: 0.6748
...
Epoch 99/100
16/16 [=====] - 0s 4ms/step - loss: 0.3778 - binary_accuracy: 0.8187 - val_loss:
0.6744 - val_binary_accuracy: 0.7073
Epoch 100/100
16/16 [=====] - 0s 5ms/step - loss: 0.3641 - binary_accuracy: 0.8147 - val_loss:
0.6834 - val_binary_accuracy: 0.7073

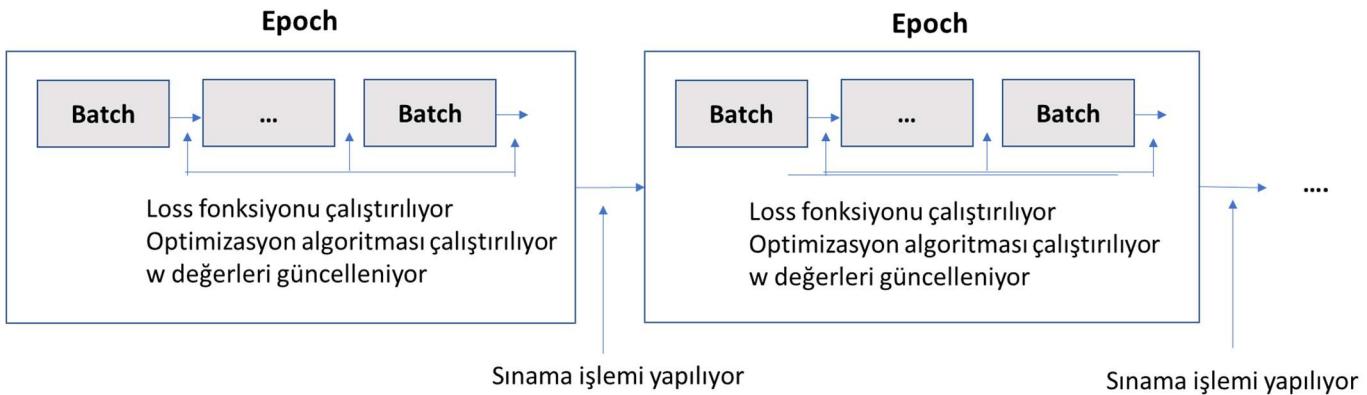
```

İşte her modelde saklı katman sayısının artırılması modeli daha başarılı hale getirmemektedir. Yukarıdaki konularda da dephinildiği gibi pek çok regresyon modelinde iki saklı katman yeterli olmaktadır. Bu katmanların sayısının artırılması işlem süresini uzattığı halde başarıyı artırmayabilir. Ancak bazı tarz problemlerde çok katmanlı derin öğrenme ağları çok iyi sonuçların elde edilmesine yol açabilmektedir

Şimdi bu noktaya kadar ögrenmeklerimizin bir özeti yapalım:

- Toplam veri kümesi başlangıçta eğitim ve test veri kümesi olmak üzere ikiye ayrılmaktadır. Genellikle test veri kümesi toplam veri kümesinin %20'si civarlarında seçilmektedir.
- Eğitim veri kümesi de kendi arasında ikiye ayrılmaktadır. Asıl kısım eğitimde kullanılacak kısımdır. Diğer kısım sınama amacıyla kullanılan kısımdır. Veri kümesinin sınama amacıyla kullanılan kısma "sınama veri kümesi (validation data set)" diyoruz.
- Batch eğitim verilerinin tek tek değil grüplanarak işleme sokulması anlamına gelmektedir. Batch işlemi hem bir yandan eğitim süresini kısaltırken hem de "overfit" durumunu engelleyebilmektedir. Overfit kavramı ileride ele alınacaktır.
- Epoch toplam eğitim verilerinin baştan sona kaç kere eğitimde kullanılacağını belirtmektedir.
- 'w' değerleri her batch işleminden sonra güncellenmektedir.
- Sınama işlemi ise her epoch'tan sonra yapılmaktadır.

Şimdi de son olarak eğitim süresindeki batch ve epoch kavramlarını bir şekilde özetleyelim:



5) Modelin Test Edilmesi: Şimdi artık sıra modelin test veri kümesiyle test edilmesine gelmiştir. Bu işlem Sequential sınıfının evaluate metoduya yapılmaktadır. evaluate metodunun parametrik yapısı şöyledir:

```
Sequential.evaluate(x=None,
                    y=None,
                    batch_size=None,
                    verbose=1,
                    sample_weight=None,
                    steps=None,
                    callbacks=None,
                    max_queue_size=10,
                    workers=1,
                    use_multiprocessing=False,
                    return_dict=False,
                    **kwargs
)
```

Metodun ilk iki parametresi test veri kümesinin girdi ve çıktı matrislerini almaktadır. Test süreci batch işlemleriyle tek bir epoch olarak gerçekleştirilmektedir. Metodun batch parametresi batch büyüklüğünü belirtmektedir. Metot bize test veri kümesinden elde edilen metrics değerlerini bir liste olarak vermektedir. Bu metrics değerleri compile metodunda belirtilen metrics fonksiyonlarının verdiği değerlerdir. (Kullanılan metrics isimleri ayrıca Sequential sınıfının metrics_names örnek özniteliğinden elde edilebilir.) evaluate metodunun geri dönüş değeri olarak verilen listenin ilk elemanı her zaman "loss" değerine diğer elemanları da metrics ile belirtilen değerlere ilişkindir. Örneğin:

```
eval_result = model.evaluate(test_dataset_x, test_dataset_y)
print(eval_result)
```

Örnek çıktı şöyledir:

```
[0.5974747538566589, 0.6623376607894897]
```

Burada listenin ilk elemanı loss fonksiyonundan elde edilen son değerdir. İkinci eleman ise compile işleminde belirttikimiz metrik 'binary_accuracy' metrik değeridir. Yazdırma işlemini şöyle de yapabiliyoruz:

```
for i in range(len(eval_result)):
    print(f'{model.metrics_names[i]}: {eval_result[i]}')
```

Buradan şöyle bir çıktı elde edilecektir:

```
loss: 0.5974747538566589
binary_accuracy: 0.6623376607894897
```

6) Model Üzerinde Kestirimlerin Yapılması: Model test edildikten sonra kullanıma hazır hale gelmiştir. Artık kestirimlerde kullanılabilir. Bizim zaten öğrenen modellerden beklediğimiz öğrenilmiş olanlardan hareketle yeni

durumlar için modelin kestirimlerde bulunabilmesidir. Kestirim işlemi Sequential sınıfının predict isimli metoduyla yapılmaktadır. Metodun parametrik yapısı şöyledir:

```
Sequential.predict(x,
    batch_size=None,
    verbose=0,
    steps=None,
    callbacks=None,
    max_queue_size=10,
    workers=1,
    use_multiprocessing=False
)
```

Metodun birinci parametresi kestirimde bulunulacak girdi verilerini temsil etmektedir. Yani bu parametre girdi katmanındaki nöronlara uygulanacak değerleri belirtmektedir. Bu girdi verisi tek bir vektör olabileceği gibi bir grup vektör de olabilir. batch_size eğer girdi verisi birden fazlaysa yine bunların kaçarlı biçimde işleme sokulacağını belirtir. Metodun çıktısı normal olarak yapay sinir ağının çıktı katmanındaki değerdir. Tabii metoda birden fazla girdi verilirse çıktı da buna göre birden fazla değerden oluşacaktır. predict metodundaki x parametresi bir matris biçiminde organize edilmelidir. Benzer biçimde predict metodunun geri dönüşü değeri de bir matris biçimindedir. Matrisin satırları girdi değerlerine sütunları ise çıktı değerlerine ilişkindir. Örneğin biz predict metoduna n tane gözlemden oluşan n satır girersek ve çıktı tek bir nörondan oluşuyorsa metot bize bize nx1'lik bir matris verecektir. Girdi matrisini şekilsel olarak şöyle gösterebiliriz:

```
X11 X12 X13 ... X1n
X21 X22 X23 ... X2n
X31 X32 X33 ... X3n
X41 X42 X43 ... X4n
X51 X52 X53 ... X5n
```

Burada 5 satırlık bilgi vardır. Her bir satır kestirim için gereken girdileri belirtir. Eğer modelin çıktısı bir tane ise çıktı matrisini de şekilsel olarak şöyle gösterebiliriz:

```
011
021
031
041
051
```

Buradaki her satır bir girdinin tahmin edilen çıktısını belirtmektedir.

Yukarıdaki model için örnek bir kestirim şöyle yapılabilir:

```
output = model.predict(x)
print(output)
if output[0, 0] > 0.5:
    print('diyabetli')
else:
    print('diyabetsiz')
```

Burada biz kişinin 8 biyomedikal özelliğini girdi olarak kullanıp bir çıktı elde ettik. Elde ettiğimiz çıktıının görünümü şöyledir:

```
[[0.8509829]]
diyabetli
```

Buradaki 0. 0.8509829 değeri ağımızın çıktı katmanındaki değerdir. Bu değer sigmoid fonksiyonun elde edildiğini anımsayınız. Halbuki biz bu işlemden ikili (binary) bir değer elde etmek istiyorduk. İşte bu durumda elde ettiğimiz bu değeri bir eşik fonksiyonuna sokmamız gereklidir. Tabii tipik eşik fonksiyonu 0.5'ten büyük değerleri 1 olarak 0.5'ten düşük değerleri 0 olarak veren fonksiyondur.

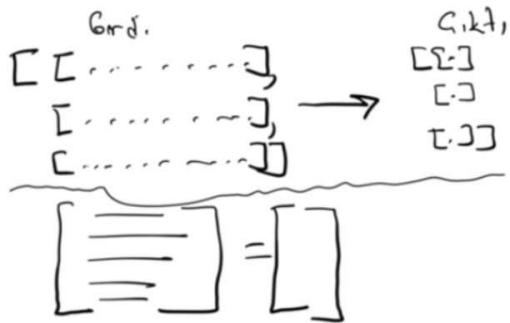
Yukarıda da belirttiğimiz gibi predict işleminin teker teker yapılmasına gerek yoktur. Biz predict metodunda bir matris de verebiliriz. Örneğin:

```
x = np.array([[1, 1, 48, 20, 0, 24.7, 0.90, 22],  
             [1, 2, 48, 30, 0, 24.7, 0.92, 22],  
             [2, 3, 48, 30, 0, 24.7, 0.94, 22]])  
  
output = model.predict(x)  
print(output)  
  
for i in range(len(output)):  
    if output[i, 0] > 0.5:  
        print('diyabetli')  
    else:  
        print('diyabetsiz')
```

Bu koddan şöyle bir çıktı elde edilmiştir:

```
[0.6931946873664856, 0.701298713684082]  
[[0.80758095]  
[0.36790228]  
[0.48114637]]  
diyabetli  
diyabetsiz  
diyabetsiz
```

Burada predict metoduna üç satırı bir matris argüman olarak verilmiştir. Yani biz tek hamlede üç farklı gözlem sonucunu tahmin etmek istemekteyiz. O halde predict metodu da bize üç elemanlı bir sütun matrisi verecektir. Bu durumu şekilsel olarak aşağıdaki gibi gösterebiliriz.



Peki predict metodundaki batch_size parametresinin anlamı nedir? Anımsanacağı gibi fit işleminde batch_size bir grup satırın tek bir satırı gibi işleme sokulması anlamına geliyordu. Yani fit işleminde 'w' değerlerinin güncellenmesi her satırda değil her batch_size kadar satırda bir yapılmaktadır. Ancak predict metodundaki batch_size farklı bir anlamdadır. Bu metottaki batch_size predict işlemini aynı anda kaçarlı gruplar halinde yapılacağını belirtmektedir. Yani buradaki batch_size değerinin predict sonucu üzerinde hiçbir etkisi yoktur. Başka bir deyişle biz buraki batch_size değerini 32 versek de (default durum) 1 versek de tamamen aynı sonucu elde ederiz. O halde bu parametrenin ne faydası vardır? Bu parametre içsel olarak Keras'in predict işlemini hızlı yapması konusunda faydalı olabilmektedir.

Keras'ta Kullanılan Veri Kümeleri ve Algoritmalarla İlişkin Kavramların Özeti

Temel olarak bir sinir ağı eğitilirken üç veri kümesi söz konusudur: Eğitim Veri Kümesi (Training Data Set), Sınamma Veri Kümesi (Validation Data Set) ve Test Veri Kümesi (Test Data Set). Bunların oranları uygulamadan uygulamaya değişimle de genellikle test veri kümesi eğitim veri kümesinin %20'si, sınavma veri kümesi de eğitim veri kümesinin %20'si olarak alınmaktadır. Eğitim veri kümesi -ismi üzerinde- eğitim sürecinde kullanılan veri kümesidir. Sınamma veri kümesi ise eğitim sürecinde her epoch işleminden sonra yapılması gereken kullanılmaktadır. (Sınamma veri kümesinin her batch size'dan sonra değil her epoch'tan sonra uygulandığına dikkat ediniz.) Test veri kümesi tüm eğitim bittikten

sonra model test edilirken test için kullanılan veri kümesidir. Pekiyi bu durumda sınama veri kümesi ile test veri kümesi arasında ya da sınama işlemi ile test işlemi arasında ne fark vardır? Sınama veri kümesi her epoch işleminden sonra uygulandığı için modelin epoch işlemlerine göre davranışları gözlenebilmektedir. Halbuki test veri kümesi tüm model eğitildikten sonra performans ölçütlerini belirlemek için kullanılır.

Model eğitilirken her batch size kadar eğitim veri kümesi bir araya getirilerek eğitimde kullanılır. Yani eğitim işlemi birer birer değil çanak çanak (batch batch) yapılmaktadır. Loss fonksiyonunun hesaplanması ve 'w' katsayılarının optimizasyon algoritmasına göre güncellenmesi her batch işleminden sonra yapılmaktadır. Aslında Keras'ta her batch işleminde de callback mekanizmasıyla araya girmek mümkündür

Loss fonksiyonu 'w' değerlerinin güncellenmesi için bir amaç fonksiyonu olarak kullanılmaktadır. Optimizasyon algoritması ise loss fonksiyonuna bakılarak 'w' değerlerinin nasıl güncelleneceğini belirleyen algoritmadır. Yani başka bir deyişle "optimizasyon algoritması loss fonksiyonunu minimize etmek için 'w' değerlerinin nasıl değiştirileceğini belirten algoritmadır."

Pekiyi metrik fonksiyonlar ne anlama gelmektedir? Metrik değerler sınama süreci ile ilgilidir. Genellikle metrik fonksiyonlar loss fonksiyonıyla aynı olabilir. Ancak eğitimde birden fazla metrik fonksiyon da kullanılabilmektektir. Pekiyi loss fonksiyonu ile metrik fonksiyonları arasında ne fark vardır? İşte loss fonksiyonu optimizasyon algoritmasını uygulamak için bir hedef belirtirken metrik fonksiyonları ise doğrulama işlemi sonucunda bir performs belirtmektedir.

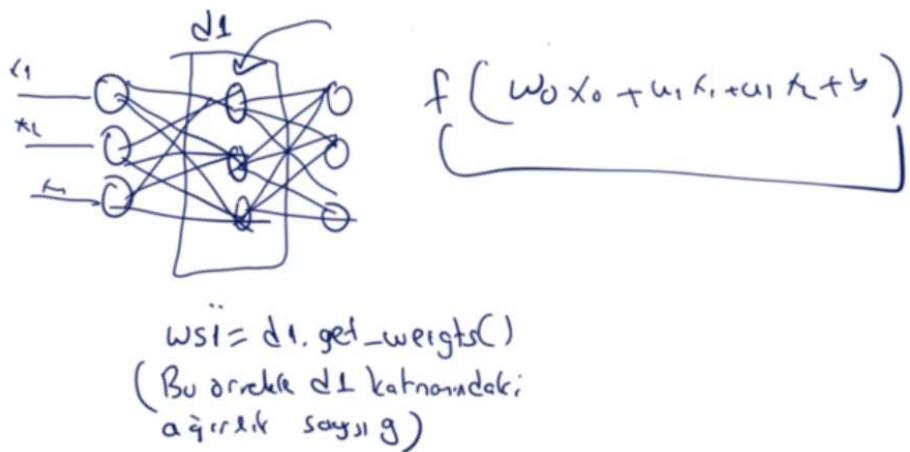
Batch İşleminin Anlamı

Eğitim ve test sırasında girdi kümelerinin birer birer değil bir grup olarak işleme sokulduğunu belirtmiştık. Bu grubun büyüğüğünne "batch size" denilmektedir. Pekiyi girdilerin teker teker eğitime sokulmayıp grup grup sokulmasının anlamı nedir? Batch işleminin genel olarak bir hızlandırma sağladığı söylenebilir. Batch işleminin yapılış biçimini kullanılan optimizasyon ve loss fonksiyonuna göre değişimle birlikte tipik uygulanışı şöyledir: Biz batch büyüğüğünü (batch size) 32 kabul edelim. Bu durumda tipik olarak 32'lik bir kümenin elemanları (bu elemanlara "örnek (sample)" da denilmektedir) tek tek ve ayrı ayrı ağa sokulup buradan bir loss değeri hesaplanır. Sonra bu loss değerinin batch için ortalaması alınır (örneğimizde bu değer 32'ye bölünecektir.) Bu ortalama değer tek bir değer olarak optimizasyon algoritmasına verilmektedir. Böylece 'w' değerlerini güncellenmesi birer birer değil batch-batch yapılır. 'w' değerlerinin böyle batch batch hesaplanması genel olarak hesaplamayı hızlandırmaktadır. Batch işlemleri aynı zamanda overfit olgusuna karşı bir direnç de oluşturabilmektedir. Overfit olgusu sonraki bölümlerde ele alınmaktadır.

Tabii batch içerisindeki bir grup veri aslında Python'da bir hamlede işleme sokulur. Yani bunun için bir döngü gerekmemektedir. Zaten numpy kütüphanesi bu işlemi yapmaktadır. Pekiyi biz her ne kadar numpy'da iki vektörü tek hamlede işleme sokabiliyor olsak da arka planda numpy bu işlemi işlemciye bir döngü içerisinde sıralı olarak mı yaptırmaktadır? İşte numpy arka planda C'de yazılmış bir kütüphanedir. İşlemcilerin vektörel işlem yapma yeteneğini de kullanmaktadır. (Örneğin Intel işlemcileri ve ARM işlemcileri böyle vektörel işlemler yapabilmektedir.)

Keras Modelindeki Nöron Ağırlıklarının (W Değerlerinin) Elde Edilmesi ve Yüklenmesi

Keras'ta eğitim sonrasında nöronlarda oluşan ağırlık değerlerini almak isteyebiliriz. Keras model olarak bu bilgileri katman nesnesi ile ilişkilendirmiştir. Dolayısıyla bu bilgiler eğitim sonrasında katman nesnelerinden elde edilebilmektedir. İşte Dense sınıfının (ve diğer katman sınıflarının) `get_weights` ve `set_weights` metodları bu ağırlıkların alınması ve yüklenmesi için kullanılmaktadır. Tabii her katman kendi ağırlık değerlerini bize verecektir.



Bir katmandaki nöron ağırlıkları o katmandaki nöronlardan çıkan değil o katmandaki nöronlara giren ağırlık değerleridir.

Mademki nöron ağırlıkları katman nesnelerinin içerisindeidir o halde bizim katmanlardaki nöronların ağırlıklarını elde edebilmemiz için yaratmış olduğumuz katman nesnelerine erişebilmemiz gereklidir. Halbuki biz yukarıda Dense katman nesnelerini yaratır yaratmaz hemen Sequential sınıfının add metoduyla modele ekledik. Yani katman nesnelerini bir değişkende tutmadık. Yukarıdaki örneğin ilgili kısmını anımsayınız:

```
model = Sequential()
model.add(Dense(100, input_dim=8, activation='relu', name='Hidden-1'))
model.add(Dense(100, activation='relu', name='Hidden-2'))
model.add(Dense(1, activation='sigmoid', name='Output'))
...
```

Katman nesnelerini elde edebilmenin bir yolu add metodunu uygulamadan önce onları değişkenlerde saklamak olabilir. Örneğin:

```
model = Sequential()
d1 = Dense(100, input_dim=8, activation='relu', name='Hidden-1')
model.add(d1)
d2 = Dense(100, activation='relu', name='Hidden-2')
model.add(d2)
d3 = Dense(1, activation='sigmoid', name='Output')
model.add(d3)
...
wd1 = d1.get_weights()
wd2 = d2.get_weights()
wd3 = d3.get_weights()
```

Aslında bu yönteme gerek yoktur. Çünkü Sequential sınıfının zaten layers isimli örnek özniteliği bize katman nesnelerini bir liste biçiminde vermektedir. Bu durumda örneğin biz modelimizin katmanlarına ilişkin ağırlık değerlerini şöyle elde edebiliriz:

```
wd1 = model.layers[0].get_weights()
wd2 = model.layers[1].get_weights()
wd3 = model.layers[2].get_weights()
```

get_weights metodu bize duruma göre iki elemanlı ya da tek elemanlı bir numpy dizi listesi vermektedir. Şöyle ki: Eğer katman yaratılırken Dense fonksiyonunda bias=False değeri geçilirse ilgili katmandaki nöronlarda bias değerleri kullanılmayacaktır. Bu durumda get_weights metodu da yalnızca ağırlık matrisinden oluşan bir elemanlı liste verecektir. Eğer katman yaratılırken Dense fonksiyonunda bias=True değeri geçilirse (default durum zaten böyledir) artık get_weights fonksiyonu iki elemanlı bir listeye geri dönecektir. Bu durumda listenin ilk elemanı ağırlık matrisinden diğer elemanı bias değerlerinden oluşacaktır. Anımsanacağı gibi bias değeri her nöron için oluşturulan

bir değerdir. Dolayısıyla örneğin eğer katmanımızda 100 nöron varsa 100 tane bias değeri olacaktır. Bu bias değerlerinin ne işe yaradığı ileride ele alınacaktır. Ancak bu bias değerleri de eğitilmiş modelin bir parçasını oluşturmaktadır.

get_weights metodlarının verdiği ağırlık değerleri önceki katmanın çıkışlarının ilgili katmanın girişlerine bağlılığı ağırlık değerleridir. Örneğin get_weights metodunun çağrıldığı katmanda 100 nöron olsun. Önceki katmanda da 8 nöron olsun. Bu durumda get_weights bize 8×100 'luk bir ağırlık matrisi verecektir. Bu matrisin satırları önceki katman nöronlarına sütunları ise ilgili katmanın nöronlarına ilişkindir.

Yukarıdaki örnek modeldeki d1 katmanından (birinci saklı katmandan) elde edilen ağırlık değerlerine ilişkin wd1 dizisinin eleman türlerine ve uzunluklarına dikkat ediniz:

```
In [93]: wd1[0].shape  
Out[93]: (8, 100)
```

```
In [94]: wd1[1].shape  
Out[94]: (100,)
```

```
In [95]: type(wd1)  
Out[95]: list
```

```
In [96]: len(wd1)  
Out[96]: 2
```

```
In [97]: type(wd1[0])  
Out[97]: numpy.ndarray
```

```
In [98]: wd1[0].shape  
Out[98]: (8, 100)
```

```
In [99]: type(wd1[1])  
Out[99]: numpy.ndarray
```

```
In [100]: wd1[1].shape  
Out[100]: (100,)
```

Listenin 0'inci indeksli elemanı o katmandaki nöronların ağırlıklarını vermektedir. Bu katmanda 8 girdi nöronu dense bir biçimde katmandaki 100 nörona bağlandığına göre toplamda her biri 8 elemandan oluşan 100 ağırlık değer olacaktır. Yine bu katmandaki bias değerlerinin nöron sayısı olan 100 tane olduğuna dikkat ediniz. Pekiyi şimdi de modelin ikinci saklı katmanına ilişkin ağırlık değerlerinin kaçar tane olduğuna bakalım. İkinci saklı katmandaki nöronun her birine toplam 100 nöron girecektir. Bu durumda ikinci saklı katmandaki ağırlık değerleri (100, 100) biçiminde bir matris ile ifade edilecektir. Bu matrisin 10000 elemanı olduğuna dikkat ediniz. Bu katmandaki toplam bias değerleri 100 tane (katmandaki nöron sayısı) olacaktır:

```
In [110]: wd2[0].shape  
Out[110]: (100, 100)
```

```
In [111]: wd2[1].shape  
Out[111]: (100,)
```

Şimdi de çıktı katmanındaki ağırlık değerlerinin hangi sayıda olması gerektiğine bakalım. Çıktı katmanının nöron sayısı 1 tanedir. Önceki katmandaki 100 nöron bu katmandaki bu tek nörona girdi yapılmıştır. O halde çıktı katmanındaki ağırlık matrisinin (100, 1) olması gerekir. Çıktı katmanı tek nörondan oluştuğuna göre toplamda tek bir bias değeri vardır:

```
In [116]: wd3[0].shape
```

```
Out[116]: (100, 1)
```

```
In [117]: wd3[1].shape
```

```
Out[117]: (1,)
```

set_weights metodu da benzer biçimde ağırlık değerlerini ilgili katmana yüklemek için kullanılmaktadır. Yani örneğin aşağıdaki gibi biz katmandaki ağırlık değerlerini get_weights metodu ile alıp set_weights metodu ile yükleyebiliriz:

```
wd1 = model.layers[0].get_weights()  
...  
model.layers[0].set_weights(wd1)
```

set_weights metodunun da iki elemanlı uygun uzunluklarda indeksli dolaşılabilir nesne aldığına dikkat ediniz.

Keras Modelinin Özeti Bilgisinin Görüntülenmesi

Bir Keras modeli oluşturulduktan sonra model sınıflarının summary isimli metodları bize model hakkında özeti bilgiler vermektedir. Örneğin aşağıdaki gibi bir model söz konusu olsun:

```
import numpy as np  
  
from tensorflow.keras import Sequential  
from tensorflow.keras.layers import Dense  
from sklearn.model_selection import train_test_split  
  
dataset = np.loadtxt('diabetes.csv', skiprows=1, delimiter=',')  
dataset_x = dataset[:, :-1]  
dataset_y = dataset[:, -1]  
training_dataset_x, test_dataset_x, training_dataset_y, test_dataset_y =  
train_test_split(dataset_x, dataset_y,  
test_size=0.20)  
  
model = Sequential()  
model.add(Dense(100, input_dim=8, activation='relu', name='Hidden-1'))  
model.add(Dense(100, activation='relu', name='Hidden-2'))  
model.add(Dense(1, activation='sigmoid', name='Output'))  
  
model.summary()
```

Elde edilen summary bilgisi şöyledir:

Layer (type)	Output Shape	Param #
=====		
Hidden-1 (Dense)	(None, 100)	900
Hidden-2 (Dense)	(None, 100)	10100
Output (Dense)	(None, 1)	101
=====		
Total params: 11,101		
Trainable params: 11,101		
Non-trainable params: 0		

Burada birinci sütunda katmanın ismini görüyorsunuz. İkinci sütunda ilgili katmanın çıkışındaki nöron miktarları belirtilmektedir. Nihayet son sütunda her katmandaki tahmin edilecek parametre sayılarını görmektesiniz. Birinci

katmandaki parametre sayısının $8 * 100 + 100 = 900$, ikinci katmandaki parametre sayısının $100 * 100 + 100 = 10100$ ve üçüncü katmandaki parametre sayısının ise $100 * 1 + 1 = 101$ olduğuna dikkat ediniz.

Oluşturulan Modelin ve Modeldeki Tüm Ağırlık Değerlerinin Bir Dosyada Saklanması ve Geri Yüklenmesi

Bir modeli eğittikten sonra bilgisayarın kapadığımızda tüm eğitim bilgileri kaybolacaktır. İşte onun başka zaman kullanılabilmesi için bir dosyada saklanması gerekebilir. Keras bu işlemler için HDF (Hierarchical Data Format) dosya formatını kullanmaktadır. Sequential sınıfının save isimli metodu bizden .h5 ya da .hdf5 uzantılı HDF dosyasının yol ifadesini alır; model bilgilerini, ağırlık ve bias değerlerini bu dosyaya save eder. Örneğin:

```
model.save('diabetes.h5')
```

Artı biz tüm modeli her şeyle bir dosya içerisinde saklamış olduk. Onu geri almak için tensorflow.keras.models modülündeki load_model fonksiyonu kullanılmaktadır. Bu fonksiyon da benzer biçimde HDF dosyasının yol ifadesini argüman olarak alır. Tüm modeli ağırlık ve bias değerleriyle yeniden yükler, yeni bir model nesnesi yaratarak bize onu verir. Örneğin:

```
from tensorflow.keras.models import load_model  
  
model = load_model('diabetes.h5')
```

Şimdi yukarıdaki örneği save edip tekrar yükleyerek predict işlemi yapalım:

```
# keras-model-save.py  
  
import numpy as np  
  
from tensorflow.keras import Sequential  
from tensorflow.keras.layers import Dense  
from sklearn.model_selection import train_test_split  
  
dataset = np.loadtxt('diabetes.csv', skiprows=1, delimiter=',')  
dataset_x = dataset[:, :-1]  
dataset_y = dataset[:, -1]  
training_dataset_x, test_dataset_x, training_dataset_y, test_dataset_y =  
train_test_split(dataset_x, dataset_y,  
test_size=0.20)  
  
model = Sequential()  
model.add(Dense(100, input_dim=8, activation='relu', name='Hidden-1'))  
model.add(Dense(100, activation='relu', name='Hidden-2'))  
model.add(Dense(1, activation='sigmoid', name='Output'))  
  
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['binary_accuracy'])  
  
model.fit(training_dataset_x, training_dataset_y, batch_size=32, epochs=200,  
validation_split=0.2)  
result = model.evaluate(test_dataset_x, test_dataset_y)  
  
for i in range(len(result)):  
    print('{} = {}'.format(model.metrics_names[i], result[i]))  
  
model.save('diabetes.h5')
```

```
# keras-model-Load.py
```

```
import numpy as np  
from tensorflow.keras.models import load_model  
  
model = load_model('diabetes.h5')
```

```

predict_x = np.array([[5.0, 73.0, 60.0, 0., 0., 29.8, 0.368, 25.0],
                     [6.0, 43.0, 70.0, 0., 0., 29.8, 0.368, 25.0],
                     [6.0, 73.0, 70.0, 0., 0., 39.8, 0.668, 25.0]])

predict_result = model.predict(predict_x)

for i in range(len(predict_result)):
    if result[i, 0] < 0.5:
        print('Şeker hastası değil')
    else:
        print('Şeker hastası')

```

Bazen programcı yalnızca ağırlık değerlerini (tabii bias değerleri de dahil olmak üzere) saklayıp geri yüklemek isteyebilir. Bu işlem Sequential sınıfının `save_weights` ve `load_weights` metodlarıyla yapılmaktadır. Örneğin:

```

ws = model.save_weights('diabtes_weights.h5')
...
model.load_weights('diabtes_weights.h5')

```

Tabii bu durumda biz bu ağırlık değerlerini kullanabilmek için modeli yeniden oluşturmalıyız. Çünkü `save_weights` model bilgilerini saklamamaktadır. Yalnızca modeldeki tüm nöronların ağırlık değerlerini saklamaktadır.

İstenirse `load` işlemi yapılırken yalnızca aynı isimli katmanların ağırlıkları da yüklenebilmektedir. Şöyle ki: Dense fonksiyonunda name isimli parametre yoluyla katmanlara tek olan (unique) isimler verebilmekteyiz. İşte Sequential sınıfının `load_weights` isimli metodunun da `by_name` isimli bool değer alan bir parametresi vardır. Bu parametre default durumda `False` biçimdedir. Eğer bu parametre `True` yapılrsa bu durumda diskte saklanmış olan modeldeki katman isimleriyle hali hazırda çalışmakta olan model nesnesindeki katmanların isimleri karşılaştırılır. Aynı isimli olan katmandaki ağırlık ve bias değerleri yüklenir. Örneğin:

```
model.load_weights('diabtes_weights.h5', by_name=True)
```

Burada diskte 'x' ve 'y' isimli iki katman olsun. Şu andaki model nesnesinde de bu isimli iki katmanın bulunduğu varsayıyalım. Artık `load_weights` metodu yalnızca bu katmanların ağırlık ve bias değerlerini mevcut modeldekilerin üzerine yükleyecektir.

Bir modeli diske save ettikten sonra onu tekrar yükleyerek eğitime devam edebiliriz. Yani eğitimin (fit metodunun) yalnızca bir kez aynı zaman dilimi içerisinde yapılması gerekmektedir.

Yapay Sinir Ağlarını Kullanarak Kestirim Yapmak İçin Aşamalar

Aslında yapay sinir ağları bir regresyon modeli oluşturmaktadır. Yani biz yapay sinir ağının eğitilmesi sonucunda $y = f(x)$ gibi bir fonksiyon elde elde ederiz. Sonra bu fonksiyona x değerini verdigimizde fonksiyon da bize kestirilen y değerini verir. Tabii yapay sinir ağlarının dışında istatistikte de regresyon amacıyla kullanılan çok sayıda istatistiksel yöntem vardır. Bu yöntemler ilerde ele alınacaktır. Ancak yapay sinir ağları çok sayıda özelliğin (feature) olduğu ve özellikler arasında karmaşık etkileşimlerin bulunduğu durumlarda diğer yöntemlere göre çok daha iyi sonuçlar vermektedir.

Yapay sinir ağlarının kullanılması için tipik aşamalar şunlardır:

1) Hedefin Belirlenmesi Süreci: Önce bizim yapay sinir ağını neden oluşturmak istediğimizi biliyor olmamız gereklidir. Yani bizim ağı oluşturmaktaki amacımız nedir? Biz ağı kullanarak neyi kestirmeye çalışmak istemekteyiz? Örneğin bir dershanedeki öğrenciler arasında üniversite sınavında belli bir puanın yukarısında puan elde edebilecek kişileri önceden tespit etmem gereklidir. Bu öğrencilere özel bazı olanakların sunulması söz konusu olabilir ve kurum da bu olanakları sunacağı öğrencileri mümkün olduğu kadar önce tespit etmek isteyebilir. Ya da örneğin bir kurumun çeşitli mağazaları olabilir ve yönetim belli bir mağazadaki dönemsel ciroyu tahmin etmek isteyebilir. Bunun sonucu olarak kurum bazı politikaları daha uygun belirleyebilecektir.

Amaç kabaca tespit edildikten sonra bunun daha ayrıntılı ve operasyonel hale getirilmesi gerekmektedir. Yani kaba bir amaçtan ziyade giridisi çıktıları belli değerlerle temsil edilen bir amaç haline dönüştürülmelidir.

2) Kestirimle İlgili Olabilecek Özelliklerin (Features) Belirlenmesi Süreci: Veri bilimcisinin kestirimde bulunacağı olguya etkileyen etmenleri (features) bir biçimde tespit etmiş olması gerekmektedir. Biz bunu önceki bölümlerde "özellik seçimi (feature selection)" başlığı altında kısaca ele almıştık. Kestirimle ilgili olabilecek özelliklerin neler olabileceğiının belirlenmesi sürecin önemli adımlarından biridir. Eğer özellikler yanlış ya da yetersiz seçilirse öğrenme bizim istediğimiz ölçüde olamayacaktır. Örneğin bir taşıtin kilometre başına yaktığı yakıtın tahmin edilmesinde rol oynayan özelliklerin bazıları şunlar olabilir:

- Aracın motor hacmi
- Aracın ağırlığı
- Aracın benzinle mi motorinle mi gazla mı çalıştığı
- Aracın yaşı

Biz şimdi burada aracın motor hacmini bir özellik olarak ağımızda dahil etmezsek ağımız ne kadar doğru kestirimlerde bulunabilir ki?

Örneğin bir öğrencinin başarısına etki eden faktörler neler olabilir?

- Öğrencinin zeka düzeyi
- Öğrencinin derslere devam oranı
- Öğrencinin geçmişteki not ortalaması
- Öğrencinin ailesinin ekonomik durumu
- Öğrencinin ailesinin eğitim düzeyi
- Öğrencideki sağlık problemleri
- Ailedeki sağlık problemleri
- Öğrencinin sınıf dışında toplam günlük ortalama çalışma zamanı
- Öğrencinin daha önce nereden geldiği
- Öğrencinin hobilerine ayırdığı zaman
- Öğrencinin anne babasının ayrı olup olmadığı
- Öğrencinin okul ile evi arasındaki uzaklık

Bir eczandanın cırosunu etkileyebilecek özellikler de şunlar olabilir:

- Eczanenin önünden günlük ortalama geçen insan sayısı
- Eczanenin bulunduğu bölge (kategorik)
- Eczanenin yakınılarında bir ya da birden fazla sağlık kurumunun olup olmadığı bilgisi
- Eczanenin hangi kurumlarla anlaşmalı olup olmadığı
- Eczanenin yakınındaki eczane sayısı
- Eczanenin güzellik ürünü satıp satmadığı (kategorik)
- Eczanenin takviye ürünler satıp satmadığı (kategorik)
- Eczanenin büyülüğu

Bu özellikler belirlendikten sonra onların mümkün olduğu kadar operasyonel hale getirilmeleri gerekmektedir. Operasyonel demek bir özelliği ölçülebilir bir biçimde ifade etmek demektir. Örneğin bir mağazanın albenisi operasyonel bir özellik değildir. Ancak bunu likert tarzı bir ölçekte operasyonel hale getirebiliriz.

Burada önemli bir noktayı vurgulamak istiyoruz. Özelliklerin belirlenmesi uzmanlık gerektiren bir alandır. Eğer veri bilimcisi konu ile ilgili uzman bilgisine sahip değilse bu aşamada mutlaka uzmanlardan yardım alması gerekebilir. Biz uzmanı olmadığımız konulardaki özelliklerini isabetli bir biçimde belirleyemeyebiliriz.

3) Eğitim İçin Verilerin Toplanması Süreci: Veri bilimcisinin eğitimde kullanacağı geçmiş birtakım verileri ve bunlara karşı gelen gerçek değerleri (training dataset_x ve training dataset_y) bir biçimde elde etmiş olması gerekmektedir. Biz kursumuzda genellikle zaten elde edilmiş olan hazır veriler üzerinde çalışacağız. İyi de bu veriler kimler tarafından

ve nasıl elde edilmiştir? İşte veriler değişik kaynaklardan elde edilmiş olabilirler. Örneğin bazı veriler zaten elde edilmiş ve veritabanlarında bulunuyor olabilir. Bazı veriler birtakım kurumlardaki (örneğin Türkiye İstatistik Kurumu gibi) veritabanlarından elde edilebilir. Bazen verileri biz anketler yoluyla elde etmek zorunda kalabiliz. Bazı veriler ise sensörler yoluyla otomatik bir biçimde elde ediliyor olabilir. Özellikle son 20 yıldır sensörler yoluyla verilerin elde edilmesi konusunda çok ilerlemeler kaydedilmiştir. IOT uygulamalarının yaygınlaşmasıyla otomatik veri elde etme çok yaygınlaşmıştır. Örneğin biz belli bir kavşakta hangi günün hangi saatinde ne kadar aracın bulunuyor olabileceğini tahmin etmeye çalışalım. Burada verileri kavşağa yerlestireceğimiz sensörler yoluyla elde edebiliriz. (Artık dünyada adaptif kavşak yönetim sistemlerinde bu tür sensörler yaygın olarak kullanılmaktadır.)

4) Verilerin Kullanıma Hazır Hale Getirilmesi Süreci: Veriler toplandıktan sonra bunların önişleme sokulması gerekebilmiştir. Biz daha önce bu konuda özet bilgiler vermişizdir. Önişlem sırasında geçersiz verilerin atılması ya da düzeltmesi, kategorik verilerin sayısallaştırılması, gereksiz ve yüksek korelasyonlu sütunların atılması en çok uygulanan yöntemlerdendir.

5) Yapay Sinir Ağı Modelinin Kurulması Süreci: Artık sıra sinir ağı yapay modelinin kurulmasına gelmiştir. Ağımızın mimarisi nasıl olacaktır? Ağımızda kaç saklı katman bulunacaktır? Katmanlardaki aktivasyon fonksiyonları, ağ için kullanılacak loss fonksiyonu ve optimizasyon fonksiyonu nasıl olacaktır? Bu aşamada bu belirlemeler yapılmalıdır.

6) Modelin Eğitilmesi ve Test Edilmesi Süreci: Bundan sonra model gerçek verilerle eğitilir ve test edilir. Artık elimizde kesitirmde kullanılabilecek ve başarısı hakkında fikrimiz olan bir model vardır.

7) Kestirimde Bulunma Süreci: Artık ağımız hazır olduğuna göre istediğimiz zaman kestirimde bulunabiliriz.

Keras Modelinde Callback Mekanızması

Keras kütüphanesinde bazı işlevlerde programcının bilgilendirilmesi için "callback" denilen bir mekanizma bulundurulmuştur. Bu mekanizma sayesinde fit, evaluate ve predict metodları işlevlerini yaparken programcının belirlediği kodları çalıştırılabilir. fit, evaluate ve predict metodlarının callbacs isimli parametreleri vardır. Bu parametreye callback görevini yapacak bir fonksiyon ya da sınıf nesnesi (callable bir nesne) yerleştirilir. Bazı callback sınıfları zaten hazır durumdadır. Bu nedenle programcının kendi callback sınıflarını yazaması genellikle gerekmemektedir. Ancak programcı isterse kendi callback sınıflarını da yazabilir. Biz burada önce hazır birkaç callback sınıfını ele alacağız sonra da callback sınıflarının nasıl yazılacağı hakkında bir örnek vereceğiz. Diğer callback sınıfları için Keras dokümanlarına başvurulabilir. Keras'taki tüm callback sınıfları tensorflow.keras.callbacks modülündeki Callback isimli sınıfından türetilmiş durumdadır.

History isimli callback sınıfı kayıt işlemi yapmaktadır. Yani örneğin fit metodu çalışırken bu callback sınıfı bizim için bazı kayıtlar tutar. fit metodunun çalışması bittikten sonra d biz bu kayıtları alıp kullanabiliriz. Aslında fit metodunun callbacks parametresine biz History callback sınıfı nesnesini geçirmek zorunda değiliz. fit metodu zaten her zaman bu kaydı yaparak bize bu History nesnesini geri dönüş değeri olarak vermektedir. Biz de yapılan kayıtları fit metodunun geri dönüş değerinden alarak kullanabiliriz. Örneğin:

```
hist = model.fit(training_set_x, training_set_y, batch_size=30, epochs=100)
```

History nesnesinin history isimli örnek özniteliği dict türündendir. Bu sözlük nesnesinin anahtarları "elde edilecek bilginin isimlerini", değerleri de her epoch için ilgili değerleri bize vermektedir. history örnek özniteliği ile biz verilen sözlükte her zaman 'loss' isimli bir anahtar bulunmaktadır. Bu sözlüğün diğer anahtarları ise bizim fit metodunda metrics'te verdığımız metrik isimleridir. Örneğin yukarıdaki model şöyle konfigüre edilmiş olsun:

```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['binary_accuracy'])
```

Bu durumda history nesnesinin keys metodunu bize üç anahtar verecektir:

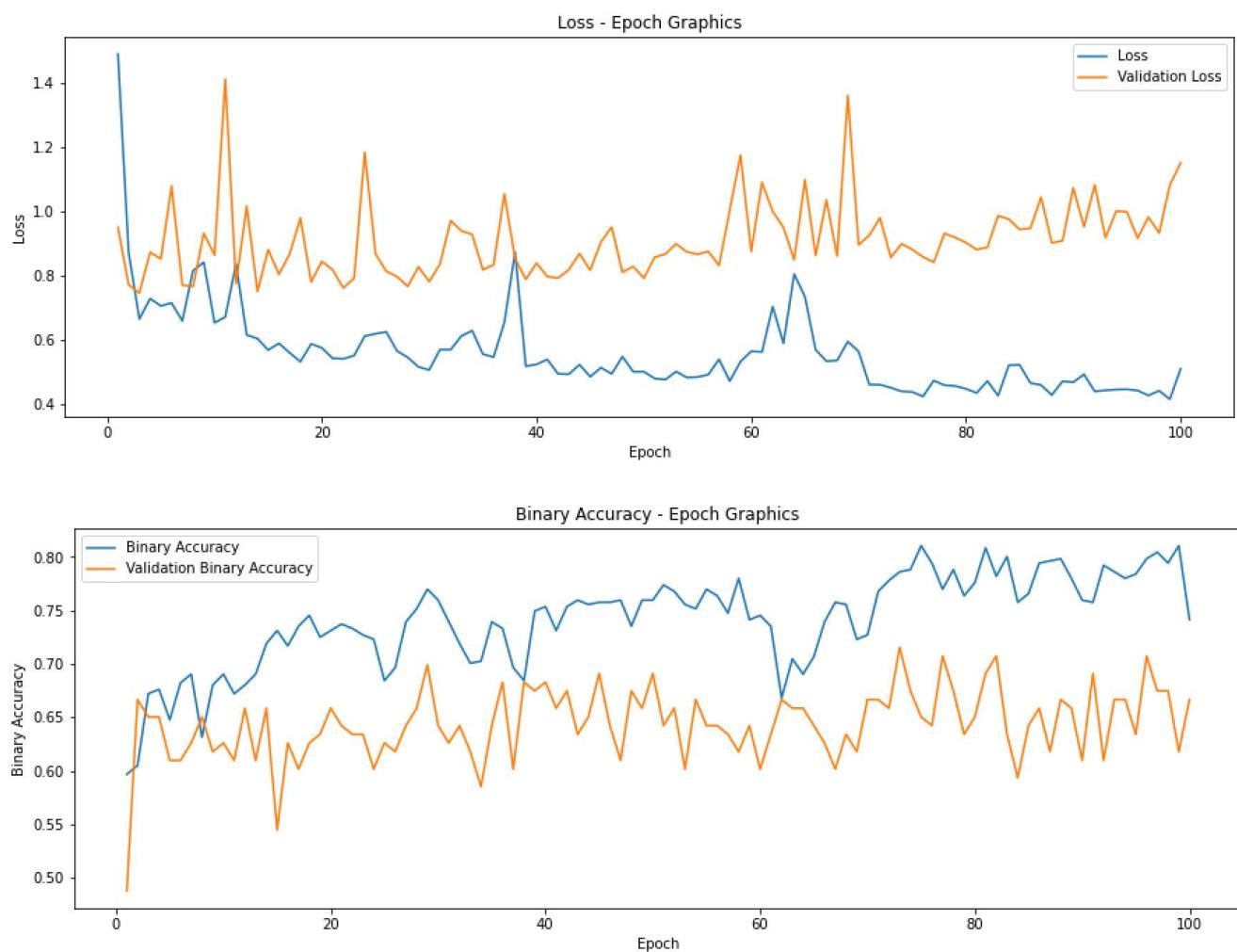
```
In [38]: hist.history.keys()
Out[38]: dict_keys(['loss', 'binary_accuracy', 'val_loss', 'val_binary_accuracy'])
```

Burada başı "val_" ile başlayan "val_xxx" biçimindeki anahtarlar epoch sonrasında sinama işlemlerinden elde edilen değerleri, başı "val_" ile başlamayan anahtarlar ise doğrudan eğitim veri kümese iliskin değerleri belirtmektedir. Bu anahtarları biz sözlüğe verdigimizde sözlük bize her epoch sonrasında ilgili değeri bir liste biçiminde vermektedir. Örneğin burada 100 epoch uygulandığına göre bu dizinin elemanları 100'lük olacaktır. Bu durumda biz fit işleminde her epoch'tan elde edilen metrik değerlerinin grafiğini şöyle çizdirebiliriz:

```
import matplotlib.pyplot as plt

figure = plt.gcf()
figure.set_size_inches((15, 5))
plt.title('Loss - Epoch Graphics')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.plot(range(1, len(hist.history['loss']) + 1), hist.history['loss'])
plt.plot(range(1, len(hist.history['val_loss']) + 1), hist.history['val_loss'])
plt.legend(['Loss', 'Validation Loss'])
plt.show()

figure = plt.gcf()
figure.set_size_inches((15, 5))
plt.title('Binary Accuracy - Epoch Graphics')
plt.xlabel('Epoch')
plt.ylabel('Binary Accuracy')
plt.plot(range(1, len(hist.history['binary_accuracy']) + 1), hist.history['binary_accuracy'])
plt.plot(range(1, len(hist.history['val_binary_accuracy']) + 1),
hist.history['val_binary_accuracy'])
plt.legend(['Binary Accuracy', 'Validation Binary Accuracy'])
plt.show()
```



Pekiyi history bilgilerine neden gereksinim duyabiliyoruz? İşte modeli eğitirken en uygun epoch değerini belirlemeye bu grafiklerden faydalananızı istiyoruz. Çünkü grafikler çizildiğinde belli epoch değerlerinden sonra anomaliler gözle görülebilmektedir. Aynı zamanda hangi epoch değerinden sonra artık epoch'u artırmanın bir fayda sağlama olmadığı da görülebilmektedir. Aynı zamanda yukarıdaki grafikler bize epoch kaynaklı overfit durumu hakkında da bilgi verebilmektedir. Örneğin epoch değerini 100'den 1000'e çıkartarak grafikleri yeniden oluşturalım:

```
import numpy as np
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
from sklearn.model_selection import train_test_split

dataset = np.loadtxt('diabetes.csv', skiprows=1, delimiter=',')
dataset_x = dataset[:, :-1]
dataset_y = dataset[:, -1]
training_dataset_x, test_dataset_x, training_dataset_y, test_dataset_y =
train_test_split(dataset_x, dataset_y,
test_size=0.20)

model = Sequential()
model.add(Dense(100, input_dim=8, activation='relu', name='Hidden-1'))
model.add(Dense(100, activation='relu', name='Hidden-2'))
model.add(Dense(1, activation='sigmoid', name='Output'))

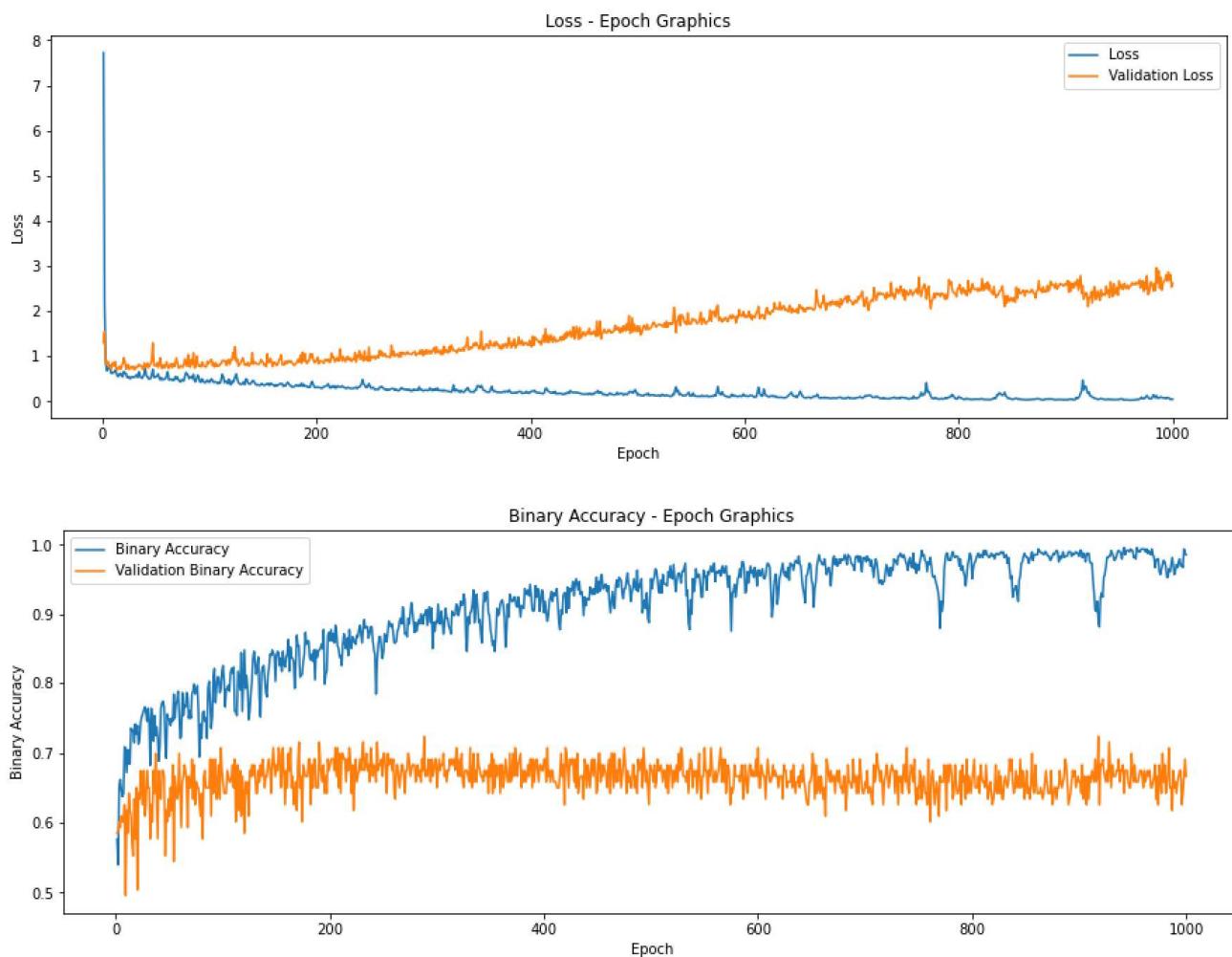
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['binary_accuracy'])

hist = model.fit(training_dataset_x, training_dataset_y, batch_size=32, epochs=1000,
validation_split=0.2)

import matplotlib.pyplot as plt

figure = plt.gcf()
figure.set_size_inches((15, 5))
plt.title('Loss - Epoch Graphics')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.plot(range(1, len(hist.history['loss']) + 1), hist.history['loss'])
plt.plot(range(1, len(hist.history['val_loss']) + 1), hist.history['val_loss'])
plt.legend(['Loss', 'Validation Loss'])
plt.show()

figure = plt.gcf()
figure.set_size_inches((15, 5))
plt.title('Binary Accuracy - Epoch Graphics')
plt.xlabel('Epoch')
plt.ylabel('Binary Accuracy')
plt.plot(range(1, len(hist.history['binary_accuracy']) + 1), hist.history['binary_accuracy'])
plt.plot(range(1, len(hist.history['val_binary_accuracy']) + 1),
hist.history['val_binary_accuracy'])
plt.legend(['Binary Accuracy', 'Validation Binary Accuracy'])
plt.show()
```



Gördüğünüz gibi 100 civarında bir epoch'tan sonra eğitim verilerinin verdiği metrik değerleriyle sınama verilerinin verdiği metrik değerler birbirinden kopmaya başlamıştır. Bu olguya "overfit" denilmektedir. Overfit olgusu ilerde ayrı bir başlıkta ele alınacaktır.

CSVLogger isimli callback sınıfı tipki History gibi epoch temelinde kayıt yapmaktadır. Ancak history'den farklı olarak bu kaydı bir CSV dosyasına yazar. Örneğin:

```
csv_callback = CSVLogger('diabetes.csv')
h = model.fit(training_set_x, training_set_y, batch_size=30, epochs=500,
callbacks=[csv_callback])
```

Metotlardaki callbacks parametresinin bir liste olduğuna dikkat ediniz. Yani biz bu metotlara birden fazla callback nesnesi geçirebiliriz.

Düzen çok kullanılan hazır callback sınıflarından biri de LambdaCallback sınıfıdır. Bu callback nesnesi aslında çeşitli olaylarda bizim ona verdığımız fonksiyonları çağırılmaktadır. LambdaCallback sınıfının `__init__` metodu şöyledir:

```
tf.keras.callbacks.LambdaCallback(
    on_epoch_begin=None,
    on_epoch_end=None,
    on_batch_begin=None,
    on_batch_end=None,
    on_train_begin=None,
    on_train_end=None,
    **kwargs
)
```

Metodun parametreleri olayları anlatmaktadır. Biz bu her parametreye bir fonksiyon girebiliriz. Bu fonksiyonların parametreleri de vardır. Parametreleri şunlardır:

Fonksiyon	Parametreler
on_epoch_begin	epoch ve logs
on_epoch_end	epoch ve logs
on_batch_begin	batch ve logs
on_batch_end	batch ve logs
on_train_begin	logs
on_train_end	logs

Buradaki epoch ve batch parametreleri epoch ve batch numaralarını vermektedir. logs parametresi ise bir sözlük biçimindedir. Tıpkı history nesnesinde olduğu gibi bize loss ve metrik değerleri bize verir. logs parametresinin verdiği loss ve metrik değerler on_xxx_begin fonksiyonlarında batch ya da epoch başlangıcındaki değerler, on_xxx_end fonksiyonlarında ise batch ya da epoch sonundaki değerlerdir.

Programcı kendisi de (custom) callback sınıfı yazabilir. Bunun için sınıfını tensorflow.keras.callbacks.Callback sınıfından türetmesi gereklidir. İşte aşağıdaki metodlar sınıfta yazılırsa ilgili işlemde bu metodlar deveye girecektir:

```
on_epoch_begin  
on_epoch_end  
on_batch_begin  
on_batch_end  
on_train_begin  
on_train_end
```

Örneğin:

```
from tensorflow.keras.callbacks import Callback  
  
class MyCallback(Callback):  
    def on_epoch_begin(self, epoch, logs):  
        print('epoch {} begins...'.format(epoch))  
  
    def on_epoch_end(self, epoch, logs):  
        print('epoch {} ends...'.format(epoch))  
  
my_callback = MyCallback()  
hist = model.fit(training_dataset_x, training_dataset_y, batch_size=32, epochs=10,  
validation_split=0.2, callbacks=[my_callback])  
result = model.evaluate(test_dataset_x, test_dataset_y)
```

Genel olarak buradaki metodların parametrik yapıları tensorflow.keras.callbacks.Callback sınıfında aşağıdaki biçimde belirtilmiştir. Programcının metodlarını yazarken bu parametrik yapılara uyması gereklidir:

```
class Callback(object):  
    def __init__(self):  
        self.validation_data = None  
        self.model = None  
  
    def set_params(self, params):  
        self.params = params  
  
    def set_model(self, model):  
        self.model = model  
  
    def on_epoch_begin(self, epoch, logs=None):  
        pass
```

```

def on_epoch_end(self, epoch, logs=None):
    pass

def on_batch_begin(self, batch, logs=None):
    pass

def on_batch_end(self, batch, logs=None):
    pass

def on_train_begin(self, logs=None):
    pass

def on_train_end(self, logs=None):
    pass

```

Metotlardaki batch parametresi o andaki batch uzuluğunu, epoch parametresi ise o andaki epoch numarasını belirtmektedir. logs parametresi ise yine metrik değerlerinin bulunduğu biz sözlük nesnesidir.

Verilerin Kullanıma Hazır Hale Getirilmesi Sürecinde Özellik Ölçeklendirmesi (Feature Scaling)

Biz daha önce verilerin kullanıma hazır hale getirilmesi sürecini çeşitli alt başlıklar halinde grupperdirmiştik. İşte bu alt başlıklardan "veri dönüştürmesi (data transformation)" grubundaki önemli bir teknik de "özellik ölçeklendirmesi (feature scaling)" denen tekniktir. Eğer bir veri tablosundaki sütunlarda bulunan değerlerin mertebeleri birbirinden çok farklı ise yapay sinir ağlarının bu durumlarda başarısı düşmektedir. Verilerin farklı mertebelere sahip olması aynı zamanda "geç yakınsama" sorunları da oluşturmaktadır. Örneğin bir evin fiyatının tahmin edilmesi örneğinde sütunların birinde evin kira geliri 1000'li mertebede iken diğer bir sütunda bulunan evin yaşı çok küçük bir mertebededir.

Özellik ölçeklendirmesinin neden gerekiği sezgisel olarak da anlaşılabilir. Bir nörona giren değerlerin ağırlıklarla çarpılarak toplandığını biliyorsunuz. Bu durumda yüksek mertebedeki değerler düşük mertebedeki değerlerin etkisini azaltabilecektir değil mi? Tabii özellik ölçeklendirmesi yalnızca yapay sinir ağlarında değil makine öğrenmesinin diğer bazı konularında da uygulanan bir tekniktir. Ancak "rassal ormanlar (random forest)", "karar ağaçları (decision trees)" gibi bazı makine öğrenmesi yöntemlerinde özellik ölçeklendirmesine gerek duyulmamaktadır.

Standart Ölçekleme (Standard Scaling): Bu yöntemde sütunun ortalaması ve standart sapması bulunur. Sonra sütündaki her değer ortalamadan çıkartılıp standart sapmaya bölünerek yeniden sütuna yazılır.

$$yeni \ x = \frac{x - \bar{x}}{\sigma_x}$$

Anımsayacağınız gibi aslında bu işlem farklı ortalama ve standart sapmaya ilişkin normal dağılım değerlerinin standart normal dağılım değerlerine (ortalaması 0, standart sapması 1 olan normal değerlerine) dönüştürülmesi için de kullanıldığını biliyorsunuz. Bu nedenle bu yönteme "özelliklerin standart hale getirilmesi (feature standardization)" de denilmektedir.

Bu işlemi aşağıdaki matris için numpy'da yapalım:

```

import numpy as np

def standard_scale(dataset):
    for col in range(dataset.shape[1]):
       冷data = dataset[:, col]
        dataset[:, col] = (coldata - np.mean(coldata)) / np.std(coldata)

```

Şimdi aşağıdaki yazdığımız fonksiyonu test edelim. Aşağıdaki gibi bir "test.csv" dosyası bulunuyor olsun:

```
1,10,13400  
2,12,14200  
4,4,15000  
2,6,12000  
4,6,11700  
8,9,14200  
3,1,34444
```

Test işlemini şöyle yapabiliriz:

```
dataset = np.loadtxt('test.csv', delimiter=',')  
standard_scale(dataset)  
print(dataset)
```

Sonuç da şöyle olacaktır:

```
[[ -1.14096529  0.90267093 -0.40588721]  
 [ -0.67115606  1.47709789 -0.29838776]  
 [  0.26846242 -0.82060994 -0.19088831]  
 [ -0.67115606 -0.24618298 -0.59401125]  
 [  0.26846242 -0.24618298 -0.63432354]  
 [  2.14769938  0.61545745 -0.29838776]  
 [ -0.20134682 -1.68225038  2.42188583]]
```

Bu işlem aslında daha pratik bir biçimde sklearn.preprocessing modülündeki StandardScaler sınıfıyla yapılmaktadır. Bu sınıf türünden bir nesne yaratıp fit_transform işlemi ile dönüştürmeyi yapabiliriz.

```
from sklearn.preprocessing import StandardScaler  
  
ss = StandardScaler()  
scaled_dataset = ss.fit_transform(dataset)  
print(scaled_dataset)
```

Bu işlemden de aynı çıktı elde edilmiştir:

```
[[ -1.14096529  0.90267093 -0.40588721]  
 [ -0.67115606  1.47709789 -0.29838776]  
 [  0.26846242 -0.82060994 -0.19088831]  
 [ -0.67115606 -0.24618298 -0.59401125]  
 [  0.26846242 -0.24618298 -0.63432354]  
 [  2.14769938  0.61545745 -0.29838776]  
 [ -0.20134682 -1.68225038  2.42188583]]
```

Aslında bu işlemleri tek adımda fit_transform ile yapmak yerine önce fit sonra transform biçiminde iki adımda da yapabilirdik. Örneğin:

```
import numpy as np  
from sklearn.preprocessing import StandardScaler  
  
ss = StandardScaler()  
ss.fit(dataset)  
scaled_dataset = ss.transform(dataset)  
print(scaled_dataset)
```

Burada fit metodu önce veri tablosundaki sütunların ortalamalarını ve standart sapmalarını heaplayarak sınıfın mean_ ve scale_ isimli örnek özniteliklerinde saklamakta transform metodu da bu değerlerden hareketle standart ölçeklendirmeyi yapmaktadır.

Standart ölçeklendirme sütunların normal dağılıma uyduğu durumlarda uygulanması tavsiye edilen bir tekniktir. Sütunların bu biçimde standart hale getirilmesi sonucunda değerlerin belli bir aralıktaki standart ölçeklendirmeyi yapmaktadır.

Ölçeklendirme yaparken dikkat edilmesi gereken önemli bir nokta aynı ölçeklendirmenin test veri kümesi için de yapılması gerekliliğidir. Ancak test veri kümesinin ölçeklendirilmesi kendi arasında değil eğitim veri kümesindeki değerlerden hareketle yapılmalıdır. Bu nedenle uygulamada fit ile transform işlemini ayırmak, fit işlemini eğitim veri kümesi için bir kez yapıp transform işlemini hem eğitim veri kümesi için hem de test veri kümesi için yapmak daha uygun bir yöntemdir. Şimdi standart ölçeklendirmeyi bu biçimde "diabetes.csv" dosyası üzerinde yapalım:

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

dataset = np.loadtxt('diabetes.csv', skiprows=1, delimiter=',')
dataset_x = dataset[:, :-1]
dataset_y = dataset[:, -1]
training_dataset_x, test_dataset_x, training_dataset_y, test_dataset_y =
train_test_split(dataset_x, dataset_y, test_size=0.2)

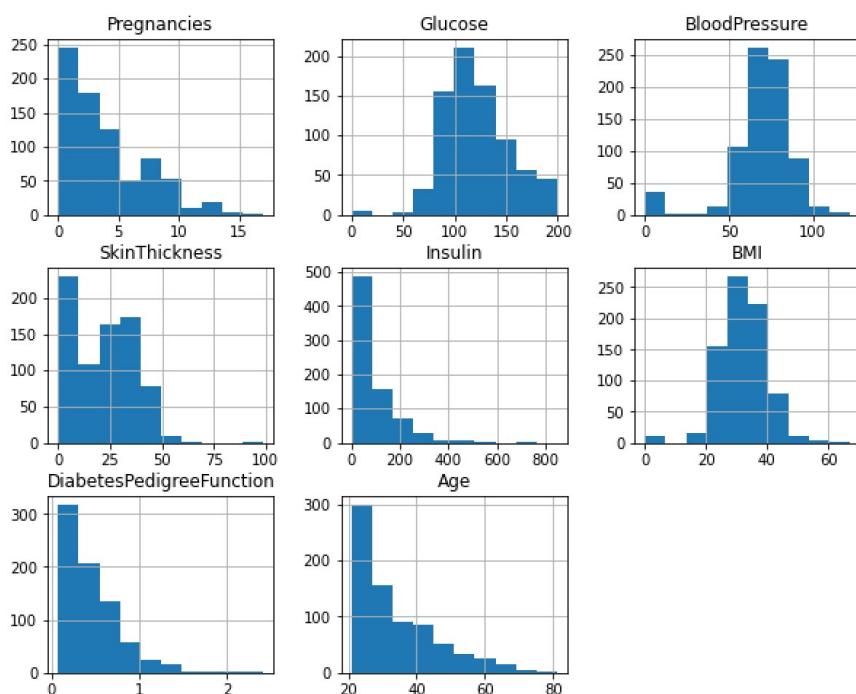
ss = StandardScaler()
ss.fit(training_dataset_x)
training_dataset_x = ss.transform(training_dataset_x)
test_dataset_x = ss.transform(test_dataset_x)
```

Biz burada yalnızca test amaçlı olarak "diabetes.csv" üzerinde standart ölçeklendirme yaptık. Yukarıda da belirttiğimiz gibi standart ölçeklendirme sütun verilerinin normal dağıldığı durumlarda tercih edilmesi gereken bir tekniktir. Dolayısıyla "diabetes.csv" sütunlarının normal dağıldığını bilmedikten sonra bu tekniği kullanmamamız gereklidir. Pekiyi acaba bu sütunlardaki verile gerçekten normal dağılıyor mu, histogram çizerek bakalım. Bu histogramları DataFrame sınıfının hist metodunu kullanarak çok daha kolay çizebiliriz:

```
import pandas as pd

df = pd.read_csv('diabetes.csv')
df.iloc[:, :-1].hist(figsize=(10, 8))
```

Elde edilen histogramlar şöyledir:



Burada sütunların bazılarının normal dağılıma benzедiğini ancak bazılarının ise benzemediğini görüyorsunuz. Normal dağılmayan sütunların bulunduğu veri tablollarında standart ölçeklendirme yapmak uygun değildir.

Ölçeklendirmede diğer önemli bir nokta da eğer veriler ölçeklendirme ile oluşturulmuşsa predict işlemi yapılırken predict metoduna geçirilecek değerlerin de aynı ortalama ve standart sapma değerleriyle ölçeklendirmeye sokulması gerektidir. Örneğin:

```
predict_result = model.predict(ss.transform(predict_data))
```

Burada ss daha önce eğitim verileriyle fit ya da fit_transform yapılmış StandardScaler nesnesini belirtmektedir.

Min-Max Ölçeklendirmesi: Bu ölçeklendirmede sütunun en küçük ve en büyük elemanları bulunur. Bu elemanlara göre ölçeklendirme yapılır:

$$\text{Yeni } x = \frac{x - x_{\min}}{x_{\max} - x_{\min}}$$

Buradan elde edilen değerler her zaman [0, 1] arasında olacaktır. Min-Max ölçeklemesini yapan fonksiyonu şöyle yazabiliriz:

```
import numpy as np

def minmax_scale(dataset):
    maxcol = np.max(dataset, axis=0)
    mincol = np.min(dataset, axis=0)
    diff = maxcol - mincol
    for col in range(dataset.shape[1]):
        dataset[:, col] = (dataset[:, col] - mincol[col]) / diff[col]

dataset = np.loadtxt('test.csv', delimiter=',')
minmax_scale(dataset)
print(dataset)
```

Kodun çalıştırılması sonucunda şöyle bir çıktı elde edilmiştir:

```
[[0.          0.81818182 0.07474499]
 [0.14285714 1.          0.1099191 ]
 [0.42857143 0.27272727 0.14509321]
 [0.14285714 0.45454545 0.01319029]
 [0.42857143 0.45454545 0.          ]
 [1.          0.72727273 0.1099191 ]
 [0.28571429 0.          1.          ]]
```

sklearn.preprocessing modülündeki MinMaxScaler sınıfı bu işlemi otomatik olarak yapmaktadır. Sınıfın kullanımı tamamen StandardScaler sınıfında olduğu gibidir. Önce MinMaxScaler sınıfı türünden bir nesne yaratılır. Ondan sonra sınıfın fit ve transform metotları çağrılır. Yine istenirse bu iki işlem fit_transform metodu ile tek hamlede de yapılabilmektedir. Örneğin:

```
import numpy as np

from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler

dataset = np.loadtxt('diabetes.csv', skiprows=1, delimiter=',')
dataset_x = dataset[:, :-1]
dataset_y = dataset[:, -1]
training_dataset_x, test_dataset_x, training_dataset_y, test_dataset_y =
train_test_split(dataset_x, dataset_y,
test_size=0.20)
```

```

mms = MinMaxScaler()
mms.fit(training_dataset_x)
training_dataset_x = mms.transform(training_dataset_x)
test_dataset_x = mms.transform(test_dataset_x)

model = Sequential()
model.add(Dense(100, input_dim=8, activation='relu', name='Hidden-1'))
model.add(Dense(100, activation='relu', name='Hidden-2'))
model.add(Dense(1, activation='sigmoid', name='Output'))

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['binary_accuracy'])

hist = model.fit(training_dataset_x, training_dataset_y, batch_size=32, epochs=100,
validation_split=0.2)

import matplotlib.pyplot as plt

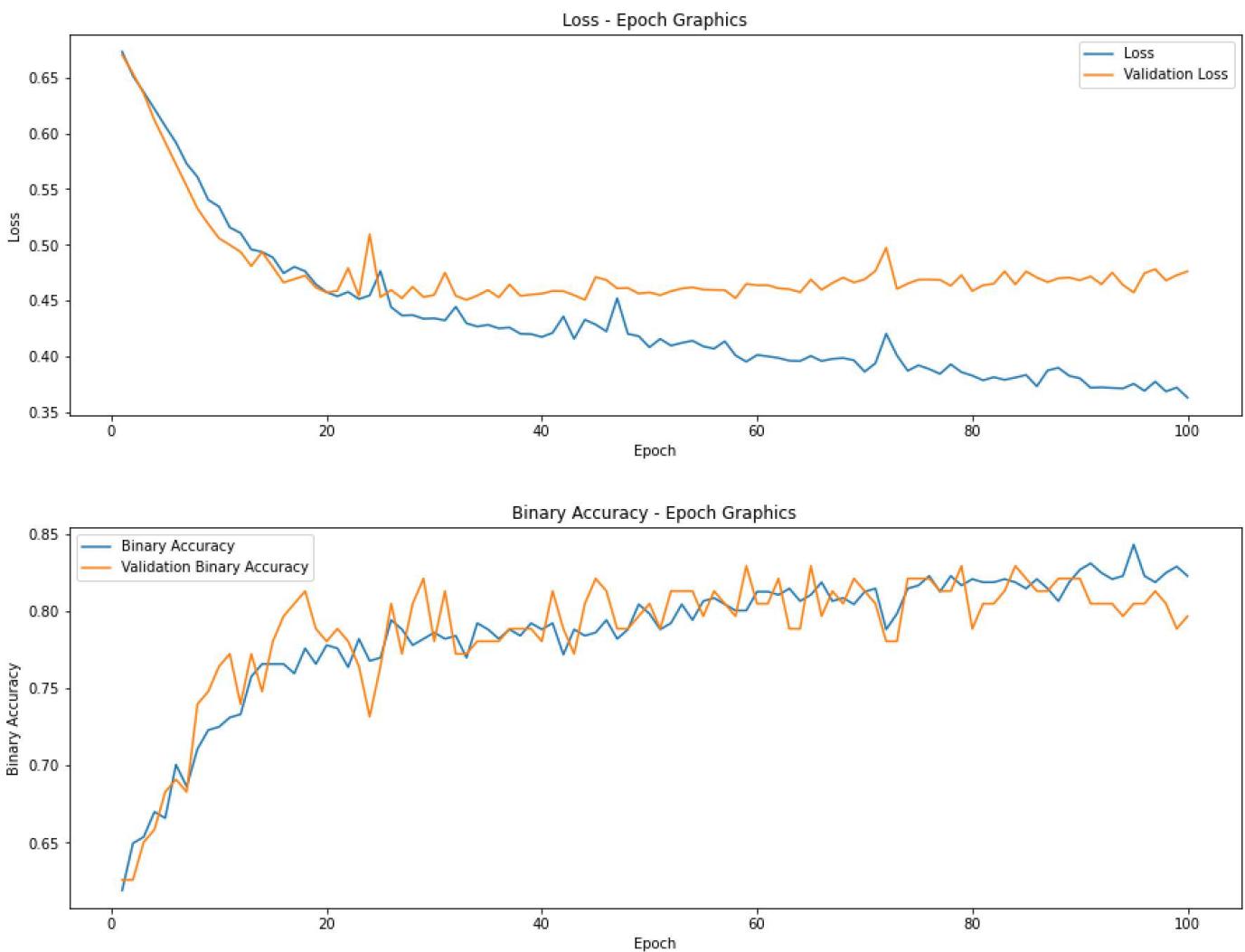
figure = plt.gcf()
figure.set_size_inches((15, 5))
plt.title('Loss - Epoch Graphics')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.plot(range(1, len(hist.history['loss']) + 1), hist.history['loss'])
plt.plot(range(1, len(hist.history['val_loss']) + 1), hist.history['val_loss'])
plt.legend(['Loss', 'Validation Loss'])
plt.show()

figure = plt.gcf()
figure.set_size_inches((15, 5))
plt.title('Binary Accuracy - Epoch Graphics')
plt.xlabel('Epoch')
plt.ylabel('Binary Accuracy')
plt.plot(range(1, len(hist.history['binary_accuracy']) + 1), hist.history['binary_accuracy'])
plt.plot(range(1, len(hist.history['val_binary_accuracy']) + 1),
hist.history['val_binary_accuracy'])
plt.legend(['Binary Accuracy', 'Validation Binary Accuracy'])
plt.show()

eval_result = model.evaluate(test_dataset_x, test_dataset_y)
for i in range(len(eval_result)):
    print(f'{model.metrics_names[i]}: {eval_result[i]}')

```

Elde edilen çıktılar şöyledir:



loss: 0.4925406277179718
 binary_accuracy: 0.7922077775001526

Buradan elde ettiğimiz değerleri Min-Max ölçeklemesi yapılmadan elde edilen değerlerle kıyaslayalım. Ölçekleme yapmadan elde ettiğimiz değerler şöyledir:

loss: 0.5974747538566589
 binary_accuracy: 0.6623376607894897]

Gördüğünüz gibi Min-Max ölçeklemesi modeli %13 civarında iyileştirmiştir.

Mutlak Max Ölçeklendirmesi (Absolute Max Scaling): Bu ölçeklendirmede sütundaki değerler sütunun en büyük değerinin mutlak değerine bölünerek dönüştürülmektedir:

$$\text{Yeni } x = \frac{x}{|\max(x)|}$$

Bu işlem sonucunda değerlerin [min, 1] aralığında ölçeklendirildiğine dikkat ediniz. Eğer tüm değerler pozitif ise bu ölçeklendirme Min-Max ölçeklendirmesine benzemektedir. Mutlak Max ölçeklendirmesi `sklearn.preprocessing.MaxAbsScaler` sınıfıyla temsil edilmiştir. Sınıfın kullanılması benzer biçimdedir. Örneğin en son örnekte kullandığımız "test.csv" dosyasındaki sütunları mutlak max ölçeklendirmesiyle ölçeklendirelim:

```
import numpy as np
from sklearn.preprocessing import MaxAbsScaler

dataset = np.loadtxt('test.csv', delimiter=',')
mas = MaxAbsScaler()
```

```

scaled_dataset = mas.fit_transform(dataset)
print(dataset)
print()
print(scaled_dataset)

```

Kodun çıktısı şöyle olacaktır:

```

[[1.0000e+00 1.0000e+01 1.3400e+04]
 [2.0000e+00 1.2000e+01 1.4200e+04]
 [4.0000e+00 4.0000e+00 1.5000e+04]
 [2.0000e+00 6.0000e+00 1.2000e+04]
 [4.0000e+00 6.0000e+00 1.1700e+04]
 [8.0000e+00 9.0000e+00 1.4200e+04]
 [3.0000e+00 1.0000e+00 3.4444e+04]]

[[0.125      0.83333333 0.38903728]
 [0.25       1.          0.41226338]
 [0.5        0.33333333 0.43548949]
 [0.25       0.5          0.34839159]
 [0.5        0.5          0.3396818 ]
 [1.         0.75         0.41226338]
 [0.375     0.08333333 1.          ]]

```

Yukarıda da belirttiğimiz gibi eğer sütun verileri normal dağılmışsa standart ölçeklendirme daha uygun olmaktadır. Böyle bir normalilik söz konusu değilse ve sütundaki verilerin en büyük değeri önceden biliniyorsa Min-Max ölçeklendirmesini kullanmak uygun olur. Biz de kursumuzda hep bu iki ölçeklendirmeyi kullanacağız.

Yapay Sinir Ağlarının Eğitilmesinde "Overfitting" ve "Underfitting" Olgusu

Yapay sinir ağlarının eğitilmesinde çok karşılaşılan iki terim "overfitting" ve "underfitting" terimleridir. "Overfitting" kabaca modelin "eğitim veri kümesinde iyi performans gösterdiği halde sınama ve test veri kümelerinde düşük bir performans göstermesi" anlamına gelmektedir. Yani eğitim sırasında model bir şeyi öğrenmiş olabilir ancak öğrendiği şey bizim arzu ettiğimiz şey olmayı bilmez. (Örneğin derste çözülen soruların aynısını çözebilen ama derste çözülmeyen sorularda aynı başarıyı gösteremeyen bir öğrenci düşünün. Bu öğrenciye derste çözülen sorular sorulursa o çok başarılı olacaktır. Ancak aynı konuya ilgili derste çözülmeyen sorular sorulduğunda başarısı düşecektir. Bu durumda öğrencinin konuyu iyi bir biçimde öğrendiğini söyleyebilir miyiz?) Overfitting olusunu sinyal-gürültü (singal-noise) teorisiyle de açıklayabiliriz. Eğitim verilerimizde öğrenilmesi gereken kalıplar "sinyallerdir" ancak bu sinyalların arasında "gürültüler" de vardır. Eğer modelimiz sinyalleri değil de gürültülerini öğrenme eğilimindeyse overfitting oluşmaktadır.

Pekiyi overfitting oluşturan kaynaklar nelerdir? İşte overfitting çeşitli nedenlerden dolayı oluşabilmektedir. Bazı overfitting durumları modelin doğru bir biçimde oluşturulamamasından kaynaklanırken bazı overfitting durumları ise eğitim verilerinin uygun biçimde oluşturulamamasından kaynaklanmaktadır.

Örneğin modelin çok fazla parametreye sahip olması overfitting sorununa yol açabilmektedir. Bu durumda model parametrelerinin (yani 'w' değerlerinin) toplam sayısının çeşitli tekniklerle düşürülmesi gerekebilir. Modelin eğitilmesi sırasında fazla epoch uygulamak diğer önemli overfitting nedenlerindendir. Eğer overfitting epoch temelli olarak ortaya çıkıysa veri bilimcisinin eğitimi uygun bir epoch'ta kesmesi gereklidir. Keras kütüphanesi belli koşullarda eğitimin sonlandırılmasına (early stopping) izin vermektedir.

Eğitim veri kümesinin uygun seçilmiş olmaması da overfitting sorununa yol açabilmektedir. Eğitim veri kümesi yanlış (bias) bir biçimde seçilmiş olabilir. Bu durumda eğitim sırasında ağ öğrenmesi gerekenleri değil başka şeyleri öğreniyor olabilir. Yanlılık dışında eğitim veri kümesinin değişkenliğinin (variance) düşük olması da bir "overfitting" kaynağını oluşturabilmektedir. Çünkü düşük değişkenlik (low variance) ana kütlenin iyi bir biçimde temsil edilememesi anlamına gelmektedir.

Pekiyi "overfit" durumunu nasıl anlarız? İşte eğitim sırasında eğitim verilerinin verdiği metrik değerler iyileşirken bu iyileşme sınama verilerinde gözükmemiyorsa yani eğitim verilerinin kendisinden elde edilen başarı sınama sırasında

ortaya çıkmıyorsa bu durumda bir "overfitting" şüphesi akla gelmelidir. Bu nedenle eğitim sırasında eğitim kümesinden elde edilen metrik değerlerle sınama kümesinden elde edilen metrik değerlerin belli bir uyumda olup olmadığından bir biçimde kontrol edilmesi gerekmektedir. Bu kontrolün gözle yapılması kolaydır. Biz Keras'ta eğitim sırasında fit metodunun verdiği history nesnesinden hareketle eğitim verileri ve sınama verileri için elde edilen metrik değerlerin grafiğini birlikte çizebiliriz. Grafikteki ayışmayı bu yolla görebiliriz.

Overfitting olsusunun tersine "underfitting" denilmektedir. Underfitting olsusu da yine modelin yanlış oluşturulmasından ve eğitim veri kümesindeki sorunlardan kaynaklanıyor olabilir. Örneğin özelliklerin (yani tablo sütunlarının) yanlış ve eksik seçilmesi, modeldeki nöronların sayısının azlığı, yanlış optimizasyon algoritmalarının ve loss fonksiyonlarının kullanılması modelin yanlış oluşturulduğuna ilişkin akla gelen en önemli unsurlardır. Öte yandan nasıl düşük varyans overfitting durumuna yol açıysa yüksek varyans da underfitting durumuna yol açabilmektedir. Eğitim veri kümesindeki yüksek varyanstan dolayı metrik değerler bir türlü iyileşemeyebilir. Eğitim veri kümesinin azlığı, eğitim verilerinin yüksek bir entropide olması da önemli underfitting kaynaklarındanandır. Underfitting olsusunu yine sinyal-gürültü teorisiyle açıklayabiliriz. Eğer gürültü sinyalden çok daha fazla ise bu durumda sinyal bir türlü elde edilemeyecektir.

Regresyon Kavramı ve Regresyon Modellerinin Sınıflandırılması

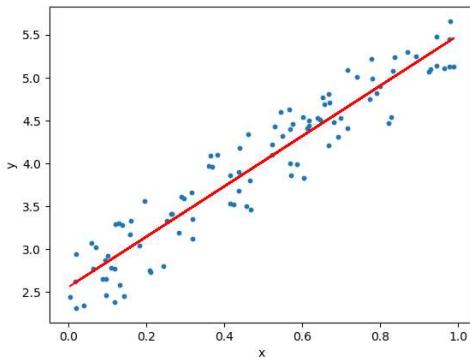
Girdi ile çıktı arasında ilişki kurma sürecine istatistikte "regresyon" denilmektedir. Girdiyle çıktı arasındaki ilişki bir fonksiyonla ifade edilebildiğine göre regresyon sürecini de biz aslında girdi ile çıktı arasındaki ilişkiyi belirten uygun bir fonksiyonun bulunması süreci olarak ele alabiliriz. Matematiksel birimde açıklarsak regresyon aslında $y = f(x)$ biçiminde bir f fonksiyonun bulunması sürecidir. (Burada x ve y birer vektör olarak düşünülmelidir. Yani x bir tane değil x_0, x_1, \dots, x_n biçiminde n tane olabileceği gibi y de bir tane değil y_0, y_1, \dots, y_m biçiminde m tane olabilir.)

Pekiyi girdi ile çıktı arasında ilişki kurmanın amacı ne olabilir? Şüphesiz en önemli amaç kestirimde bulunmaktır. Örneğin elimizde birtakım geçmiş veriler vardır. Biz de gelecekteki durumun ne olabileceğile ilgili karar vermek isteyebiliriz. Bu durumda gelecekteki verileri regresyon sonucunda bulduğumuz f fonksiyonuna girdi yaparak sonucu elde ederiz. Biz kursumuzda girdiler yoluyla çıktıların tahmin edilmesine yönelik problemlere "regresyon problemleri" diyeceğiz.

Regresyon problemlerinin çözümü için değişik yöntemler kullanılmaktadır. İstatistiksel regresyon analizi makine öğrenmesinin popüler hale gelmesinden çok önceleri uygulanan ve oldukça bilinen bir teknikler topluluğudur Ancak makine öğrenmesinde istatistiksel regresyon analizinin dışında kullanılan başka özel teknikler de bulunmaktadır. Örneğin yapay sinir ağları da aslında bir regresyon sürecidir ve regresyon problemlerinde kestirim amacıyla kullanılmaktadır.

Regresyon modelleri ve problemleri istatistikte çeşitli biçimlerde sınıflandırılabilirmektedir. Maalesef herkesin hemfikir olduğu bir sınıflandırma biçimi yoktur. Biz burada regresyon modellerini tipik olarak bazı alt başlıklarla sınıflandıracağız. Regresyon sonucunda elde edilecek fonksiyonun genel yapısına göre regresyon modelleri tipik olarak aşağıdaki gibi sınıflandırılmaktadır:

- Doğrusal Regresyon (Linear Regression): Girdi çıktı arasındaki ilişkinin doğrusal olduğu kabulüyle yapılan regresyon sürecine denilmektedir. Yani doğrusal regresyonla girdi ile çıktı arasında doğrusal bir fonksiyon bulunmaya çalışılır. Örneğin:



Burada düzleme çeşitli noktaları ortalayan bir doğru elde edilmiştir. Bu doğru aslında noktaların doğruya uzaklıklarının en küçük olduğu doğrudur ve istatistikte bu doğrunun elde edilmesi yöntemine "en küçük kareler" yöntemi denilmektedir. İstatistiksel doğrusal regresyon kursumuzda ileride ele alınmaktadır.

Doğrusal regresyonda girdi (yani bağımsız değişken) bir tane ve çıktı da (bağımlı değişken) bir tane ise buna "basit doğrusal regresyon" denilmektedir. Tabii gerçek yaşama ilişkin problemlerde genellikle girdi bir tane olmaz. Örneğin bizim kullandığımız veri tablolarındaki özellikler (sütunlar) girdileri oluştururlar. Böylece regresyon modelinde çok sayıda girdi söz konusu olur. İşte eğer doğrusal regresyonda girdiler (yani bağımsız değişkenler) birden fazla ise buna "çoklu doğrusal regresyon (multiple linear regression)" denilmektedir. Doğrusallığın değişken sayısı ile ilgili olmadığına değişkenlerin dereceleri ile ilgili olduğuna dikkat ediniz. Örneğin n tane değişken içeren doğrusal bir fonksiyonun genel biçimi şöyledir:

$$f(x) = a_1x_1 + a_2x_2 + a_3x_3 + \dots + a_nx_n + b$$

Biz bu doğrusal fonksiyonu vektörel biçimde şöyle de ifade edebiliriz:

$$f(X) = AX$$

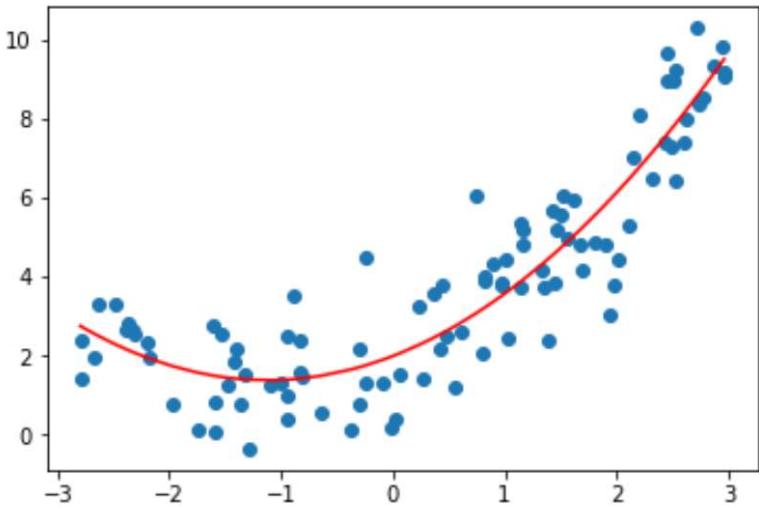
Burada A bir satır vektörünü X ise bir sütun vektörünü belirtmektedir. Aslında AX işleminin bir "dot product" oluşturduğuna dikkat ediniz.

İki boyutlu düzleme tek değişkenli doğrulara biz çok aşınayız. Burada eksenlerden biri bağımsız değişkeni diğer ise bağımlı değişkeni belirtir. Eğer bağımsız değişken iki tane ise doğru denklemi de bir doğru değil bir düzleme belirtir hale gelir. Düzleme üç boyutlu uzayda bir doğru gibi düşünülmelidir. O halde n boyutlu uzayın $n - 1$ değişkenle ifade edilebilen bir düzleme vardır. Burada anlatmak istediğimiz şey doğrusallığın genel prensiplerinin değişmediğidir. İki boyutlu uzaydaki tek değişkenli doğrusallıkla n boyutlu uzaydaki $n - 1$ boyutlu doğrusallık aynı prensiplere sahiptir.

- Polinomsal Regresyon (Polynomial Regression): Verilerin grafiğini çizdiğimizde ilişkinin doğrusal olup olmadığı hemen gözle görülebilmektedir. Bağımsız değişkenlerin herhangi birinin üssü 1'den büyükse bu tür regresyon modellerine polinomsal regresyon modelleri denilmektedir. Başka bir deyişle polinomsal regresyon aşağıdaki gibi bir polinomsal fonksiyonun bulunması sürecidir:

$$f(x) = a_0x^0 + a_1x^1 + a_2x^2 + a_nx^n$$

Polinomsal regresyona aşağıdaki gibi bir örnek verebiliriz:



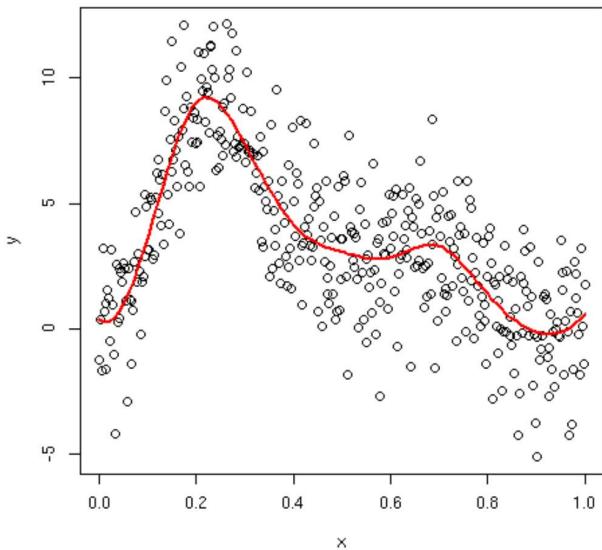
Alıntı Notu: GörSEL <https://developpaper.com/polynomial-regression-and-model-generalization-in-machine-learning/> adresinden alınmıştır.

Burada söz konusu noktalardan bir doğru değil bir polinom geçirilmiştir. Aslında polinomsal regresyon eğer katsayılarla değişkenler yer değiştirirse doğrusal regresyona benzemektedir:

$$f(x) = x^0 a_0 + x^1 a_1 + x^2 a_2 + x^n a_n$$

Polinomsal regresyon kursumuzun ilerleyen bölümlerinde ayrı bir başlık halinde ele alınacaktır.

- **Doğrusal Olmayan Regresyon (Nonlinear Regression):** Her ne kadar polinomlar doğrusal fonksiyonlar değilse de katsayılarla değişkenler yer değiştirdiğinde doğrusal gibi ele alınabilemektedir. İstatistikte doğrusal olmayan regresyon denildiğinde katsayılarla değişkenler değiştirildiğinde yine doğrusal olmayan fonksiyonların elde edildiği regresyonlar anlaşılmaktadır. Bu tür fonksiyonalar genel olarak üstel ifadeler, logaritmik ifadeler, pay ve paydasında bağımsız değişkenlerin bulunduğu oransal ifadeler içermektedir.



Alıntı Notu: GörSEL <https://stackoverflow.com/questions/15837763/b-spline-confusion> adresinden alınmıştır.

Aslında doğrusal olmayan regresyonları da elde edilmek istenen fonksiyonlara göre logaritmik regresyon, üstel regresyon gibi alt sınıflara ayırmak mümkündür.

Lojistik Regresyon (Logistic/Logit Regression): Bağımlı değişkenin sürekli bir değer almadığı, kategorik değer aldığı durumlarda uygulanan regresyonlara lojistik regresyon ya da logit regresyonu denilmektedir. Lojistik regresyon terimi tipik olarak çıktıının 0 ya da 1 gibi ikili değerlere sahip olduğu durumlar için kullanılmaktadır. (Örneğin çıktı "evli mi bekar mı", "hasta mı sağlıklı mı", "film iyi mi kötü mü" gibi iki seçenekten biri olabilmektedir.) Ancak zamanla bu

terim genişletilmiştir. Çıktının ikiden fazla kategoriye ayrıldığı durumlar için de "çok sınıflı lojistik regresyon (multinomial logistic regression)" terimi kullanılmaktadır. Ayrıca "sıralı lojistik regresyon (ordinal logistic regression)" denilen bir lojistik regresyon modelinde girdi ve çıktılar kategorik değil sıralı ölçeklere ilişkin olabilmektedir.

Regresyonlar bağımlı ve bağımsız değişken sayılarına göre de sınıflandırılmaktadır:

- **Basit Regresyon (Simple Regression):** Bağımsız değişken sayısı bir tane ise buyle regresyonlara basit regresyonlar denilmektedir.

- **Çoklu Regresyon (multiple regression):** Bağımsız değişken sayısının (yani girdilerin) birden fazla olduğu fakat bağımlı değişken sayısının (yani çıktıının) bir tane olduğu regresyon modelleri için kullanılan bir terimdir. Örneğin bir otomobilin 8 özelliğinden onun yakıt harcamasının tahmin edilmek istediği regresyon modeli çoklu regresyon modelidir. Ya da örneğin "diabeteses.csv"örneğinde olduğu gibi kişinin 8 biyomedikal bilgisinden hareketle kişinin şeker hastası olup olmadığına tahmin edilmesi için oluşturulan model de çoklu lojistik regresyon modelidir.

- **Çok Değişkenli Regresyon (Multivariate Regression):** Eğer regresyon modelinde bağımlı değişkenin sayısı birden fazlaysa (yani çıktıı birden fazlaysa) buna da "çok değişkenli (multivariate)" regresyon denilmektedir. Örneğin öğrencinin bazı girdi bilgileri olsun (bağımsız değişkenler) biz de onun sınavda alacağı notu ve ortalama kaç saat uyuduğunu tahmin etmek isteyelim. Burada tahmin etmek istediğimiz şey birden fazladır. Şüphesiz çok değişkenli regresyonlarda çıktıılar birbirlerinden bağımsız ve teker teker olarak da ele alınabilirler. O zaman iki farklı çoklu regresyondan söz ederdim. Fakat çok değişkenli regresyonlarda aynı anda birden fazla çıktıının değişiminin belirlenmesi hedeflenmektedir. Yani örneğin 15 tane biyomedikal tetkike bakarak biz bir kişinin "diabetli olup olmadığını", "kalp hastası olup olmadığını", "hipertansiyonunun olup olmadığını" ayrı ayrı çoklu lojistik regresyonla anlamaya çalışabilirim. Ancak bu üç hastalık birbirlerini de etkiliyor olabilir. O halde bu üç hastalığın birlikte değerlendirilmesi gereklidir. İşte bu durum çok değişkenli lojistik regresyon olarak modellenebilir.

Yapay Sinir Ağları ve Derin Ağlarla Regresyon Modelleri ile İstatistiksel Regresyon Modellerinin Karşılaştırılması

Yukarıda da belirttiğimiz gibi aslında regresyon modellerini çözmek için değişik teknikler kullanılabilmektedir. Kursumuzun ilerleyen zamanlarında regresyon problemlerinin başka tekniklerle çözümü üzerinde de durulacaktır. Pekiyi yapay sinir ağları ve derin ağlar klasik istatistiksel regresyon analizine göre hangi durumlarda avantaj sağlamaaktadır? Bu durumları söyle ifade edebiliriz:

- Bağımsız değişken sayısının (girdi sayılarının) fazla olduğu durumlarda (örneğin 25'ten fazla olduğu durumlarda)
- Bağımsız değişkenler arasında korelasyonların olduğu durumlarda
- Bağımsız değişkenler arasında karmaşık ilişkilerin olduğu durumlarda
- Bağımsız değişkenlerle bağımlı değişkenler arasında doğrusal olmayan ilişkilerin bulunduğu durumlarda
- Bağımsız değişkenlerin bazlarının kategorik, bazlarının sürekli olduğu durumlarda
- Bağımlı değişken sayısının fazla olduğu ve onların arasında da ilişkilerin bulunduğu

Bu tür durumlarda istatistiksel regresyon analizleri genellikle sinir ağ modellerine göre hem daha düşük performans göstermekte hem daha fazla bilgisayar zamanı almaktır.

Tabii aslında klasik istatistiksel regresyon analizi modellerinin yapay sinir ağları ve derin ağlardan daha yüksek performans göstermesi de söz konusu olabilmektedir. Özellikle girdi ile çıktı arasındaki ilişkilerin çok belirgin olduğu durumlarda istatistiksel regresyon analizi tercih edilebilmektedir. Yine benzer biçimde elde çok veri olduğu durumda yapay sinir ağlarının eğitilmesi bir sorun oluşturabilmektedir. Bu durumlarda istatistiksel regresyonlar yapay sinir ağlarına tercih edilebilmektedir.

Yapay Sinir Ağları ile Oluşturulan Regresyon Modellerinde Önemli Parametrelerin Belirlenmesi

Yapay sinir ağlarında ve derin ağlarda regresyon modeli oluşturulurken genellikle bazı sezgisel yöntemler izlenmektedir. Tabii daha önceden de bahsedildiği gibi aslında veri biliminde genel yaklaşım her zaman o andaki modele ve amaca özgü olarak değişimektedir. Ancak yine de çeşitli problem grupları için kaba bazı model belirlemeler söz konusu olabilmektedir. Bunları birkaç maddede ele almak istiyoruz:

- Regresyon modelinde genellikle saklı katmanlar için "relu", çıktı katmanı için regresyon modeli eğer ikili lojistik ise "sigmoid", çok sınıflı lojistik ise softmax, lojistik değilse "linear" aktivasyon fonksiyonları tercih edilmektedir. (Linear aktivasyon fonksiyonu girdinin hiçbir işleme sokulmadan çıktı olarak verildiği fonksiyondur. Dense katmanındaki default aktivasyonunun "linear" olduğunu anımsayınız.)
- Modeldeki optimizer algoritması için genellikle "rmsprop", "adam" ve "sgd (stochastic_gradient_descent)" seçilmektedir.
- Modelin loss fonksiyonu için genellikle ikili lojistik regresyon modelleri için "binary_crossentropy", çok sınıflı regresyon modelleri için "categorical_crossentropy" ve lojistik olmayan regresyon modelleri için de "mse (mean_squared_error)" tercih edilmektedir.
- Validation işlemi için kullanılacak metrikler de genellikle jöistik ikili regresyon modelleri için "binary_accuracy", çok sınıflı lojistik regresyon modelleri için "categorical_accuracy", lojistik olmayan regresyon modelleri için ise "mae (mean_absolute_error)" seçilmektedir.

Regresyon modelleri için özel durumlar haricinde mimari konusunda da genellikle şunlar uygulanmaktadır:

- Model katmanları Dense (yani tüm nöron çıktılarının diğerlerine bağlılığı) biçimde oluşturulur. Ancak özel durumlarda ileride de göreceğiniz gibi başka mimariler de kullanılabilir.
- Katmandaki nöron sayıları 100-200 gibi orta değerde tutulmaktadır.
- Önce tek saklı katman denenir, eğer performans düşük kalırsa katman sayısı ikiye, üçe, dörde ve duruma göre daha yüksek derecelere çıkartılır.
- Fazla miktarda katmanla hala iyileşme sağlanamıyorsa katmanlardaki nöron sayıları artırılır.
- Hala yeterli bir performans elde edilmiyorsa bu durumda mimarı, seçilen özelliklerden (sütunlardan) ya da eğitim için kullanılan verilerden şüphe edilmelidir.

Yapay Sinir Ağlarıyla Lojistik Olmayan Regresyon Örneği: Otomobillerin Mil Başına Yaktıkları Yakıtı Tahmin Etme

Şimdi lojistik olmayan regresyon problemlerinde çok kullanılan klasikleşmiş bir örnek üzerinde duracağız. Örnek verileri 70'li yılların sonlarına doğru toplanmıştır. Dolayısıyla değerler o yıllara özgüdür. Şimdi aşama aşama örneği gerçekleştirmeye çalışalım.

1) Önce modelimizde kullanacağımız veri kümesini aşağıdaki bağlantıdan indirebilirsiniz:

<https://archive.ics.uci.edu/ml/datasets/auto+mpg>

Bu bağlantıdan "auto-mpg.data" ve "auto-mpg.names" isimli dosyalar indirilebilir. Gerçek veri dosyası "auto-mpg.data" dosyasıdır. "auto-mpg.names" dosyasına açıklama bilgileri vardır. Veri kümesinde toplam 8 sütun vardır. Bu 8 sütun sırasıyla şunlardır oluşturmaktadır:

Aracın Mil başına yakıtı yakıtın galon miktarı (1 galon 3.78 litredir)
 Aracın Silindir Sayısı
 Aracın Motor Hacmi
 Aracın Beygir Gücü
 Aracın Ağırlığı
 Aracın 100 km/h ya çıkışma süresi (ivmelenmesi)
 Aracın Orijini (Kategorik 1: USA, 2: Europe, 3: Japan)
 Aracın Markası

Sütunlar dosyada birbirlerinden SPACE karakterleriyle ayrılmıştır. Ancak Markadan önce bir tane TAB karakteri bulunmaktadır. Verilerin örnek görünümü şöyledir:

1	18.0	8	307.0	130.0	3504.	12.0	70	1	"chevrolet chevelle malibu"
2	15.0	8	350.0	165.0	3693.	11.5	70	1	"buick skylark 320"
3	18.0	8	318.0	150.0	3436.	11.0	70	1	"plymouth satellite"
4	16.0	8	304.0	150.0	3433.	12.0	70	1	"amc rebel sst"
5	17.0	8	302.0	140.0	3449.	10.5	70	1	"ford torino"
6	15.0	8	429.0	198.0	4341.	10.0	70	1	"ford galaxie 500"
7	14.0	8	454.0	220.0	4354.	9.0	70	1	"chevrolet impala"
8	14.0	8	440.0	215.0	4312.	8.5	70	1	"plymouth fury iii"
9	14.0	8	455.0	225.0	4425.	10.0	70	1	"pontiac catalina"
10	15.0	8	390.0	190.0	3850.	8.5	70	1	"amc ambassador dpl"
11	15.0	8	383.0	170.0	3563.	10.0	70	1	"dodge challenger se"
12	14.0	8	340.0	160.0	3609.	8.0	70	1	"plymouth 'cuda 340"
13	15.0	8	400.0	150.0	3761.	9.5	70	1	"chevrolet monte carlo"
14	14.0	8	455.0	225.0	3086.	10.0	70	1	"buick estate wagon (sw)"
15	24.0	4	113.0	95.00	2372.	15.0	70	3	"toyota corona mark ii"
16	22.0	6	198.0	95.00	2833.	15.5	70	1	"plymouth duster"
17	18.0	6	199.0	97.00	2774.	15.5	70	1	"amc hornet"
18	21.0	6	200.0	85.00	2587.	16.0	70	1	"ford maverick"
19	27.0	4	97.00	88.00	2130.	14.5	70	3	"datsun pl510"
20	26.0	4	97.00	46.00	1835.	20.5	70	2	"volkswagen 1131 deluxe sedan"
21	25.0	4	110.0	87.00	2672.	17.5	70	2	"peugeot 504"
22	24.0	4	107.0	90.00	2430.	14.5	70	2	"audi 100 ls"
23	25.0	4	104.0	95.00	2375.	17.5	70	2	"saab 99e"
24	26.0	4	121.0	113.0	2234.	12.5	70	2	"bmw 2002"
25	21.0	6	199.0	90.00	2648.	15.0	70	1	"amc gremlin"
26	10.0	8	360.0	215.0	4615.	14.0	70	1	"ford f250"
27	10.0	8	307.0	200.0	4376.	15.0	70	1	"chevy c20"
28	11.0	8	318.0	210.0	4382.	13.5	70	1	"dodge d200"
29	9.0	8	304.0	193.0	4732.	18.5	70	1	"hi 1200d"
30	27.0	4	97.00	88.00	2130.	14.5	71	3	"datsun pl510"
31	28.0	4	140.0	90.00	2264.	15.5	71	1	"chevrolet vega 2300"
32	25.0	4	113.0	95.00	2228.	14.0	71	3	"toyota corona"
33	25.0	4	98.00	?	2046.	19.0	71	1	"ford pinto"

Son sütundaki otomobil markalarının bizim için bir önemi yoktur. Dolayısıyla son sütunu tamamen atabiliyoruz. Ayrıca dosya incelediğinde 3'üncü indeksli sütunda (bu sütun araçların beygir güçlerinden oluşmaktadır) '?' karakterlerinin olduğu görülmektedir. Bu '?' karakterleri ilgili aracın beygir gücünün bilinmediği anlamına gelir. Örneğimizde bu satırları tamamen atabiliyoruz. Şimdi dosyayı okuyup bu hazırlık işlemlerini yapalım:

```
import numpy as np

dataset = np.loadtxt('auto-mpg.data', usecols=range(8), dtype='object')
dataset = dataset[dataset[:, 3] != '?', :].astype('float32')
```

Artık elimizde '?' lerinden arındırılmış ve float32 formatına dönüştürülmüş bir ndarray nesnesi vardır. Şimdi x ve y veri kümelerini ayırtıralım. Çıktıların ilk sütunda olduğuna dikkat ediniz:

```
dataset_x = dataset[:, 1:]
dataset_y = dataset[:, 0]
```

Şimdi bizim arabanın orijinini belirten kategorik sütun için one hot encoding dönüştürmesi yapmamız gereklidir. Bu işlemi şöyle yapılabiliyoruz:

```
ohe = OneHotEncoder(sparse=False)
ohe_data = ohe.fit_transform(dataset_x[:, 6].reshape(-1, 1))
dataset_x = np.delete(dataset_x, 6, axis=1)
dataset_x = np.append(dataset_x, ohe_data, axis=1)
```

Şimdi de veri kümemizdeki sütunların mertebeleri arasında ciddi farklar olduğu için "özellik ölçeklendirmesi (feature scaling)" yapalım:

```
from sklearn.preprocessing import MinMaxScaler

mms = MinMaxScaler()
mms.fit(dataset_x)
dataset_x = mms.transform(dataset_x)
```

Şimdi veri kümemizi eğitim ve test amacıyla ikiye ayıralım:

```

from sklearn.model_selection import train_test_split

training_dataset_x, test_dataset_x, training_dataset_y, test_dataset_y =
train_test_split(dataset_x, dataset_y, test_size=0.20)

```

Artık modeli kurabiliyoruz. Modelimizde iki saklı katman olsun. Saklı katmanlar için aktivasyon fonksiyonlarını "relu" çıktı katmanı için de "linear" alabiliyoruz. Optimizer algoritması olarak aslında yukarıda belirttiğimiz "sgd", "adam" ya da "rmsprop" algoritmalarından herhangi birini seçebiliyoruz. Örneğimizde biz "rmsprop" kullanacağımız. Lojistik olmayan regresyon modellerinde loss fonksiyonunun genellikle "mse" ve metrik değerinin de "mae" olarak seçildiğini biliyorsunuz:

```

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential()
model.add(Dense(100, input_dim=training_dataset_x.shape[1], activation='relu', name='Hidden-1'))
model.add(Dense(100, activation='relu', name='Hidden-2'))
model.add(Dense(1, activation='linear', name='Output'))

model.compile('rmsprop', loss='mse', metrics=['mae'])

```

Modelimizin özet bilgilerini de yazdıralım:

```
model.summary()
```

Şöyledir bir çıktı elde edilmiş:

Model: "Auto-MPG"

Layer (type)	Output Shape	Param #
<hr/>		
Hidden-1 (Dense)	(None, 100)	1000
<hr/>		
Hidden-2 (Dense)	(None, 100)	10100
<hr/>		
Output (Dense)	(None, 1)	101
<hr/>		
Total params: 11,201		
Trainable params: 11,201		
Non-trainable params: 0		

Artık modelimizi eğitebiliriz:

```
hist = model.fit(training_dataset_x, training_dataset_y, batch_size=32, epochs=200)
```

Eğitimden elde ettiğimiz history bilgilerine dayanarak epoch temelinde loss değerinin ve 'mae' metrik değerinin grafiklerini çizelim:

```

import matplotlib.pyplot as plt

figure = plt.gcf()
figure.set_size_inches((15, 5))
plt.title('Loss - Epoch Graphics')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.plot(range(1, len(hist.history['loss']) + 1), hist.history['loss'])
plt.plot(range(1, len(hist.history['val_loss']) + 1), hist.history['val_loss'])

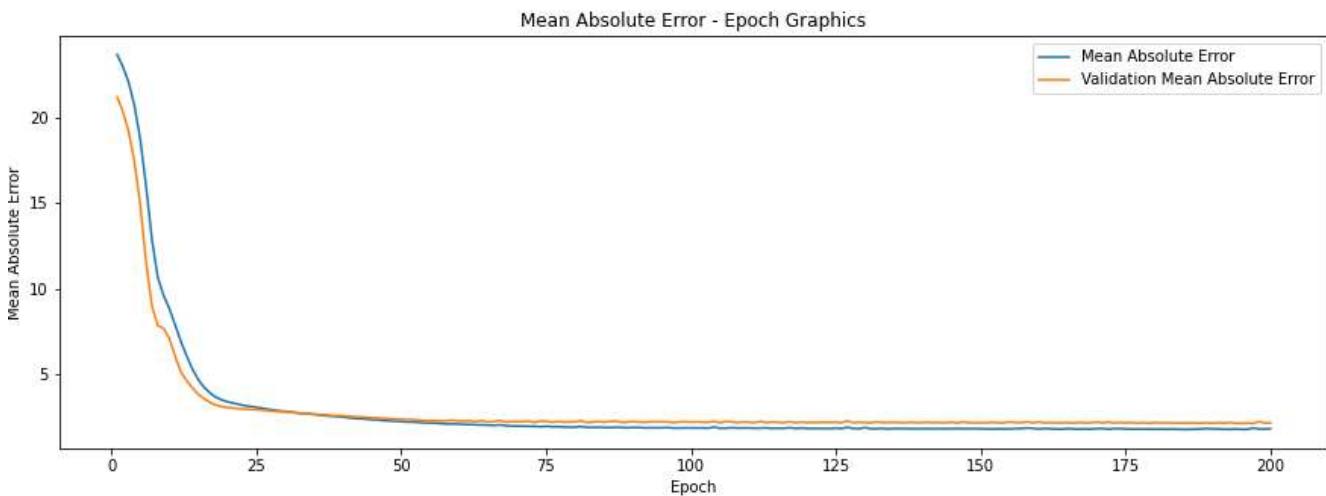
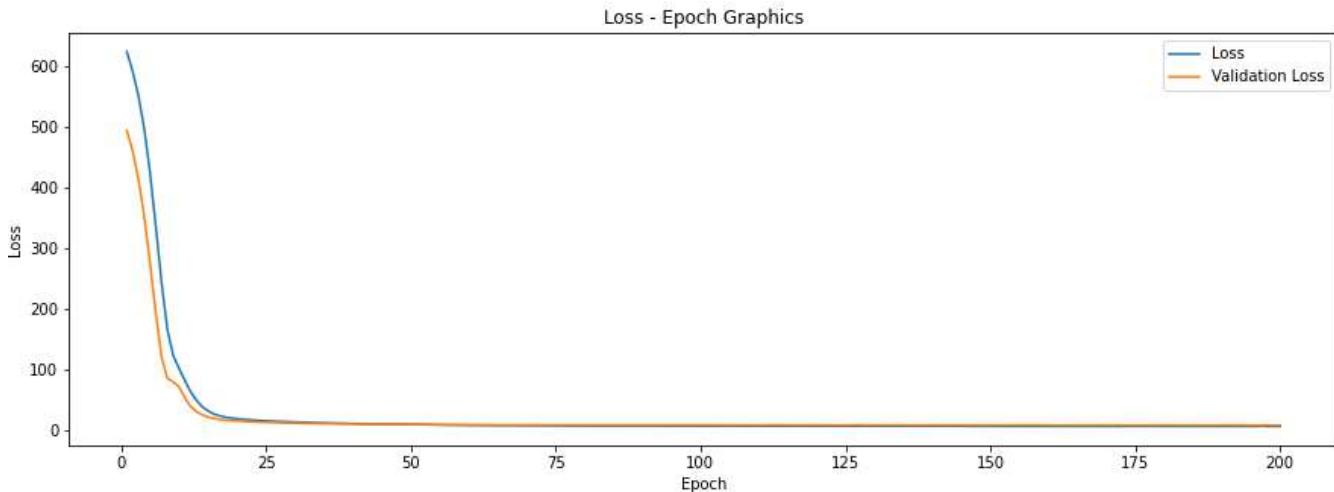
```

```

plt.legend(['Loss', 'Validation Loss'])
plt.show()

figure = plt.gcf()
figure.set_size_inches((15, 5))
plt.title('Mean Absolute Error - Epoch Graphics')
plt.xlabel('Epoch')
plt.ylabel('Mean Absolute Error')
plt.plot(range(1, len(hist.history['mae']) + 1), hist.history['mae'])
plt.plot(range(1, len(hist.history['val_mae']) + 1), hist.history['val_mae'])
plt.legend(['Mean Absolute Error', 'Validation Mean Absolute Error'])
plt.show()

```



Grafiklerden eğitim veri kümesi ile elde edilen metrik değerlerin sınama veri kümesi ile elde edilenlerle benzer olduğunu görüyorsunuz. Bu durum bir overfitting sorununun olmadığını gösteriyor. Biz eğitimde 200 epoch kullandık. Ancak aslında 75 epoch civarından sonra performansta bir artma olmamaktadır. Eğitimin 75 epoch civarında kesilmesi uygun olacaktır.

Şimdi modelin testini yapalım:

```

test_result = model.evaluate(test_dataset_x, test_dataset_y)
for i in range(len(test_result)):
    print(f'{model.metrics_names[i]} --> {test_result[i]}')

```

Şöyledir bir çıktı elde edilmişdir:

```

loss --> 8.037224769592285
mae --> 2.179290771484375

```

Mean Absolute Error değerinin 2.179 olduğunu görüyorsunuz. Bu değer gerçek değerle tahmin edilen değer arasındaki ortalama sapmayı belirtmektedir. Yani biz bir kestirim yaptığımızda gerçek değerle ortalama 2.179 kadar bir sapma ile karşılaşacağız.

Şimdi de predict işlemi yapalım:

```
predict_data = np.array([[4., 113., 95., 2228., 14., 71, 0, 0, 1]], dtype='float32')
predict_data = mms.transform(predict_data)
predict_result = model.predict(predict_data)
print(predict_result)
```

Predict işlemi yapılrken de kategorik verilerin yine one hot encoding dönüştürmesiyle kullanıldığına ve predict verilerinin yine aynı MinMaxScaler nesnesiyle normalize edildiğine dikkat ediniz. Buradan şu sonuç elde edilmiştir:

```
[[22.448292]]
```

Yukarıdaki örneğin tüm kodunu bütün olarak da vermek istiyoruz. Denemeleri bu koddan faydalananarak yapabilirsiniz:

```
import numpy as np

dataset = np.loadtxt('auto-mpg.data', usecols=range(8), dtype='object')
dataset = dataset[dataset[:, 3] != '?', :].astype('float32')

dataset_x = dataset[:, 1:]
dataset_y = dataset[:, 0]

from sklearn.preprocessing import OneHotEncoder, MinMaxScaler

ohe = OneHotEncoder(sparse=False)
ohe_data = ohe.fit_transform(dataset_x[:, 6].reshape(-1, 1))
dataset_x = np.delete(dataset_x, 6, axis=1)
dataset_x = np.append(dataset_x, ohe_data, axis=1)

from sklearn.model_selection import train_test_split

training_dataset_x, test_dataset_x, training_dataset_y, test_dataset_y =
train_test_split(dataset_x, dataset_y, test_size=0.2)

mms = MinMaxScaler()
mms.fit(dataset_x)

training_dataset_x = mms.transform(training_dataset_x)
test_dataset_x = mms.transform(test_dataset_x)

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential(name='Auto-MPG')
model.add(Dense(100, activation='relu', input_dim=dataset_x.shape[1], name='Hidden-1'))
model.add(Dense(100, activation='relu', name='Hidden-2'))
model.add(Dense(1, activation='linear', name='Output'))

model.summary()

model.compile(optimizer='adam', loss='mse', metrics=['mae'])
hist = model.fit(training_dataset_x, training_dataset_y, batch_size=32, epochs=75,
validation_split=0.2)

import matplotlib.pyplot as plt

figure = plt.gcf()
figure.set_size_inches((15, 5))
```

```

plt.title('Loss - Epoch Graphics')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.plot(range(1, len(hist.history['loss']) + 1), hist.history['loss'])
plt.plot(range(1, len(hist.history['val_loss']) + 1), hist.history['val_loss'])
plt.legend(['Loss', 'Validation Loss'])
plt.show()

figure = plt.gcf()
figure.set_size_inches((15, 5))
plt.title('Mean Absolute Error - Epoch Graphics')
plt.xlabel('Epoch')
plt.ylabel('Mean Absolute Error')
plt.plot(range(1, len(hist.history['mae']) + 1), hist.history['mae'])
plt.plot(range(1, len(hist.history['val_mae']) + 1), hist.history['val_mae'])
plt.legend(['Mean Absolute Error', 'Validation Mean Absolute Error'])
plt.show()

test_result = model.evaluate(test_dataset_x, test_dataset_y)
for i in range(len(test_result)):
    print(f'{model.metrics_names[i]} --> {test_result[i]}')

predict_data = np.array([[4., 113., 95., 2228., 14., 71, 0, 0, 1]], dtype='float32')
predict_data = mms.transform(predict_data)
predict_result = model.predict(predict_data)
print(predict_result)

```

Tabii uygulamada kestirim için değerler kullanıcı arayüzünden, varitabanlarından ya da CSV dosyalarından da alınabilir. Örneğin kestirilecek değerler "predict-data.csv" isimli bir dosyada aşağıdaki gibi bulunuyor olsun:

```

8,307,130,3504,12,70,1
8,350,165,3693,11.5,70,1
8,318,150,3436,11,70,1
8,304,150,3433,12,70,1
8,302,140,3449,10.5,70,2

```

Biz predict işlemini verileri bu dosyadan okuyarak da şöyle yapabiliyoruz:

```

predict_data = np.loadtxt('predict-data.csv', delimiter=',', dtype=np.float32)
ohe = OneHotEncoder(categories=[[1, 2, 3]], sparse=False)
ohe_data = ohe.fit_transform(predict_data[:, 6].reshape(-1, 1))
predict_data = np.delete(predict_data, 6, axis=1)
predict_data = np.append(predict_data, ohe_data, axis=1)
predict_data = mms.transform(predict_data)
predict_result = model.predict(predict_data)
print(predict_result)

```

Şöyledir bir çıktı elde edilmiştir:

```

[[15.033615]
[14.265699]
[14.969746]
[15.133997]
[15.067176]]

```

Yukarıdaki işlemde OneHotEncoder nesnesi yaratılırken categories parametresine kategorik değerler girilmiştir. Eğer bu yapılmazsa fit_transform metodu kendine verilen verilerdeki kategori sayısı kadar sütun oluşturmaktadır. Biz categories parametresinde üç kategori belirttiğimiz için fit_transform artık üç sütun oluşturacaktır.

Şimdi model bilgilerini saklamak isteyelim. Bunun için model sınıflarının save metodlarının kullanıldığını anımsayıınız:

```
model.save('autompg.h5')
```

Geri yükleme işleminin de nasıl yapıldığını biliyorsunuz:

```
from tensorflow.keras.models import load_model
model = load_model('autompg.h5')
```

Ancak burada bir noktaya dikkatinizi çekmek istiyoruz. Modeli yükledikten sonra predict işlemi yapmadan önce bizim yine "özellik ölçeklendirmesi (feature scaling)" yapmamız gereklidir. Halbuki bunun için oluşturmuş olduğumuz MinMaxScaler nesnesini biz herhangi biçimde diskte saklamadık. Pekiyi bunun için ne yapabilirdi? İşte -"Python Uygulamaları" kursunda da ele aldığımız gibi- aslında sınıf nesnelerini içlerindeki verilerle diske bir dosya içerisinde saklayabiliriz. Bu işleme nesne yönelimli dillerde "nesnelerin seri hale getirilmesi (object serialization)" denilmektedir. Python'da nesnelerin seri hale getirilmesi için pickle isimli bir modül bulunmaktadır. Bu modüldeki dump fonksiyonu nesneyi saklamak için load fonksiyonu da geri almak için kullanılmaktadır. Örneğin:

```
import pickle
with open('mms.dat', 'wb') as f:
    pickle.dump(mms, f)
```

Şimdi de "mms.dat" dosyasında saklamış olduğumuz nesne bilgilerini diskten geri yükleyerek predict işlemi yapalım:

```
import numpy as np
from tensorflow.keras.models import load_model
import pickle
model = load_model('autompg.h5')
with open('mms.dat', 'rb') as f:
    mms = pickle.load(f)
predict_data = np.array([[4., 113., 95., 2228., 14., 71, 0, 0, 1]], dtype='float32')
predict_data = ss.transform(predict_data)
predict_result = model.predict(predict_data)
print(predict_result)
```

Lojistik Olmayan Regresyon Modeli İçin "Boston Housing Prices" Örneği

Şimdi de lojistik olmayan regresyon için başka bir örnek üzerinde çalışacağımız Boston Housing Prices veri kümesi Boston'da 70'li yılların sonlarına doğru ev fiyatlarının tahmin edilmesi için toplanmış verilerden oluşmaktadır. Veri kümesi <https://www.kaggle.com/vikrishnan/boston-house-prices> adresinden indirilebilir. Dosyanın genel görünümü şöyledir:

0.00632	18.00	2.310	0	0.5380	6.5750	65.20	4.0900	1	296.0	15.30	396.90	4.98	24.00
0.02731	0.00	7.070	0	0.4690	6.4210	78.90	4.9671	2	242.0	17.80	396.90	9.14	21.60
0.02729	0.00	7.070	0	0.4690	7.1850	61.10	4.9671	2	242.0	17.80	392.83	4.03	34.70
0.03237	0.00	2.180	0	0.4580	6.9980	45.80	6.0622	3	222.0	18.70	394.63	2.94	33.40
0.06905	0.00	2.180	0	0.4580	7.1470	54.20	6.0622	3	222.0	18.70	396.90	5.33	36.20
0.02985	0.00	2.180	0	0.4580	6.4300	58.70	6.0622	3	222.0	18.70	394.12	5.21	28.70
0.08829	12.50	7.870	0	0.5240	6.0120	66.60	5.5605	5	311.0	15.20	395.60	12.43	22.90
0.14455	12.50	7.870	0	0.5240	6.1720	96.10	5.9505	5	311.0	15.20	396.90	19.15	27.10
0.21124	12.50	7.870	0	0.5240	5.6310	100.00	6.0821	5	311.0	15.20	386.63	29.93	16.50
0.17004	12.50	7.870	0	0.5240	6.0040	85.90	6.5921	5	311.0	15.20	386.71	17.10	18.90
0.22489	12.50	7.870	0	0.5240	6.3770	94.30	6.3467	5	311.0	15.20	392.52	20.45	15.00
0.11747	12.50	7.870	0	0.5240	6.0090	82.90	6.2267	5	311.0	15.20	396.90	13.27	18.90
0.09378	12.50	7.870	0	0.5240	5.8890	39.00	5.4509	5	311.0	15.20	390.50	15.71	21.70
0.62976	0.00	8.140	0	0.5380	5.9490	61.80	4.7075	4	307.0	21.00	396.90	8.26	20.40
0.63796	0.00	8.140	0	0.5380	6.0960	84.50	4.4619	4	307.0	21.00	380.02	10.26	18.20
0.62739	0.00	8.140	0	0.5380	5.8340	56.50	4.4986	4	307.0	21.00	395.62	8.47	19.90
1.05393	0.00	8.140	0	0.5380	5.9350	29.30	4.4986	4	307.0	21.00	386.85	6.58	23.10
0.78420	0.00	8.140	0	0.5380	5.9900	81.70	4.2579	4	307.0	21.00	386.75	14.67	17.50

Dosyada sütunların boşluk karakterleriyle birbirlerinden ayırdığını görüyorsunuz. Dosyanın son sütunu evin 1000 dolar cinsinden fiyatını belirtmektedir. Biz burada diğer sütun özelliklerinin neler olduğu üzerinde durmayacağımız. Ancak yukarıda verdigimiz "Kaggle" bağlantısından sütun özelliklerinin anlamlarını öğrenebilirsiniz. Biz yukarıdaki bağlantı yoluyla veri kümesi dosyasının "housing.csv" ismiyle çalışma dizini içerisinde indirdiğini varsayacağımız.

Dosyayı aşağıdaki gibi yükleyip eğitim ve test veri kümelerini ayırtılabiliriz:

```
import numpy as np

dataset = np.loadtxt('housing.csv')
dataset_x = dataset[:, :-1]
dataset_y = dataset[:, -1]

from sklearn.model_selection import train_test_split
training_dataset_x, test_dataset_x, training_dataset_y, test_dataset_y =
train_test_split(dataset_x, dataset_y, test_size=0.20)
```

Aslında bu veri kümesi test amacıyla makine öğrenmesi öğrencileri tarafından çok kullanıldığı için tensorflow.keras.datasets.boston_housing modülünde de hazır bir biçimde bulundurulmuştur. Modülün load_data fonksiyonu verileri bize ikili demetten oluşan ikili demet biçiminde vermektedir. Biz de aşağıdaki gibi verileri kullanıma hazır hale getirebiliriz:

```
from tensorflow.keras.datasets import boston_housing

(training_dataset_x, training_dataset_y), (test_dataset_x, test_dataset_y) =
boston_housing.load_data()
```

Veriler incelendiğinde bir özellik ölçeklendirmesi yapılması gerektiği görülmektedir. Veri kümesinin 3'üncü indeksli sütunu aslında 0 ve 1 değerlerinden oluşan kategorik bir sütundur. İkili kategoriler için one hot encoding işleminin gereklidğini anımsayınız. O halde bu veri kümesi için örnek modeli şöyle oluşturabiliriz.

```
from tensorflow.keras.datasets import boston_housing

(training_dataset_x, training_dataset_y), (test_dataset_x, test_dataset_y) =
boston_housing.load_data()

from sklearn.preprocessing import MinMaxScaler

mms = MinMaxScaler()
```

```

mms.fit(training_dataset_x)
training_dataset_x = mms.transform(training_dataset_x)
test_dataset_x = mms.transform(test_dataset_x)

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential()
model.add(Dense(100, input_dim=training_dataset_x.shape[1], activation='relu', name='Hidden-1'))
model.add(Dense(100, activation='relu', name='Hidden-2'))
model.add(Dense(1, activation='linear', name='Output'))

model.compile('rmsprop', loss='mse', metrics=['mae'])
hist = model.fit(training_dataset_x, training_dataset_y, epochs=300, validation_split=0.2,
batch_size=32)

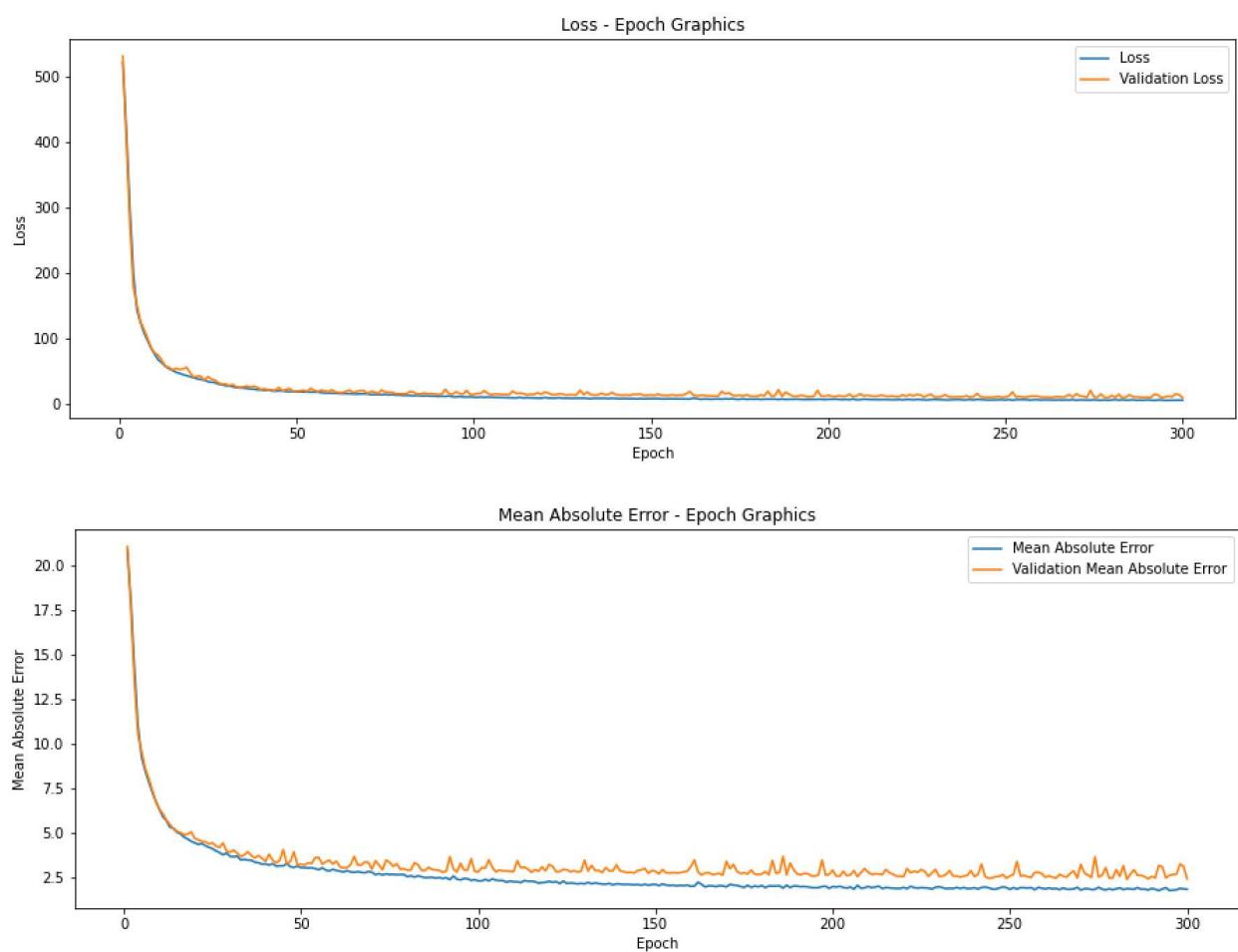
import matplotlib.pyplot as plt

figure = plt.gcf()
figure.set_size_inches((15, 5))
plt.title('Loss - Epoch Graphics')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.plot(range(1, len(hist.history['loss']) + 1), hist.history['loss'])
plt.plot(range(1, len(hist.history['val_loss']) + 1), hist.history['val_loss'])
plt.legend(['Loss', 'Validation Loss'])
plt.show()

figure = plt.gcf()
figure.set_size_inches((15, 5))
plt.title('Mean Absolute Error - Epoch Graphics')
plt.xlabel('Epoch')
plt.ylabel('Mean Absolute Error')
plt.plot(range(1, len(hist.history['mae']) + 1), hist.history['mae'])
plt.plot(range(1, len(hist.history['val_mae']) + 1), hist.history['val_mae'])
plt.legend(['Mean Absolute Error', 'Validation Mean Absolute Error'])
plt.show()

```

Burada 300 epoch uyguladık. Eğitim sırasındaki loss ve metrik grafiklerine bakalım:



Grafiklerden herhangi bir overfitting şüphesinin olmadığı görülmektedir. Ayrıca epoch işleminin 100 civarından sonra performansın pek değişmediğini görüyoruz. Artık modeli test edebiliriz:

```
eval_result = model.evaluate(test_dataset_x, test_dataset_y)
for i in range(len(eval_result)):
    print(f'{model.metrics_names[i]}: {eval_result[i]}')
```

Şöyledir bir çıktı elde edilmiş:

```
loss: 18.788570404052734
mae: 2.841968059539795
```

Hedeften ortalama sapmanın 2.84 civarında olduğunu görüyoruz. Şimdi de bir kestirim yapalım:

```
import numpy as np
```

```
predict_data = np.array([[18.0846, 0, 18.1, 0, 0.679, 6.434, 100, 1.8347, 24, 666, 20.2, 27.25, 29.05]])
predict_data = mms.transform(predict_data)
predict_result = model.predict(predict_data)
print(predict_result)
```

Şöyledir bir çıktı elde edilmiş:

```
[[9.333602]]
```

Yapay Sinir Ağlarında Çok Değişkenli (Multivariate) Regresyon Modelleri

Aslında sinir ağlarında tek değişkenli regresyon modelleri ile çok değişkenli (multivariate) regresyon modelleri arasında mimari olarak ve parametre belirleme bağlamında önemli bir fark yoktur. Yalnızca çok değişkenli modellerde çıktı katmanındaki nöronların sayısı bağımlı değişken sayısına kadar olmaktadır. Örneğin biz ev fiyatını tahmininde evin fiyatının yanı sıra kirasını da tahmin etmek isteyebiliriz. Tek yapacağımız şey çıktı katmanındaki nöron sayısını 1 yerine 2 yapmak olacaktır. Tabii eğitim ve test veri kümelerinde bu iki değerin bulunuyor olması gereklidir. Lojistik regresyon modelleri için "çok değişkenli (multivariate)" terimi yerine "çok etiketli (multilabel)" terimi daha çok tercih edilmektedir.

Yapay Sinir Ağlarında Lojistik Regresyon Modelleri

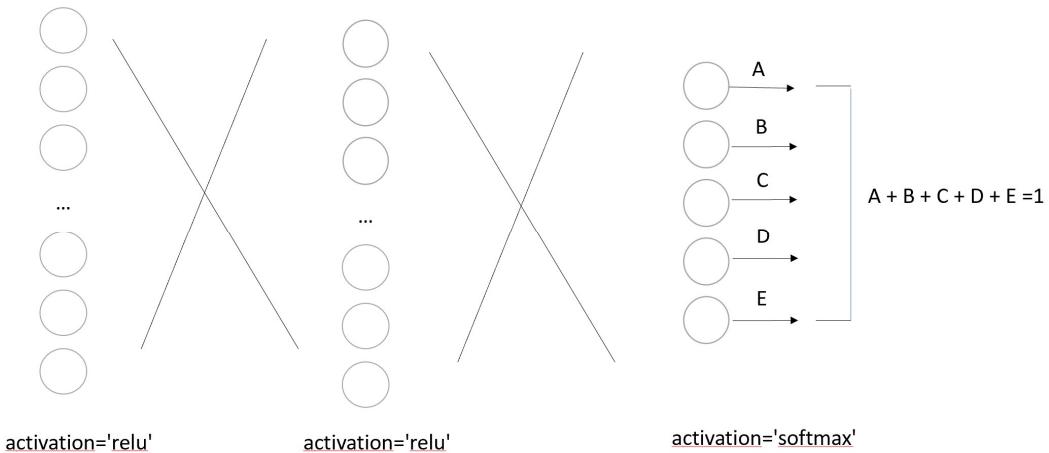
Daha önceden de belirttiğimiz gibi istatistikte sınıflandırma için "lojistik regresyon" terimi de kullanılmaktadır. Lojistik regresyon problemleri kendi aralarında "etiket (label)" ve "sınıf (class)" sayılarına göre grupperliliktedir. Etiket çıktıdaki değişken sayısını belirtmektedir. Örneğin modelin çıktısı "kişinin eğitim durumu" ve "obez olup olmadığına" ilişkin olsun. Burada iki tane etiket vardır. Tabii etiketler de kendi aralarında iki sınıfı ya da daha fazla sınıfı sahip olabilirler. Örneğin eğitim durumu beş sınıfı, obez olup olmama iki sınıfı temsil edilebilir. Bu durumda lojistik regresyon modellerini üç gruba ayıralım:

- 1) Tek etiketli iki sınıfı lojistik regresyon modelleri
- 2) Tek etiketli çok sınıfı Lojistik regresyon modelleri
- 3) Çok etiketli çok sınıfı lojistik regresyon modelleri

Şimdi bu modeller üzerinde biraz daha duralım.

1) Tek Etiketli-İkili (Single-Label Binary) Lojistik Regresyon Modelleri: Bu modeller tipik lojistik regresyon modelleridir. Daha önce yapmış olduğumuz diyabet modeli buna tipik bir örnektir. Bu modellerde yapay sinir ağının çıktı katmanında tek bir nöron bulunur. Bu nöronun çıktı değeri "doğru-yanlış", "var-yok", "hasta sağlam" gibi ikili bir değerdir. Daha önce de belirttiğimiz gibi tek etiketli ikili sınıflandırma modellerinde genellikle saklı katmanlardaki aktivasyon fonksiyonları "relu" olarak, çıktı katmanındaki aktivasyon fonksiyonu ise "sigmoid" olarak alınmaktadır. Anımsanacağı gibi sigmoid fonksiyonu 0 ile 1 arasında "S" şekli biçiminde bir değer vermektedir. Bu durumda çıktı katmanındaki nöronun çıkış değerinin "var-yok", "hasta-sağlıklı", "doğru-yanlış" gibi yorumlanması bu değerin 0.5 gibi bir eşik değerinden büyük ya da küçük olup olmamasına bakılarak yapılır. Daha önce de belirttiğimiz gibi tek etiketli ikili modellerdeki optimizasyon algoritması "adam" "sgd" ya da "rmsprop" "seçilebilir. loss fonksiyonu genellikle "binary_crossentropy" alınmaktadır. Metrik değerler için ise en çok tercih edilen "binary_accuracy" fonksiyonudur. Bu tür modellerde veri kümelerinde sütunlar arttıkça saklı katmanların ve saklı katmanlardaki sayılarının artırılması tavsiye edilmektedir. Bu tür modellerin denemelerine iki saklı katmanla başlayabilirsiniz. Duruma göre saklı katmanların sayılarını artırabilirsiniz. Saklı katmanlardaki nöron sayıları girdi katmanındaki nöron sayılarının iki katı biçiminde alınabilir. Duruma göre bu nöronlar artırılabilir. Yukarıdaki "diabetes.csv" örneğinde katmanlardaki nöron sayılarını 100 tutmuştur. Aslında bu modelde katmanlardaki nöron sayılarını 16 gibi bir değerde tutmak sonucu çok fazla etkilemeyecektir.

2) Tek Etiketli-Çok Sınıflı (Single-Label Multi-Class) Lojistik Regresyon Modelleri: Bu modellerin çıktıları kategoriktir fakat ikiden fazla sınıfından oluşmaktadır. Örneğin sinir ağı kişinin eğitim durumunu "ilkokul", "ortaokul", "lise", "üniversite" biçiminde tahmin etmek için oluşturulmuş olabilir. Ya da örneğin sinir ağı resmini aldığı meyvenin "elma", mı "armut" mu, "kayısı" mı olduğunu belirlemek için oluşturulmuş olabilir. Bu tür modellerde çıktı katmanındaki nöron sayısı toplam sınıf sayısı kadar olur. Örneğin modelimiz girdiyi A, B, C, D, E biçiminde sınıflandırıracak olsun. Bu modeldeki çıktı katmanındaki nöronlarının sayısı 5 olacaktır. Bu tür modellerde saklı katmanlar için yine genellikle "relu" aktivasyon fonksiyonu kullanılmaktadır. Ancak çıktı katmanındaki aktivasyon fonksiyonu "softmax" alınır. Softmax aktivasyon fonksiyonu çıktı nöronlarının (yani sınıfların) toplam olasılıklarının 1 değerini verdiği bir aktivasyon fonksiyonudur. Başka bir deyişle önceki örnekte çıktı katmanındaki her kategoriyi belirten nöronların çıkış değerleri toplamı 1 olacaktır.



Cıktı katmanındaki nöronların toplam değeri 1 olduğuna göre hangi sınıfın seçildiği nasıl anlaşılacaktır? İşte çok sınıflı bu tür modellerde hangi çıktı nöronunun değeri daha yüksekse o sınıfı seçtiği varsayılabılır. Numpy'da bir ndarray içerisindeki değerlerin en büyüğünün max fonksiyonuyla en büyük elemanın indeksinin ise maxarg fonksiyonuyla elde edildiğini anımsayınız. Örneğin çıktı katmanındaki nöron değerlerinin $A = 0.3$, $B = 0.2$, $C = 0.1$, $D = 0.35$ ve $E = 0.05$ biçiminde olduğunu ve "output" isimli numpy dizisi içerisinde bulunduğu varsayıalım:

```
In [8]: output
Out[8]: array([0.3 , 0.2 , 0.1 , 0.35, 0.05])

In [9]: np.sum(output)
Out[9]: 1.0

In [10]: np.max(output)
Out[10]: 0.35

In [11]: np.argmax(output)
Out[11]: 3
```

Burada np.argmax fonksiyonu ile üçüncü indeksli nöronun en yüksek değerde olduğu bulunmuştur.

Çok sınıflı modellerde en uygun loss fonksiyonu "categorical_crossentropy"dir. Optimizasyon algoritması için yine "adam", "sgd" ya da "rmsprop" kullanılabilir. Tek etiketli çok sınıflı modeller için metrik fonksiyonu da genellikle "categorical_accuracy" seçilmektedir.

Çok sınıflı modellerde deneme yine iki saklı katmanla başlatılabilir. Ancak sınıf sayısı çok fazlaysa saklı katmanların sayısını artırmak gerekebilir. Fazla sınıflı modellerin yüksek miktarda verilerle eğitilmesi önerilmektedir.

3) Çok Etiketli (Multi-Label) Lojistik Regresyon Modelleri: Çok etiketli modeller bağımlı değişkenin birden fazla olduğu modellerdir. Yukarıda da belirttiğimiz gibi istatistikte bu tür modellere "çok değişkenli (multivariate)" modeller de denilmektedir. Çok etiketli modellerdeki her bir değişken (yani etiket) iki ya da daha fazla sınıfından oluşabilmektedir. Örneğin biz bir sinir ağı ile çeşitli biyomedikal tetkik değerlerinden hareketle hem kişinin şeker hastası olup olmadığını hem de kalp hastalıklarına yatkınlık derecesini tahmin etmek isteyebiliriz. Kişiňin şeker hastası olup olmadığı iki sınıflı bir çıktı oluştururken, kalp hastalıklarına yatkınlık derecesi üç sınıflı bir çıktı oluşturabilir.

Aşağıdaki tabloda hangi lojistik regresyon modellerinde hangi parametrik değerlerin seçileceği özet olarak belirtilmiştir:

Sınıflandırma Model	Optimizasyon Algoritması	Loss Fonksiyonu	Saklı Katman Aktivasyon Fonksiyonları	Cıktı Katmanı Aktivasyon Fonksiyonu
---------------------	--------------------------	-----------------	---------------------------------------	-------------------------------------

Tek Etiketli İki Sınıflı Modeller	'rmsprop', 'adam', 'sgd'	'binary_crossentropy'	'relu'	'sigmoid'
Tek Etiketli Çok Sınıflı Modeller	'rmsprop', 'adam', 'sgd'	'categorical_crossentropy'	'relu'	'softmax'
Çok Etiketli Modeller	'rmsprop', 'adam', 'sgd'	İki Sınıflı Çıktılar İçin 'binary_crossentropy', Çok Sınıflı Çıktılar İçin 'categorical_crossentropy'	'relu'	İki Sınıflı Çıktılar İçin 'sigmoid', Çok Sınıflı Çıktılar İçin 'softmax'

Pekiyi birden fazla sınıfın içeresine girebilen sınıflandırma modelleri için ne yapılabilir? Örneğin bir müzik parçasının türünü sınıflandırmak isteyelim. Parçanın türü "pop", "rock", "jazz", "slow", "rap", "new wave" gibi kategorilerin birden fazlasına sahip olabilsin. Bu nasıl bir sınıflandırma modelidir? Bu model aslında çok etiketli bir model olarak düşünülebilir. Yani aslında burada toplamda 6 etiket vardır. Bu 6 etiket "var", "yok" biçiminde iki sınıf içermektedir. O halde bu model için 6 çıktıya sahip olan bir ağ oluşturulabilir. Bu 6 çıktılarının hepsi de "sigmoid" aktivasyon fonksiyonuna sahip olabilir. Diğer bir seçenek de problemi tek etiketli çok sınıflı olarak modellemektir. Bu durumda çıktı katmanında yine 6 nöron bulunur fakat çıktı katmanın aktivasyon fonksiyonu "softmax" alınır. Böylece en yüksek oranlı çıktıları veren birden fazla sınıf alınabilir. Veri analistleri daha bu tür durumlarda çok etiketli iki sınıfı model oluşturmayı tercih etmektedir.

İki Sınıflı Lojistik Regresyon Modeli İçin IMDB Örneği

Daha önce tek etiketli iki sınıflı lojistik regresyon modeli için "diabetes" örneğini vermiştık. O örnekte 8 biyomedikal tetkik sonucuna göre kişinin şeker hastası olup olmadığını tahmin etmeye çalışıyorduk. Şimdi de IMDB isimli tek etiketli iki sınıflı başka bir örnek üzerinde çalışacağız.

IMDB veri kümesi filmler hakkındaki yorumlardan oluşmaktadır. Her yorum "olumlu (pozitif)" ya da "olumsuz (negative)" biçimde değerlendirilmektedir. Biz burada içeriğe bakarak bir yorumun olumlu mu yoksa olumsuz mu olduğunu tahmin edecek bir model üzerinde çalışacağız. IMDB veri kümesi CSV dosyası olarak <https://www.kaggle.com/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews> adresinden indirilebilir. Bu CSV dosyası iki sütünden oluşmaktadır. Birinci sütundan yorum yazısı ikinci sütundan yorumun olumlu mu olumsuz mu olduğunu belirten "positive" ya da "negative" yazısı vardır. Yorum yazısı iki tırnak içerisine alınmıştır. Dosyanın görünümü şöyledir:

```
review,sentiment
"One of the other reviewers has mentioned that after watching just 1 Oz episode you'll be hooked. They are right, as this is exactly what happened with me.<br />"A wonderful little production. <br /><br />The filming technique is very unassuming- very old-time-BBC fashion and gives a comforting, and sometimes discomfort "I thought this was a wonderful way to spend time on a too hot summer weekend, sitting in the air conditioned theater and watching a light-hearted comedy. The p "Basically there's a family where a little boy (Jake) thinks there's a zombie in his closet & his parents are fighting all the time.<br /><br />This movie is si "Petter Mattei's ""Love in the Time of Money"" is a visually stunning film to watch. Mr. Mattei offers us a vivid portrait about human relations. This is a movi "Probably my all-time favorite movie, a story of selflessness, sacrifice and dedication to a noble cause, but it's not preachy or boring. It just never gets old "I sure would like to see a resurrection of a updated Seabunt series with the tech they have today it would bring back the kid excitement in me.I grew up on bi "This show was an amazing, fresh & innovative idea in the 70's when it first aired. The first 7 or 8 years were brilliant, but things dropped off after that. By "Encouraged by the positive comments about this film on here I was looking forward to watching this film. Bad mistake. I've seen 950+ films and this is truly or "If you like original gut wrenching laughter you will like this movie. If you are young or old then you will love this movie, hell even my mom liked it.<br /><br />"Phil the Alien is one of those quirky films where the humour is based around the oddness of everything rather than actual punchlines.<br /><br />At first it wa "I saw this movie when I was about 12 when it came out. I recall the scariest scene was the big bird eating men dangling helplessly from parachutes right out of "So im not a big fan of Boll's work but then again not many are. I enjoyed his movie Postal (maybe im the only one). Boll apparently bought the rights to use Fi "The cast played Shakespeare.<br /><br />Shakespeare lost.<br /><br />I appreciate that this is trying to bring Shakespeare to the masses, but why ruin somethir This a fantastic movie of three prisoners who become famous. One of the actors is george clooney and I'm not a fan but this roll is not bad. Another good thing "Kind of drawn in by the erotic scenes, only to realize this was one of the most amateurish and unbelievable bits of film I've ever seen. Sort of like a high sc "Some films just simply should not be remade. This is one of them. In and of itself it is not a bad film. But it fails to capture the flavor and the terror of i "This movie made it into one of my top 10 most awful movies. Horrible. <br /><br />There wasn't a continuous minute where there wasn't a fight with one monster "I remember this film,it was the first film i had watched at the cinema the picture was dark in places i was very nervous it was back in 74/75 my Dad took me my "An awful film! It must have been up against some real stinkers to be nominated for the Golden Globe. They've taken the story of the first famous female Renais: "After the success of Die Hard and it's sequels it's no surprise really that in the 1990s, a glut of 'Die Hard on a ....' movies cashed in on the wrong guy, w "I had the terrible misfortune of having to view this ""b-movie"" in it's entirety.<br /><br />All I have to say is--- save your time and money!!! This has got "What an absolutely stunning movie, if you have 2.5 hrs to kill, watch it, you won't regret it, it's too much fun! Rajnikant carries the movie on his shoulders: "First of all, let's get a few things straight here: a) I AM an anime fan- always has been as a matter of fact (I used to watch Speed Racer all the time in Pres This was the worst movie I saw at WorldFest and it also received the least amount of applause afterwards! I can only think it is receiving such recognition base "The Karen Carpenter Story shows a little more about singer Karen Carpenter's complex life. Though it fails in giving accurate facts, and details.<br /><br />C ""The Cell"" is an exotic masterpiece. a dizzvine trip into not only the vast mind of a serial killer. but also into one of a verv talented director. This is <
```

Yazı içerisinde satır başı için HTML `
` kullanıldığına dikkat ediniz. Burada her satırda tek bir yorum ve sonuç olduğu için ve yorum yazısı da uzun olduğu için sonuç kısmı görülemiyor. Şimdi sözcük sarmalamayı (word wrap) etkin hale getirip görüntüyü bir kez daha verelim:

review,sentiment
"One of the other reviewers has mentioned that after watching just 1 Oz episode you'll be hooked. They are right, as this is exactly what happened with me.
>
The first thing that struck me about Oz was its brutality and unflinching scenes of violence, which set in right from the word GO. Trust me, this is not a show for the faint hearted or timid. This show pulls no punches with regards to drugs, sex or violence. Its is hardcore, in the classic use of the word.
>
It is called OZ as that is the nickname given to the Oswald Maximum Security State Penitentiary. It focuses mainly on Emerald City, an experimental section of the prison where all the cells have glass fronts and face inwards, so privacy is not high on the agenda. Em City is home to many..Aryans, Muslims, gangstas, Latinos, Christians, Italians, Irish and more....so scuffles, death stares, dodgy dealings and shady agreements are never far away.
>
I would say the main appeal of the show is due to the fact that it goes where other shows wouldn't dare. Forget pretty pictures painted for mainstream audiences, forget charm, forget romance..OZ doesn't mess around. The first episode I ever saw struck me as so nasty it was surreal, I couldn't say I was ready for it, but as I watched more, I developed a taste for Oz, and got accustomed to the high levels of graphic violence. Not just violence, but injustice (crooked guards who'll be sold out for a nickel, inmates who'll kill on order and get away with it, well mannered, middle class inmates being turned into prison bitches due to their lack of street skills or prison experience) Watching Oz, you may become comfortable with what is uncomfortable viewing...thats if you can get in touch with your darker side.",positive

"A wonderful little production.

The filming technique is very unassuming- very old-time-BBC fashion and gives a comforting, and sometimes disconcerting, sense of realism to the entire piece.

The actors are extremely well chosen- Michael Sheen not only ""has got all the polaris"" but he has all the voices down pat too! You can truly see the seamless editing guided by the references to Williams' diary entries, not only is it well worth the watching but it is a terrifically written and performed piece. A masterful production about one of the great master's of comedy and his life.

The realism really comes home with the little things: the fantasy of the guard which, rather than use the traditional 'dream' techniques remains solid then disappears. It plays on our knowledge and our senses, particularly with the scenes concerning Orton and Halliwell and the sets (particularly of their flat with Halliwell's murals decorating every surface) are terribly well done.",positive

"I thought this was a wonderful way to spend time on a too hot summer weekend, sitting in the air conditioned theater and watching a light-hearted comedy. The plot is simplistic, but the dialogue is witty and the characters are likable (even the well bread suspected serial killer). While some may be disappointed when they realize this is not Match Point 2: Risk Addiction, I thought it was proof that Woody Allen is still fully in control of the style many of us have grown to love.

This was the most I'd laughed at one of Woody's comedies in years (dare I say a decade?). While I've never been impressed with Scarlet Johanson, in this she managed to tone down her ""sexy"" image and jumped right into a average, but spirited young woman.

This may not be the crown jewel of his career, but it was wittier than ""Devil Wears Prada"" and more interesting than ""Superman"" a great comedy to go see with friends.",positive

"Basically there's a family where a little boy (Jake) thinks there's a zombie in his closet & his parents are fighting all the time.

This movie is slower than a soap opera... and suddenly, Jake decides to become Rambo and kill the zombie.

OK, first of all when you're going to make a film you must Decide if its a thriller or a drama! As a drama the movie is watchable. Parents are divorcing & arguing like in real life. And then we have Jake with his closet which totally ruins all the film! I expected to see a BOOGYMAN similar movie, and instead i watched a drama with some meaningless thriller spots.

3 out of 10 just for the well playing parents & descent dialogs. As for the shots with Jake: just ignore them.",negative

"Petter Mattei's ""Love in the Time of Money"" is a visually stunning film to watch. Mr. Mattei offers us a vivid portrait about human relations. This is a movie that seems to be telling us what money, power and success do to people in the different situations we encounter.

This being a variation on the Arthur Schnitzler's play about the same theme, the director transfers the action to the present time New York where all these different characters meet and connect. Each one is connected in one way. or another to the next person. but no one seems to know the previous point of contact. Stylishv. the film has a

Tırnaklanmış içeriğin numpy.load.txt fonksiyonu ile okunması biraz zahmetlidir. Bu tür dosyaların okunması pandas.read_csv fonksiyonu ile çok daha kolay yapılabilir:

```
import pandas as pd

df = pd.read_csv('imdb.csv')
```

Bildiğiniz gibi yapay sinir ağları tamamen sayısal düzeydeki verilerle işlemler yapmaktadır. O halde bizim yorumdaki sözcükleri bir biçimde sayılarla ifade etmemiz gereklidir. Bunun bir yolu yorumlardaki tüm sözcükleri, anahtarlar sözcüklerden değerler de onların indeks numaralarından oluşan bir sözlükte tutmak olabilir. Bunu sağlamak için veri kümesindeki tüm yorumlardaki tüm sözcükler bir sözlüğe yerleştirilebilir. Sözlükteki her sözcüğe bir indeks numarası karşı getirilebilir. Bir yorum da böylece sözcüklerden değil indeks numaralarından oluşturulmuş olur. Şimdi önce DataFrame nesnesi içerisindeki tüm yazıları sözcüklere ayırip bir yeniden DataFrame içerisinde yerlestirelim. Bu sırada da tüm sözcükleri tek olacak biçimde bir kümeye toplayalım. Bunu yapmanın çok yolu olabilir. Ancak biz dosyayı bir kez okumak için read_csv fonksiyonunda okuma sırasında dönüştürme yapmayı tercih edeceğiz:

```
import numpy as np
import re

class WordConverter:
    def __init__(self):
        self.word_set = set()

    def __call__(self, text):
        words = re.findall("[a-zA-Z0-9'-]+", text.lower())
        self.word_set.update(words)

        return np.array(words)

    def get_word_dict(self):
        return {word: index for index, word in enumerate(wc.word_set)}

import pandas as pd

wc = WordConverter()
df = pd.read_csv('imdb.csv', converters={0: wc})
word_dict = wc.get_word_dict()
```

Bu işlemden sonra df içerisinde artık ilk sütunda küçük harfe dönüştürülmüş biçimde yazılarından oluşan bir numpy dizisi elde edilecektir. DataFrame nesnesinin ilk satırını aşağıda görüyorsunuz:

```
In [77]: df.iloc[0]
Out[77]:
review      [one, of, the, other, reviewers, has, mentione...
sentiment                           positive
Name: 0, dtype: object
```

WordConverter sınıfının get_word_dict isimli metodunun tüm yorumlardaki tüm sözcükleri içeren bir sözlük nesnesi verdiğine dikkat ediniz. word_dict sözlüğünden 20 eleman yazdırıralım:

```
In [47]: dict(itertools.islice(word_dict.items(), 20))
Out[47]:
{'stewert': 0,
 'theatre-style': 1,
 'metabolismic': 2,
 'blaring': 3,
 'right-and-good': 4,
 'ralph-bernadette': 5,
 'ferrero': 6,
 'heterosexuals': 7,
 'reinvents': 8,
 "fargo)": 9,
 'secondtime': 10,
 'post-atomic': 11,
 'bulletins': 12,
 'retains': 13,
 "'gahden)": 14,
 "allows)": 15,
 'hedge-hog-thing': 16,
 'levels': 17,
 'pined': 18,
 'goire': 19}
```

Şimdi de word_dict sözlüğünün uzunluğuna bakalım:

```
In [51]: len(word_dict)
Out[51]: 156840
```

Şimdi de tüm yazıları indeks numaralarıyla ifade edelim:

```
for i in range(len(df)):
    df.iloc[i, 0] = np.array([word_dict[word] for word in df.iloc[i, 0]])
```

DataFrame nesnesinin ilk satırına bakınız:

```
In [59]: df.iloc[0]
Out[59]:
review      [126434, 128794, 29339, 17370, 57739, 8688, 76...
sentiment                           positive
Name: 0, dtype: object
```

Göründüğü gibi artık yapılan yorum yazı olmaktan çıkmış bir sayı dizisi haline dönüştürülmüştür. Böyle bir sayı dizisini yazıya dönüştürmek isterseniz yukarıda oluşturmuş olduğumuz word_dict sözlüğünü kullanamazsınız. Bu sözlüğü anahtar değer bağlamında ters çevirmek gerekir:

```
rev_word_dict = {index: word for word, index in word_dict.items()}
```

Şimdi artık yeniden yorumu yazışal biçimde dönüştürebiliriz. Örneğin ilk yorumu yazışal biçimde dönüştürmek isteyelim:

```
text = ' '.join([rev_word_dict[index] for index in df.iloc[0, 0]])
print(text)
```

Şöyledir bir çıktı elde edilmiş:

one of the other reviewers has mentioned that after watching just 1 oz episode you'll be hooked they are right as this is exactly what happened with me br br the first thing that struck me about oz was its brutality and unflinching scenes of violence which set in right from the word go trust me this is not a show for the faint hearted or timid this show pulls no punches with regards to drugs sex or violence its is hardcore in the classic use of the word br br it is called oz as that is the nickname given to the oswald maximum security state penitentiary it focuses mainly ...

Ancak burada önemli bir sorun üzerinde durmak istiyoruz. Bu örnekte her yorumdaki sözcük sayıları farklıdır. İnsanlar yorumlarını değişik uzunlukta yapabilmektedir. Bu ise yapay sinir ağı modeline uygun değildir. Çünkü yapay sinir ağlarının çalışması için her satırın eşit sayıda sütun bilgisine (yani girdiye) sahip olması gereklidir. Peki bu durum IMDB örneğinde nasıl sağlanabilir? İşte bunun iki yolu olabilir: Birincisi yorumlardaki sözcük sayılarını eşit hale getirmektir (kısa olanları atıp uzun olanları kırmak). İkincisi de tüm yorumlardaki tüm sözcükleri temel olarak eşit uzunlukta bir bool vektör oluşturmaktır. Yorumlardaki yazıları kırmak iyi bir fikir değildir. Çünkü bu durumda yorumlar anlamsızlaşabilir ya da yorumlar yanlış anlaşılabilir. Tüm yorumları eşit uzunlukta bool vektörlerle ifade etmek daha iyi bir çözümdür. Bool vektör yönteminde her yorum için tüm yorumlardaki tüm sözcüklerin sayısı kadar bool bir vektör oluşturulur. Yorum içerisinde geçen sözcükler 1 ile geçmeyen sözcükler 0 ile kodlanır. Örneğin bir yorumda 100 sözcük bulunuyor olsun. Yorumlardaki tüm sözcük sayısının da 156840 olduğunu biliyoruz. Bu durumda biz her yorum için 156840'luk bir bool vektör oluştururuz. Bu 156840'luk bool vektördeki 100 elemanı 1 diğerlerini 0 yaparız. 1 olan vektör elemanları yorumda geçen sözcüklere karşı gelen indeksteki elemanlardır. Biz burada bu yöntemi izleyeceğiz. Yorumların bu biçimde bool bir vektörle ifade edilmesi aynı zamanda sözcüklerin sıra numaralarının sıralı ölçüye ilişkinmiş gibi değerlendirilmesine de engel olacaktır. Yani bu işlem bir çeşit "binary encoding" dönüştürmesi anlamına da gelmektedir. Şimdi yukarıdaki DataFrame nesnesini bu biçimde dönüştürecek bir fonksiyon yazalım:

```
def vectorize(iterable, colsizes):
    result = np.zeros((len(iterable), colsizes), dtype=np.int8)
    for index, vals in enumerate(iterable):
        result[index, vals] = 1

    return result
```

Fonksiyon sözcüklerin indeks numaralarından oluşan dolaşılabilir nesneyi ve toplam sözcük sayısını parametre olarak alıp hedeflenen bool matrise geri dönmektedir. Fonksiyonun içerisinde önce içi 0'larla doldurulmuş hedef matris oluşturulmuş, sonra bunun ilgili elemanları 1 yapılmıştır. Numpy'da indeksleme işleminde birden fazla indeks değeri belirtilebildiğini biliyorsunuz. Şimdi yukarıdaki veri kümесini bu biçimde bool bir matrise dönüştürelim:

```
dataset = df['review'].to_list()
dataset_x = vectorize(dataset, len(word_dict))
```

Buradan elde edilen dataset_x matrisinin toplam yorum sayısı kadar satıldan, toplam sözcük sayısı kadar sütundan oluştuğuna dikkat ediniz. Matrisin her elemanı 1 byte olduğuna göre bu matris çok büyük bir yer kaplamaktadır. Görünümü aşağıdaki gibidir:

```
In [62]: dataset_x
Out[62]:
array([[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]], dtype=int8)
```

```
In [63]: dataset_x.shape
Out[63]: (50000, 156840)
```

```
In [64]: np.sum(dataset_x[0])
Out[64]: 189
```

Yukarıdaki örnekte matrisin ilk satırında yalnızca 189 değerin 1, geri kalanlarının 0 olduğunu görüyorsunuz. Bu ilk yorumun 189 farklı sözcükten oluştuğu anlamına gelmektedir:

```
In [66]: len(np.unique(df.iloc[0, 0]))
Out[66]: 189
```

Artık eğitimin sonuçlarını da ayrı bir vektörde toplayabiliyoruz:

```
df.iloc[df.iloc[:, 1] == 'positive', 1] = 1
df.iloc[df.iloc[:, 1] == 'negative', 1] = 0
dataset_y = df.iloc[:, 1].to_numpy(dtype=np.int8)
```

Artık modelimizi kurabiliyoruz. Modelimiz ikş sınıfı bir lojistik regresyon modelidir. Biz daha önce "diabetes" örneğinde bu biçimde bir model üzerinde çalıştık. Modelimizi yine iki katmanlı olarak tasarlayacağız:

```
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
from sklearn.model_selection import train_test_split

training_dataset_x, test_dataset_x, training_dataset_y, test_dataset_y =
train_test_split(dataset_x, dataset_y,
test_size=0.20)

model = Sequential(name='IMDB')
model.add(Dense(100, input_dim=training_dataset_x.shape[1], activation='relu', name='Hidden-1'))
model.add(Dense(100, activation='relu', name='Hidden-2'))
model.add(Dense(1, activation='sigmoid', name='Output'))
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['binary_accuracy'])
hist = model.fit(training_dataset_x, training_dataset_y, batch_size=64, epochs=5,
validation_split=0.2)

import matplotlib.pyplot as plt

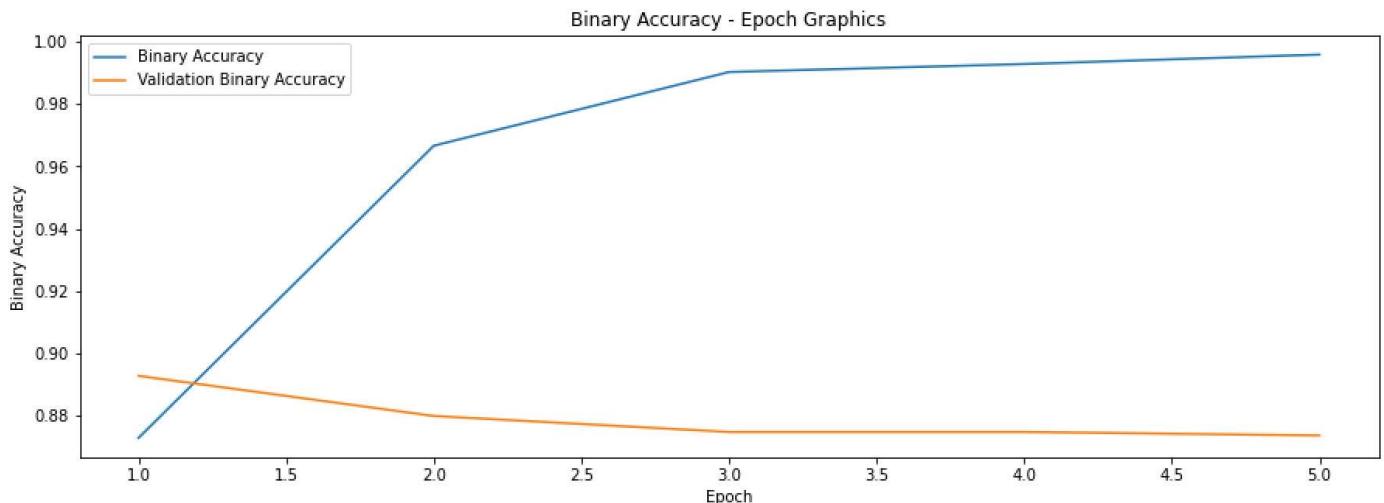
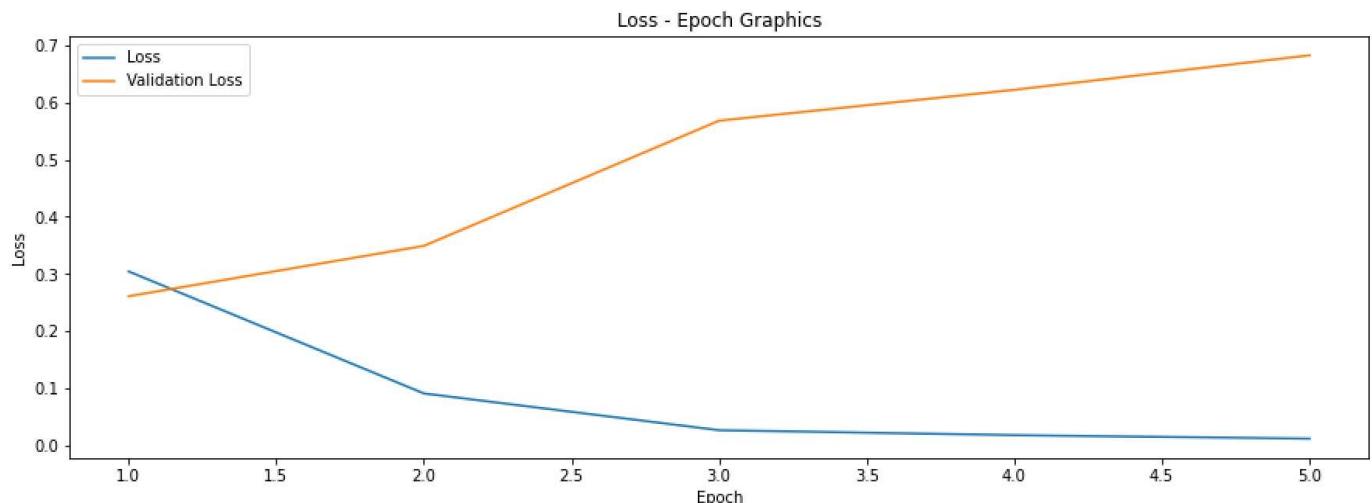
figure = plt.gcf()
figure.set_size_inches((15, 5))
plt.title('Loss - Epoch Graphics')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.plot(range(1, len(hist.history['loss']) + 1), hist.history['loss'])
plt.plot(range(1, len(hist.history['val_loss']) + 1), hist.history['val_loss'])
plt.legend(['Loss', 'Validation Loss'])
plt.show()
```

```

figure = plt.gcf()
figure.set_size_inches((15, 5))
plt.title('Binary Accuracy - Epoch Graphics')
plt.xlabel('Epoch')
plt.ylabel('Binary Accuracy')
plt.plot(range(1, len(hist.history['binary_accuracy']) + 1), hist.history['binary_accuracy'])
plt.plot(range(1, len(hist.history['val_binary_accuracy']) + 1),
hist.history['val_binary_accuracy'])
plt.legend(['Binary Accuracy', 'Validation Binary Accuracy'])
plt.show()

```

Loss ve metrik grafikleri şöyle elde edilmiştir:



Burada birinci epoch'tan sonra overfit durumunun olduğunu görüyoruz. Bu durumda eğitim 1 epoch'ta kesilebilir. Burada eğitim veri kğmelerinin oldukça büyük olduğuna dikkatinizi çekmek istiyoruz:

In [69]: `training_dataset_x.shape`
Out[69]: `(40000, 156840)`

In [70]: `training_dataset_y.shape`
Out[70]: `(40000,)`

Şimdi de test işlemini yapalım:

```

eval_result = model.evaluate(test_dataset_x, test_dataset_y)
for i in range(len(eval_result)):
    print(f'{model.metrics_names[i]}: {eval_result[i]}')

```

Şöyledir bir çıktı elde edilmiştir:

```
loss: 0.6228826642036438
binary_accuracy: 0.883899986743927
```

%88.3 lük bir başarının elde edildiğini görüyorsunuz. Şimdi tüm kodu bir bütün olarak yeniden verelim. Ancak eğitimde 1 epoch kullanacağımız için metrik grafiklerini koddan çıkartıyoruz:

```
import numpy as np
import re

class WordConverter:
    def __init__(self):
        self.word_set = set()

    def __call__(self, text):
        words = re.findall("[a-zA-Z0-9'-]+", text.lower())
        self.word_set.update(words)

        return np.array(words)

    def get_word_dict(self):
        return {word: index for index, word in enumerate(wc.word_set)}

import pandas as pd

wc = WordConverter()
df = pd.read_csv('imdb.csv', converters={0: wc})
word_dict = wc.get_word_dict()

for i in range(len(df)):
    df.iloc[i, 0] = np.array([word_dict[word] for word in df.iloc[i, 0]])

def vectorize(iterable, colszie):
    result = np.zeros(len(iterable), colszie, dtype=np.int8)
    for index, vals in enumerate(iterable):
        result[index, vals] = 1

    return result

dataset = df['review'].to_list()
dataset_x = vectorize(dataset, len(word_dict))

df.iloc[df.iloc[:, 1] == 'positive', 1] = 1
df.iloc[df.iloc[:, 1] == 'negative', 1] = 0
dataset_y = df.iloc[:, 1].to_numpy(dtype=np.int8)

from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
from sklearn.model_selection import train_test_split

training_dataset_x, test_dataset_x, training_dataset_y, test_dataset_y =
train_test_split(dataset_x, dataset_y,
test_size=0.20)

model = Sequential(name='IMDB')
model.add(Dense(100, input_dim=training_dataset_x.shape[1], activation='relu', name='Hidden-1'))
model.add(Dense(100, activation='relu', name='Hidden-2'))
model.add(Dense(1, activation='sigmoid', name='Output'))
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['binary_accuracy'])
hist = model.fit(training_dataset_x, training_dataset_y, batch_size=64, epochs=1,
```

```

validation_split=0.2)

eval_result = model.evaluate(test_dataset_x, test_dataset_y)
for i in range(len(eval_result)):
    print(f'{model.metrics_names[i]}: {eval_result[i]}')

```

Şimdi de kurulan model üzerinde kestirim yapalım. Kötü bir yorum oluşturalım:

```

predict_text = "A bad production. It's a bad movie."

```

Oluşturduğumuz bu yorumu ağımıza sokmadan önce eğitimde kullandığımız biçimde dönüştürmemiz gerekiyor. Yorum yazısını sözcüklerine ayırıp bool bir vektöre dönüştüren küçük bir fonksiyon yazabiliriz:

```

def prepare_review(predict_text):
    predict_words = re.findall("[a-zA-Z0-9'-]+", predict_text.lower())
    result = np.zeros(len(predict_words), dtype=np.int8)
    for i in range(len(predict_words)):
        result[i] = word_dict[predict_words[i]]

    return vectorize([result], len(word_dict))

```

Fonksiyonun global word_dict değişkenini kullandığına dikkat ediniz. Genel olarak fonksiyonların böyle global değişkenleri kullanması iyi bir teknik değildir. Ancak biz burada programın organizasyonundan ziyade modelin oluşturulması ve kullanılması üzerine odaklıyoruz. Şimdi kestirimimizi yapalım:

```

predict_data = prepare_review(predict_text)
predict_result = model.predict(predict_data)

if predict_result[0, 0] > 0.5:
    print(f'Olumlu Yorum: {predict_result[0, 0]}')
else:
    print(f'Olumsuz Yorum: {predict_result[0, 0]}')

```

Şöyledir bir çıktı elde edilmiştir:

```

Olumsuz Yorum: 0.41155344247817993

```

Şimdi birkaç değişik yorum için kestirimde bulunalım:

```

predict_texts = ["A nice film. But some parts are bad", "Neither a good nor a bad movie. Some parts are exaggerated.", "it's a nice film. I recommend you to watch the movie"]

for text in predict_texts:
    predict_data = prepare_review(text)
    predict_result = model.predict(predict_data)

    if predict_result[0, 0] > 0.5:
        print(f'Olumlu Yorum: {predict_result[0, 0]}')
    else:
        print(f'Olumsuz Yorum: {predict_result[0, 0]}')

```

Şöyledir bir sonuç elde edilmiştir:

```

Olumsuz Yorum: 0.15564844012260437
Olumsuz Yorum: 0.16812017560005188
Olumlu Yorum: 0.9306429624557495

```

Bu örnekte öğrenme sürecinde sözcüklerin hangi bağlamda kullanıldıklarının bir öneminin olmadığına dikkatinizi çekmek istiyoruz. Yani örneğin aslında yorumlardaki sözcüklerin yerleri değiştirilse de bunun eğitim üzerinde bir etkisi olmayacağıdır. Çünkü modelimizde sözcüklerin nerede olduğunu bir önemi yoktur. Yalnızca onların bir arada

aynı yazı içerisinde bulunuyor olmalarının önemi vardır. Tabii sözcüklerin bağımsal etkilerinin göz ardı edilmesi aslında öğrenme sürecini olumsuz etkileyebilecek bir süreç olarak değerlendirilebilir. İleride bağımsal etkinin nasıl oluşturulacağına yönelik modeller ele alınacaktır.

Aslında Keras'ta IMDB veri kümesi hazır bir biçimde tensorflow.keras.datasets paketi içerisindeki imdb modülünde bulunmaktadır. Veri kümesi modülün load_data fonksiyonu ile yüklenmektedir. load_data fonksiyonunun num_words isimli parametresi en çok kullanılan n tane sözcüğü yorumlarda bulunduracak biçimde veri kümelerini bize vermektedir. (Yani örneğin biz num_word değerini 10000 yaptığımızda tüm yorumların içerisindeki tüm farklı sözcüklerin sayısı 10000 tane olmaktadır.) Örneğin:

```
from tensorflow.keras.datasets import imdb

VOCAB_SIZE = 10000

(training_dataset_x, training_dataset_y), (test_dataset_x, test_dataset_y) =
imdb.load_data(num_words=VOCAB_SIZE)
```

Burada training_dataset_x ve test_dataset_x birer ndarray nesnesidir. Ancak bu ndarray nesnesi list nesnelerinden oluşmaktadır. Örneğin:

```
In [4]: training_dataset_x
Out[4]:
array([list([1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 458, 4468, 66, 3941, 4, 173, 36, 256, 5, 25,
100, 43, 838, 112, 50, 670, 2, 9, 35, 480, 284, 5, 150, 4, 172, 112, 167, 2, 336, 385, 39, 4, 172,
4536, 1111, 17, 546, 38, 13, 447, 4, 192, 50, 16, 6, 147, 2025, 19, 14, 22, 4, 1920, 4613, 469, 4, 22,
71, 87, 12, 16, 43, 530, 38, 76, 15, 13, 1247, 4, 22, 17, 515, 17, 12, 16, 626, 18, 2, 5, 62, 386, 12,
8, 316, 8, 106, 5, 4, 2223, 5244, 16, 480, 66, 3785, 33, 4, 130, 12, 16, 38, 619, 5, 25, 124, 51, 36,
135, 48, 25, 1415, 33, 6, 22, 12, 215, 28, 77, 52, 5, 14, 407, 16, 82, 2, 8, 4, 107, 117, 5952, 15,
256, 4, 2, 7, 3766, 5, 723, 36, 71, 43, 530, 476, 26, 400, 317, 46, 7, 4, 2, 1029, 13, 104, 88, 4,
381, 15, 297, 98, 32, 2071, 56, 26, 141, 6, 194, 7486, 18, 4, 226, 22, 21, 134, 476, 26, 480, 5, 144,
30, 5535, 18, 51, 36, 28, 224, 92, 25, 104, 4, 226, 65, 16, 38, 1334, 88, 12, 16, 283, 5, 16, 4472,
113, 103, 32, 15, 16, 5345, 19, 178, 32]),

      list([1, 194, 1153, 194, 8255, 78, 228, 5, 6, 1463, 4369, 5012, 134, 26, 4, 715, 8, 118, 1634,
14, 394, 20, 13, 119, 954, 189, 102, 5, 207, 110, 3103, 21, 14, 69, 188, 8, 30, 23, 7, 4, 249, 126,
93, 4, 114, 9, 2300, 1523, 5, 647, 4, 116, 9, 35, 8163, 4, 229, 9, 340, 1322, 4, 118, 9, 4, 130, 4901,
19, 4, 1002, 5, 89, 29, 952, 46, 37, 4, 455, 9, 45, 43, 38, 1543, 1905, 398, 4, 1649, 26, 6853, 5,
163, 11, 3215, 2, 4, 1153, 9, 194, 775, 7, 8255, 2, 349, 2637, 148, 605, 2, 8003, 15, 123, 125, 68, 2,
6853, 15, 349, 165, 4362, 98, 5, 4, 228, 9, 43, 2, 1157, 15, 299, 120, 5, 120, 174, 11, 220, 175, 136,
50, 9, 4373, 228, 8255, 5, 2, 656, 245, 2350, 5, 4, 9837, 131, 152, 491, 18, 2, 32, 7464, 1212, 14, 9,
6, 371, 78, 22, 625, 64, 1382, 9, 8, 168, 145, 23, 4, 1690, 15, 16, 4, 1355, 5, 28, 6, 52, 154, 462,
33, 89, 78, 285, 16, 145, 95]),

      list([1, 14, 47, 8, 30, 31, 7, 4, 249, 108, 7, 4, 5974, 54, 61, 369, 13, 71, 149, 14, 22, 112,
4, 2401, 311, 12, 16, 3711, 33, 75, 43, 1829, 296, 4, 86, 320, 35, 534, 19, 263, 4821, 1301, 4, 1873,
```

Modülün get_word_index isimli bir fonksiyonu bize sözcüklere karşı gelen indeks numaralarını bir sözlük olarak vermektedir:

```
word_dict = imdb.get_word_index()
```

get_word_index bize tüm yorumlarda kullanılan sözcük ve indeksleri vermektedir. Bu sözlük nesnesinde anahtarlar sözcüğün kendisi, değerler ise onların numaralarını belirtmektedir. Yani biz sözcüğü verdigimizde onun numarasını hızlı bir biçimde elde ederiz. Bunun tersini yine bir sözlük içlemi ile elde edebiliriz:

```
rev_word_dict = {index: word for word, index in word_dict.items()}
```

Keras imdb ve reuters gibi sözcük içeren veri kümelerindeki sözcük numaralarını hep 3 fazla almıştır. Yani örneğin bir yorumdaki 5 numaralı sözcük aslında $5 - 3 = 2$ numaralı sözcüktür. Yorumlardaki sözcükleri oluşturan sayıların ilk üçü (0, 1, 2) ayrılmıştır (rezerve edilmişdir). Bu durumda biz ilk yorumu şu biçimde sözcüklere dökebiliriz:

```
def print_review(review):
    text = ' '.join([rev_word_dict[index - 3] for index in review if index > 2])
    print(text)
```

Örneğin:

```
print_review(training_dataset_x[0])
```

Şöyledir:

this film was just brilliant casting location scenery story direction everyone's really suited the part they played and you could just imagine being there robert is an amazing actor and now the same being director father came from the same scottish island as myself so i loved the fact there was a real connection with this film the witty remarks throughout the film were great it was just brilliant so much that i bought the film as soon as it was released for and would recommend it to everyone to watch and the fly fishing was amazing really cried at the end it was so sad and you know what they say if you cry at a film it must have been good and this definitely was also to the two little boy's that played the of norman and paul they were just brilliant children are often left out of the list i think because the stars that play them all grown up are such a big profile for the whole film but these children are amazing and should be praised for what they have done don't you think the whole story was so lovely because it was true and was someone's life after all that was shared with us all

Yine önceki örnekteki gibi girdi katmanındaki nöronları sabit uzunluklu bool türden bir vektörle söyle ifade edebiliriz:

```
def vectorize(iterable, colsizes):
    result = np.zeros((len(iterable), colsizes), dtype=np.int8)
    for index, vals in enumerate(iterable):
        result[index, vals] = 1

    return result

training_dataset_x = vectorize(training_dataset_x, VOCAB_SIZE)
test_dataset_x = vectorize(test_dataset_x, VOCAB_SIZE)
```

Modelimizi de aynı biçimde oluşturabiliriz:

```
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense

model = Sequential(name='IMDB')
model.add(Dense(100, input_dim=training_dataset_x.shape[1], activation='relu', name='Hidden-1'))
model.add(Dense(100, activation='relu', name='Hidden-2'))
model.add(Dense(1, activation='sigmoid', name='Output'))
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['binary_accuracy'])
hist = model.fit(training_dataset_x, training_dataset_y, batch_size=64, epochs=5,
validation_split=0.2)
```

Loss ve metrik grafiklerini de çizelim:

```
import matplotlib.pyplot as plt

figure = plt.gcf()
figure.set_size_inches((15, 5))
plt.title('Loss - Epoch Graphics')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.plot(range(1, len(hist.history['loss']) + 1), hist.history['loss'])
plt.plot(range(1, len(hist.history['val_loss']) + 1), hist.history['val_loss'])
plt.legend(['Loss', 'Validation Loss'])
plt.show()

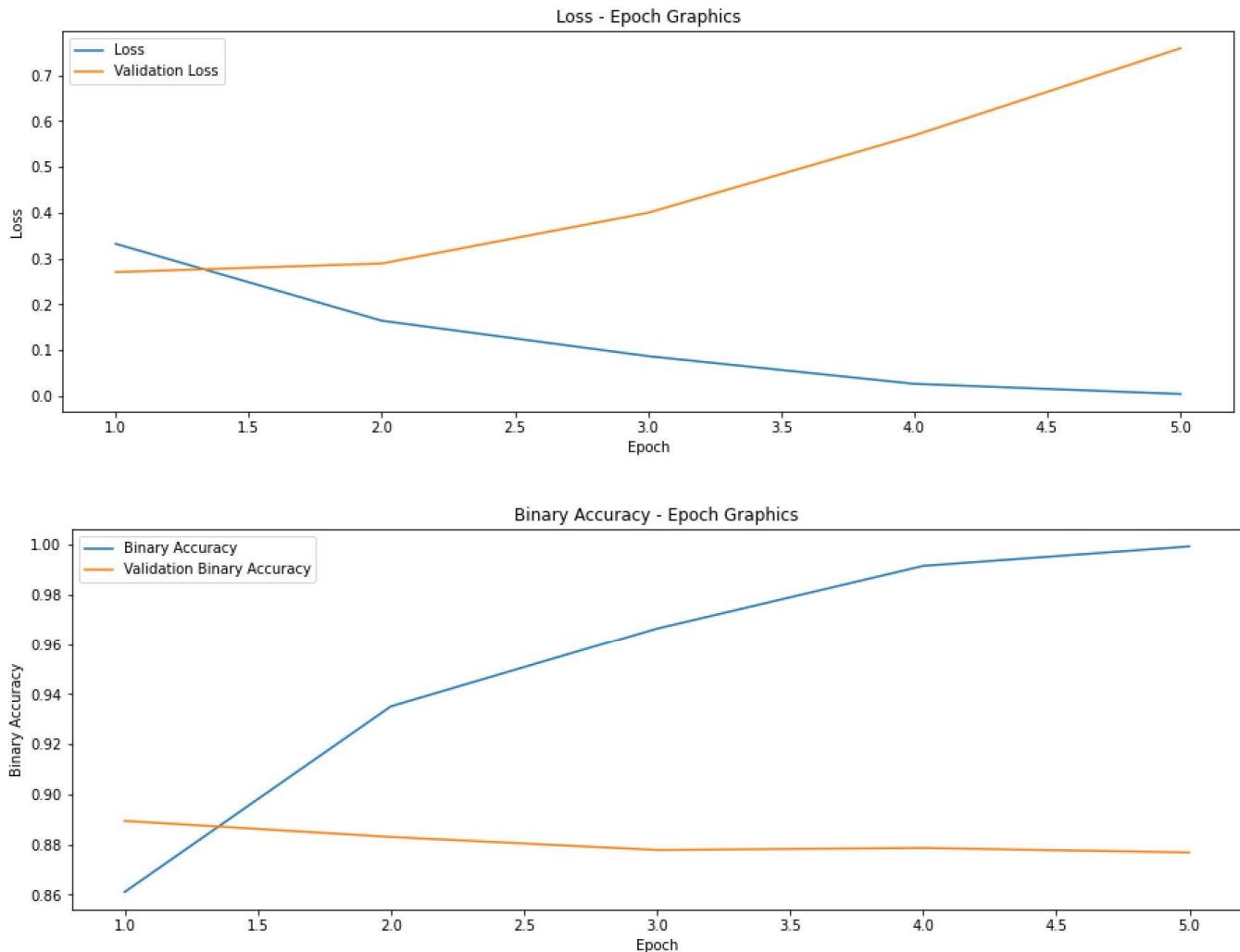
figure = plt.gcf()
figure.set_size_inches((15, 5))
plt.title('Binary Accuracy - Epoch Graphics')
plt.xlabel('Epoch')
```

```

plt.ylabel('Binary Accuracy')
plt.plot(range(1, len(hist.history['binary_accuracy']) + 1), hist.history['binary_accuracy'])
plt.plot(range(1, len(hist.history['val_binary_accuracy']) + 1),
hist.history['val_binary_accuracy'])
plt.legend(['Binary Accuracy', 'Validation Binary Accuracy'])
plt.show()

```

Aşağıdaki grafikler elde edilmiştir:



Burada yine 1 epoch'tan sonra overfit sorunu oluşmaya başlamıştır. Bu durumda eğitim 1 epoch'tan sonra sonlandırılabilir. Şimdi de modelimizi test edelim:

```

eval_result = model.evaluate(test_dataset_x, test_dataset_y)
for i in range(len(eval_result)):
    print(f'{model.metrics_names[i]}: {eval_result[i]}')

```

Şöyle bir sonuç elde edilmiştir:

```

loss: 0.7935787439346313
binary_accuracy: 0.864359974861145

```

Bu sonuç modelin %86.4'lük bir başarısı olduğunu göstermektedir. Yine tüm kodu bir bütün olarak vermek istiyoruz. Eğitimde tek bir epoch kullanacağımız için metrik grafikleri koddan kaldırıyoruz:

```

import numpy as np
from tensorflow.keras.datasets import imdb

VOCAB_SIZE = 10000

```

```

(training_dataset_x, training_dataset_y), (test_dataset_x, test_dataset_y) =
imdb.load_data(num_words=VOCAB_SIZE)

word_dict = imdb.get_word_index()
rev_word_dict = {index: word for word, index in word_dict.items()}

def vectorize(iterable, colsizes):
    result = np.zeros((len(iterable), colsizes), dtype=np.int8)
    for index, vals in enumerate(iterable):
        result[index, vals] = 1

    return result

training_dataset_x = vectorize(training_dataset_x, VOCAB_SIZE)
test_dataset_x = vectorize(test_dataset_x, VOCAB_SIZE)

from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense

model = Sequential(name='IMDB')
model.add(Dense(100, input_dim=training_dataset_x.shape[1], activation='relu', name='Hidden-1'))
model.add(Dense(100, activation='relu', name='Hidden-2'))
model.add(Dense(1, activation='sigmoid', name='Output'))
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['binary_accuracy'])
hist = model.fit(training_dataset_x, training_dataset_y, batch_size=64, epochs=1,
validation_split=0.2)

eval_result = model.evaluate(test_dataset_x, test_dataset_y)
for i in range(len(eval_result)):
    print(f'{model.metrics_names[i]}: {eval_result[i]}')

```

Şimdi de kurulan model üzerinde kestirim yapalım:

```

predict_texts = ["A nice film. But some parts are bad", "Neither a good nor a bad movie. Some
parts are exaggerated.", "it's a nice film. I recommend you to watch the movie"]

import re

def prepare_review(predict_text):
    predict_words = re.findall("[a-zA-Z0-9'-]+", predict_text.lower())
    result = np.zeros(len(predict_words), dtype=np.int8)
    for i in range(len(predict_words)):
        result[i] = word_dict[predict_words[i]] + 3

    return vectorize([result], VOCAB_SIZE)

for text in predict_texts:
    predict_data = prepare_review(text)
    predict_result = model.predict(predict_data)

    if predict_result[0, 0] > 0.5:
        print(f'Olumlu Yorum: {predict_result[0, 0]}')
    else:
        print(f'Olumsuz Yorum: {predict_result[0, 0]}'')

```

Şöyledir bir sonuç elde edilmişdir:

```

Olumsuz Yorum: 0.2089349627494812
Olumsuz Yorum: 0.14014840126037598
Olumlu Yorum: 0.7686631679534912

```

Tek Etiketli Çok Sınıflı Lojistik Regresyon Modeli İçin "Reuters" Örneği

Reuters bir yazı verildiğinde yazının konusunu 46 değişik konu arasından belirleyebilme çalışması için kullanılan popüler bir veri kümeleridir. Keras bu reuters veri kümelerini tensorflow.keras.datasets modülü içerisinde barındırmaktadır. Reuters verileri 11228 adet haber yazısını içermektedir. Bu haber yazıları 46 değişik kategoriye ayrılmıştır. Ancak maalesef Keras'ın resmi dokümanlarında bu 46 Reuters kategorisinin neler olduğu belirtilmemiştir. Bu 46 kategori başka kaynaklarda şöyle listelenmiştir:

```
category_list = ['cocoa', 'grain', 'veg-oil', 'earn', 'acq', 'wheat', 'copper', 'housing', 'money-supply', 'coffee', 'sugar', 'trade', 'reserves', 'ship', 'cotton', 'carcass', 'crude', 'nat-gas', 'cpi', 'money-fx', 'interest', 'gnp', 'meal-feed', 'alum', 'oilseed', 'gold', 'tin', 'strategic-metal', 'livestock', 'retail', 'ipi', 'iron-steel', 'rubber', 'heat', 'jobs', 'lei', 'bop', 'zinc', 'orange', 'pet-chem', 'dlr', 'gas', 'silver', 'wpi', 'hog', 'lead']
```

Aslında Keras'ın reuters verileri tamamen IMDB verileri gibi organize edilmiştir. Yani yazılar yine sözcük indeksleri ile ifade edilmiş durumdadır. Bu nedenle Reuters verilerini kullanma hazır hale getirirken IMDB örneğindekine benzer işlemleri yapacağız:

```
import numpy as np
from tensorflow.keras.datasets import reuters

VOCAB_SIZE = 10000

(training_dataset_x, training_dataset_y), (test_dataset_x, test_dataset_y) = reuters.load_data(num_words=VOCAB_SIZE)

category_list = ['cocoa', 'grain', 'veg-oil', 'earn', 'acq', 'wheat', 'copper', 'housing', 'money-supply', 'coffee', 'sugar', 'trade', 'reserves', 'ship', 'cotton', 'carcass', 'crude', 'nat-gas', 'cpi', 'money-fx', 'interest', 'gnp', 'meal-feed', 'alum', 'oilseed', 'gold', 'tin', 'strategic-metal', 'livestock', 'retail', 'ipi', 'iron-steel', 'rubber', 'heat', 'jobs', 'lei', 'bop', 'zinc', 'orange', 'pet-chem', 'dlr', 'gas', 'silver', 'wpi', 'hog', 'lead']

word_dict = reuters.get_word_index()
rev_word_dict = {value: key for key, value in word_dict.items()}
```

Yorum yazısını ekrana yazdırın fonksiyonu da şöyle yazabiliriz:

```
def print_review(review):
    for index in review:
        if index > 2:
            print(rev_word_dict[index - 3], end=' ')
```



```
print_review(training_dataset_x[0])
```

IMDB örneğinde yaptığımız gibi yine veri kümelerini bool vektöre dönüştürebiliriz:

```
def vectorize(iterable, colsizes):
    result = np.zeros(len(iterable), colsizes), dtype=np.int8)
    for index, vals in enumerate(iterable):
        result[index, vals] = 1

    return result
```



```
training_dataset_x = vectorize(training_dataset_x, VOCAB_SIZE)
test_dataset_x = vectorize(test_dataset_x, VOCAB_SIZE)
```

Burada çıktıının da aslında 46 değerli kategorik bir özellik olduğuna dikkat ediniz. Biz şimdije kadar hep girdi kümesi üzerinde one hot encoding uygulamıştık. Ancak çıktı kümesi de iki sınıfından fazlaysa çıktı kümesi üzerinde de one hot encoding uygulamamız gereklidir. Yukarıda yazdığımız vectorize fonksiyonu aslında one hot encoding işlemini de yapabilmektedir.

```
training_dataset_y = vectorize(training_dataset_y, 46)
test_dataset_y = vectorize(test_dataset_y, 46)
```

Artık ağı modelimizi oluşturabiliriz. Ağıımızda yine iki saklı katman bulunabilir. Ancak çıktı sınıflarının sayısı fazlalaştıkça katmanlardaki nöron sayılarının artırılması doğru olacaktır. Benzer biçimde kategori sayısı arttıkça ağı derinleştirmek de sonucun kalitesini artırabilmektedir. Ağıımızdaki saklı katmanların aktivasyon fonksiyonlarını yine "relu" biçiminde alacağız. İkiden fazla sınıf içeren lojistik regresyon modellerinde çıktı katmanındaki aktivasyon fonksiyonunun "softmax" alınması gerektiğini belirtmiştik. (Anımsanacağı gibi çıktı katmanın aktivasyon fonksiyonun "softmax" olması tüm çıktı nöronlarının toplam değerlerinin 1 olmasını sağlamaktadır.) Böylece biz bu değerlerin en büyüğünü temel alarak sonucu kestirebileceğiz. Model için loss fonksiyonu "categorical_crossentropy" olarak, optimizasyon algoritmasını da yine "rmsprop" olarak alacağız ve metrik değerler için de yine "categorical_accuracy" kullanacağız.

Şimdi modelimizi oluşturalım:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential(name='Reuters')
model.add(Dense(100, input_dim=VOCAB_SIZE, activation='relu', name='Hidden-1'))
model.add(Dense(100, activation='relu', name='Hidden-2'))
model.add(Dense(46, activation='softmax', name='Output'))

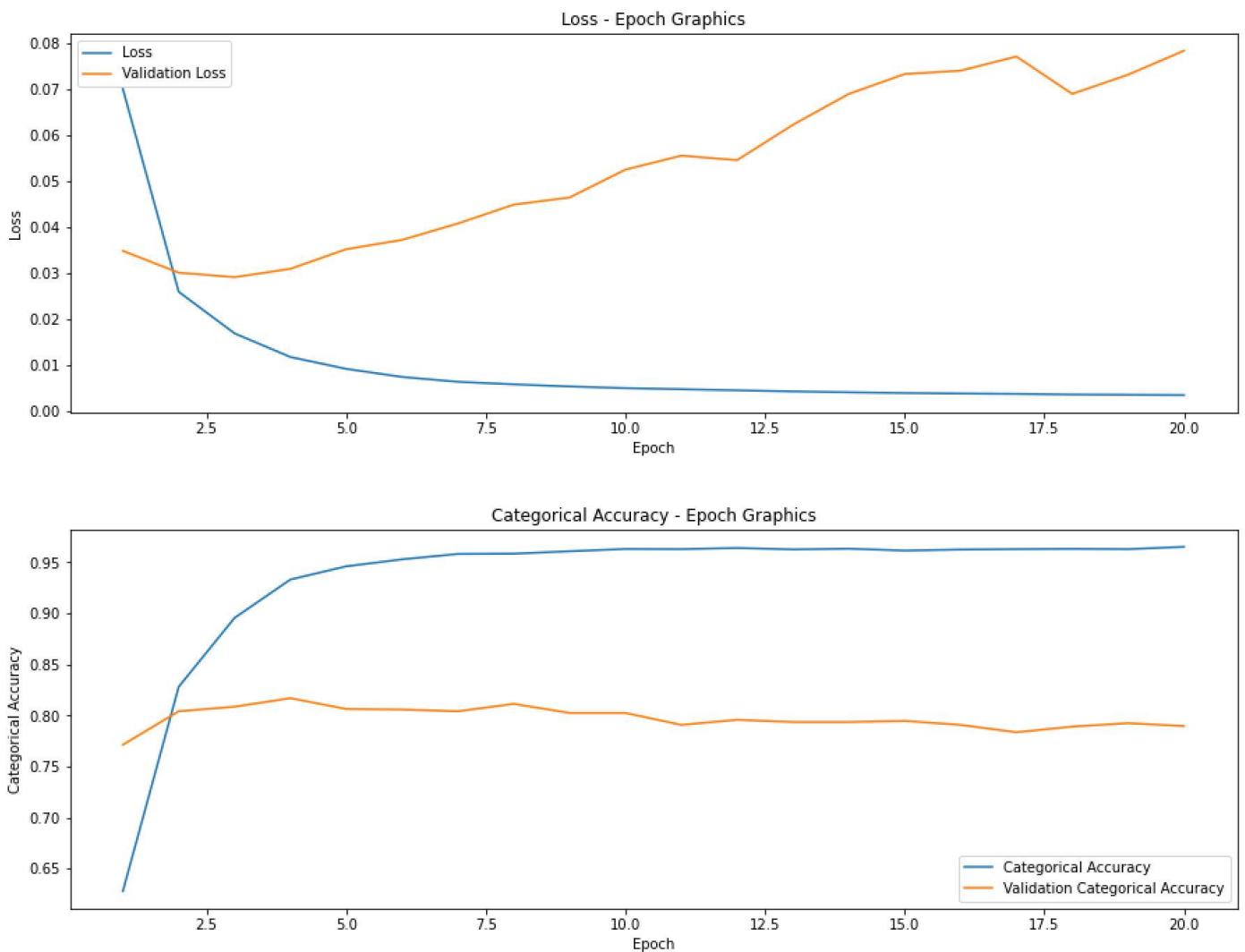
model.compile('rmsprop', loss='categorical_crossentropy', metrics=[ 'categorical_accuracy'])
hist = model.fit(training_dataset_x, training_dataset_y, epochs=20, batch_size=32,
validation_split=0.2)
```

Grafiklerimizi çizelim:

```
import matplotlib.pyplot as plt

figure = plt.gcf()
figure.set_size_inches((15, 5))
plt.title('Loss - Epoch Graphics')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.plot(range(1, len(hist.history['loss']) + 1), hist.history['loss'])
plt.plot(range(1, len(hist.history['val_loss']) + 1), hist.history['val_loss'])
plt.legend(['Loss', 'Validation Loss'])
plt.show()

figure = plt.gcf()
figure.set_size_inches((15, 5))
plt.title('Categorical Accuracy - Epoch Graphics')
plt.xlabel('Epoch')
plt.ylabel('Categorical Accuracy')
plt.plot(range(1, len(hist.history['categorical_accuracy']) + 1),
hist.history['categorical_accuracy'])
plt.plot(range(1, len(hist.history['val_categorical_accuracy']) + 1),
hist.history['val_categorical_accuracy'])
plt.legend(['Categorical Accuracy', 'Validation Categorical Accuracy'])
plt.show()
```



Burada 2'inci epoch'tan sonra overfit durumu gözlemlenmektedir. Bu epoch kaynaklı overfit nedeniyle epoch sayısı 2 ile sınırlı tutulabilir. Modelimizi test edelim:

```
eval_result = model.evaluate(test_dataset_x, test_dataset_y)
for i in range(len(eval_result)):
    print(f'{model.metrics_names[i]}: {eval_result[i]}'')
```

Şöyledir bir sonuç elde edilmiş:

```
loss: 0.08465898036956787
categorical_accuracy: 0.7822796106338501
```

Şimdi modeli 2 epoch ile eğitip de kestirim işlemini yapalım:

```
text = '''said as a result of its december acquisition of space co it expects earnings per
share in 1987 of 1 15 to 1 30 dlrs per share up from 70 cts in 1986 the company said pretax net
should rise to nine to 10 mln dlrs from six mln dlrs in 1986 and rental operation revenues to
19 to 22 mln dlrs from 12 5 mln dlrs it said cash flow per share this year should be 2 50 to
three dlrs reuter 3'''

import re

def prepare_review(predict_text):
    predict_words = re.findall("[a-zA-Z'-]+", predict_text.lower())
    result = np.zeros(len(predict_words), dtype=np.int8)
    for i in range(len(predict_words)):
        result[i] = word_dict[predict_words[i]] + 3
```

```

    return vectorize([result], VOCAB_SIZE)

predict_data = prepare_review(text)
predict_result = model.predict(predict_data)

category = np.argmax(predict_result)
print('category = {}, category name = {}'.format(category, category_list[category]))

```

Kestirim işleminin sonucunda şu çıktı elde edilmiştir:

```
category = 3, category name = earn
```

Burada biz predict işlemi sonucunda biz 46 elemanlı bir ndarray nesnesi elde etmiş olduk. Bu nesnedeki tüm değerlerin toplamı softmax fonksiyonu nedeniyle 1 olacaktır. Biz de hangi kategorinin seçildiğini en yüksek değere bakarak belirleyebiliriz. numpy'in argmax fonksiyonunun bir ndarray içerisindeki değerlerin en büyüğünün indeksini verdiğini anımsayınız.

Aşağıda örneğin tüm kodlarını yeniden veriyoruz:

```

import numpy as np
from tensorflow.keras.datasets import reuters

VOCAB_SIZE = 10000

(training_dataset_x, training_dataset_y), (test_dataset_x, test_dataset_y) =
reuters.load_data(num_words=VOCAB_SIZE)

category_list = ['cocoa', 'grain', 'veg-oil', 'earn', 'acq', 'wheat', 'copper', 'housing',
'money-supply', 'coffee', 'sugar', 'trade', 'reserves', 'ship', 'cotton', 'carcass', 'crude',
'nat-gas', 'cpi', 'money-fx', 'interest', 'gnp', 'meal-feed', 'alum', 'oilseed', 'gold', 'tin',
'strategic-metal', 'livestock', 'retail', 'ipi', 'iron-teel', 'rubber', 'heat', 'jobs', 'lei',
'bop', 'zinc', 'orange', 'pet-chem', 'dlr', 'gas', 'silver', 'wpi', 'hog', 'lead']

word_dict = reuters.get_word_index()
rev_word_dict = {value: key for key, value in word_dict.items()}

def vectorize(iterable, colsizes):
    result = np.zeros((len(iterable), colsizes), dtype=np.int8)
    for index, vals in enumerate(iterable):
        result[index, vals] = 1

    return result

training_dataset_x = vectorize(training_dataset_x, VOCAB_SIZE)
test_dataset_x = vectorize(test_dataset_x, VOCAB_SIZE)

training_dataset_y = vectorize(training_dataset_y, 46)
test_dataset_y = vectorize(test_dataset_y, 46)

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential(name='Reuters')
model.add(Dense(100, input_dim=VOCAB_SIZE, activation='relu', name='Hidden-1'))
model.add(Dense(100, activation='relu', name='Hidden-2'))
model.add(Dense(46, activation='softmax', name='Output'))

model.compile('rmsprop', loss='categorical_crossentropy', metrics=['categorical_accuracy'])
hist = model.fit(training_dataset_x, training_dataset_y, epochs=2, batch_size=32,
validation_split=0.2)

```

```

import matplotlib.pyplot as plt

figure = plt.gcf()
figure.set_size_inches((15, 5))
plt.title('Loss - Epoch Graphics')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.plot(range(1, len(hist.history['loss']) + 1), hist.history['loss'])
plt.plot(range(1, len(hist.history['val_loss']) + 1), hist.history['val_loss'])
plt.legend(['Loss', 'Validation Loss'])
plt.show()

figure = plt.gcf()
figure.set_size_inches((15, 5))
plt.title('Categorical Accuracy - Epoch Graphics')
plt.xlabel('Epoch')
plt.ylabel('Categorical Accuracy')
plt.plot(range(1, len(hist.history['categorical_accuracy']) + 1),
hist.history['categorical_accuracy'])
plt.plot(range(1, len(hist.history['val_categorical_accuracy']) + 1),
hist.history['val_categorical_accuracy'])
plt.legend(['Categorical Accuracy', 'Validation Categorical Accuracy'])
plt.show()

eval_result = model.evaluate(test_dataset_x, test_dataset_y)
for i in range(len(eval_result)):
    print(f'{model.metrics_names[i]}: {eval_result[i]}')


text = '''said as a result of its december acquisition of space co it expects earnings per
share in 1987 of 1 15 to 1 30 dlr per share up from 70 cts in 1986 the company said pretax net
should rise to nine to 10 mln dlr from six mln dlr in 1986 and rental operation revenues to
19 to 22 mln dlr from 12 5 mln dlr it said cash flow per share this year should be 2 50 to
three dlr reuter 3'''


import re

predict_words = re.findall("[a-zA-Z'-]+", text)

def prepare_review(predict_text):
    predict_words = re.findall('\w+', predict_text.lower())
    result = np.zeros(len(predict_words), dtype=np.int8)
    for i in range(len(predict_words)):
        result[i] = word_dict[predict_words[i]] + 3

    return vectorize([result], VOCAB_SIZE)

predict_data = prepare_review(text)
predict_result = model.predict(predict_data)

category = np.argmax(predict_result)
print('category = {}, category name = {}'.format(category, category_list[category]))

```

Keras Modelinin Parçalı Verilerle Eğitilmesi ve Test Edilmesi

Yapay sinir ağı modellerinde yüksek miktarda verilerle eğitimlerin yapılması gerekmektedir. Bu tür durumlarda verilerin hepsinin önce bellekte oluşturulması sonra eğitim işleminde kullanılması kapasite sorunlarına yol açabilmektedir. Örneğin biz yukarıdaki örneklerin hepsinde önce training_dataset_ ve training_dataset_y verilerini hazırlayıp fit metoduna verdik. Peki de eğitimde kullanılacak bu x ve y verileri belleğe sığmayacak ölçüde büyükse ne olacaktır? Şüphesiz çözümlerden biri bilgisayarınızı değiştirmek ya da bulut sistemlerinden faydalanan mak olabilir. Ancak bu tür yöntemler genellikle ölçeklenebilir (scalable) bir çözüm oluşturamazlar.

Yukarıda vermiş olduğumuz ilk IMDB örneğine bakalım. O örnekte eğitime sokulacak training_dataset_x dizisi 40000x156840 elemanlı, training_dataset_y dizisi ise 40000x1 elemanlı idi. Bu dizilerin her bir elemanı 1 byte yer kapladığına göre kabaca bu dizilerin bellekte kapladığı toplam alan 6 GB civarında olacaktır. Ayrıca IMDB ve Reuters örneklerindeki training_dataset_x matrislerinin aslında çok büyük oranda 0'lardan oluştuğuna da dikkat ediniz. Büyük bölümü 0'lardan oluşan matrlisler veri yapıları dünyasında "seyrek matrisler (sparse matrixes)" denilmektedir.

İşte Keras'ta çok büyük miktarda verilerle eğitimler yapılırken eğitim verileri tek parçada değil de batch işlemleri sırasında parça parça da oluşturulabilmektedir. Eğitim verilerinin parçalar halinde oluşturulması "üretici fonksiyonlar (generators)" yoluya ya da "Sequence" nesneleri yoluyla yapılmaktadır. Eskiden bu biçimde eğitim için model sınıflarının fit_generator isimli metodları kullanılıyordu. Ancak Keras'in yeni versiyonlarında fit_generator metodu "deprecated" hale getirilmiştir ve parçalı eğitim işlemlerinin de yine fit metoduyla yapılması sağlanmıştır.

Eğer fit metodunun birinci parametresinde training_dataset_x verileri yerine bir üretici fonksiyon nesni girilirse eğitim sırasında her batch işlemi için üretici fonksiyondan next fonksiyonu ile bir demet biçiminde x ve y değerleri elde edilip batch işlemi bu değerlerle yürütülmektedir. Bu durumda bir epoch işleminin kaç batch işleminden olusacağı fit metodunun steps_per_epoch parametresiyle ayarlanmaktadır. Aşağıda rastgele veriler kullanılarak bu mekanizmayla eğitimin yapıldığı bir örnek görüyorsunuz:

```
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

def data_generator():
    while True:
        training_dataset_x = np.random.rand(32, 10)
        training_dataset_y = np.random.randint(0, 2, 32)

        yield training_dataset_x, training_dataset_y

model = Sequential()
model.add(Dense(100, input_dim=10, activation='relu', name='Hidden-1'))
model.add(Dense(100, activation='relu', name='Hidden-2'))
model.add(Dense(1, activation='softmax', name='Output'))

model.compile('rmsprop', loss='binary_crossentropy', metrics=['binary_accuracy'])
hist = model.fit(data_generator(), epochs=1, steps_per_epoch=10)
```

Bu örnekte fit metoduna üretici fonksiyon parametre olarak geçirilmiştir. fit metodu da her bir batch işleminde üretici fonksiyonu next fonksiyonuyla bir kez çalıştırıp yield deyi̇miyle verilen x ve y verisini eğitimde kullanacaktır. Örneğimizde steps_per_epoch parametresi 10 olarak geçilmiştir. Bu 10 değeri 1 epoch işleminin 10 batch işleminden olusacağını belirtmektedir. Parçalı eğitim sırasında fit metodunun batch_size parametresinin fonksiyon tarafından dikkate alınmadığını da belirtmek istiyoruz. Yukarıdaki üretici fonksiyonun sonsuz döngü içerisinde olduğu dikkatinizi çekmiştir. Üretici fonksiyonun işletilmesi next işlemi ile yapıldığından bu durum bir soruna yol açmayacaktır. Normal olarak üretici fonksiyon içerisinde en azından epochs * steps_for_epoch kadar yield işleminin yapılması gerektigine dikkat ediniz.

Eğitim sırasında her epoch'tan sonra sınama işlemeye sokulacak sınama verileri de fit metodunun validation_data parametresiyle her epoch için parçalı biçimde oluşturulabilmektedir. Eğer validation_data parametresi bir üretici fonksiyon biçiminde girilirse her epoch işleminin sonunda next fonksiyonu yoluyla bu üretici fonksiyonda yield deyi̇mi ile elde edilen değerler sınama verisi olarak kullanılacaktır. fit metodunun validation_steps parametresi bir epoch için toplamda kaç kere next işlemi ile veri toplanacağını belirtir. Bu parametrenin default değerinin None olduğunu dikkat ediniz. Bu durumda veri toplama işlemi üretici fonksiyon bitene kadar yapılır ki bu da yalnızca 1 epoch için anlamlı olabilir. Örneğin:

```
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

```

def data_generator():
    for i in range(10):
        training_dataset_x = np.random.rand(32, 10)
        training_dataset_y = np.random.randint(0, 2, 32)

        yield training_dataset_x, training_dataset_y

def validation_generator():
    for i in range(10):
        validation_dataset_x = np.random.rand(32, 10)
        validation_dataset_y = np.random.randint(0, 2, 32)

        yield validation_dataset_x, validation_dataset_y

model = Sequential()
model.add(Dense(100, input_dim=10, activation='relu', name='Hidden-1'))
model.add(Dense(100, activation='relu', name='Hidden-2'))
model.add(Dense(1, activation='softmax', name='Output'))

model.compile('rmsprop', loss='binary_crossentropy', metrics=['binary_accuracy'])
hist = model.fit(data_generator(), epochs=1, steps_per_epoch=10,
validation_data=validation_generator(), validation_steps=10)

```

Burada validation_generator isimli üretici fonksiyon içerisinde rastgele değerlerle 10 ayrı yield işlemi yapılmıştır. Bu yield işlemlerinin sonunda elde edilen toplam veri sınaması verisi olarak kullanılacaktır. Modelin test edilmesi sırasında evaluate metodunda da aynı biçimde veriler parçalı olarak üretici fonksiyonlar yoluyla girilebilmektedir.

Şimdi "imdb.csv" örneği üzerinde üretici fonksiyon yoluyla verilerin elde edilmesine ilişkin bir örnek verelim. Örneğimizde "imdb.csv" dosyası tek hamlede pandas.read_csv fonksiyonu ile DataFrame nesnesi olarak okuyacağız. Sonra bu DataFrame nesnesi üzerindeki verileri eğitim, sınaması ve test sırasında üretici fonksiyon yardımıyla elde edeceğiz:

```

import numpy as np
import pandas as pd
import re

BATCH_SIZE = 64
EPOCHS = 5

class WordConverter:
    def __init__(self):
        self.word_set = set()

    def __call__(self, review):
        words = re.findall("[a-zA-Z0-9']+", review.lower())
        self.word_set.update(words)

        return np.array(words)

    def get_word_dict(self):
        return {word: index for index, word in enumerate(self.word_set)}

wc = WordConverter()
df = pd.read_csv('imdb.csv', converters={0: wc})
word_dict = wc.get_word_dict()
rev_word_dict = {index: word for word, index in word_dict.items()}

for i in range(len(df)):
    df.iloc[i, 0] = np.array([word_dict[word] for word in df.iloc[i, 0]])

"""
text = ' '.join([rev_word_dict[index] for index in df.iloc[0, 0]])

```

```

print(text)
"""

test_zone = int(len(df) * 0.8)
validation_zone = int(test_zone * 0.8)

df_training_dataset = df.iloc[:validation_zone]
df_validation_dataset = df.iloc[validation_zone:test_zone]
df_test_dataset = df.iloc[test_zone:]

df.iloc[df.iloc[:, 1] == 'positive', 1] = 1
df.iloc[df.iloc[:, 1] == 'negative', 1] = 0

def vectorize(iterable, colszie):
    result = np.zeros((len(iterable), colszie), dtype=np.int8)
    for index, values in enumerate(iterable):
        result[index, values] = 1

    return result

def data_generator(df, epochs, batch_size):
    for _ in range(epochs):
        for i in range(0, len(df) // batch_size):
            start = i * batch_size
            stop = (i + 1) * batch_size
            dataset_x = vectorize(df.iloc[start:stop, 0], len(word_dict))
            dataset_y = df.iloc[start:stop, 1].to_numpy(dtype=np.int8)

            yield dataset_x, dataset_y

training_spe = len(df_training_dataset) // BATCH_SIZE
validation_spe = len(df_validation_dataset) // BATCH_SIZE
test_spe = len(df_test_dataset) // BATCH_SIZE

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential(name='IMDB')
model.add(Dense(100, activation='relu', input_dim=len(word_dict), name='Hidden-1'))
model.add(Dense(100, activation='relu', name='Hidden-2'))
model.add(Dense(1, activation='sigmoid', name='Output'))
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['binary_accuracy'])
hist = model.fit(data_generator(df_training_dataset, EPOCHS, BATCH_SIZE), epochs=EPOCHS,
steps_per_epoch=training_spe,
validation_data=data_generator(df_validation_dataset, EPOCHS, BATCH_SIZE),
validation_steps=validation_spe)

import matplotlib.pyplot as plt

figure = plt.gcf()
figure.set_size_inches((15, 5))
plt.title('Loss - Epoch Graphics')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.plot(range(1, len(hist.history['loss']) + 1), hist.history['loss'])
plt.plot(range(1, len(hist.history['val_loss']) + 1), hist.history['val_loss'])
plt.legend(['Loss', 'Validation Loss'])
plt.show()

figure = plt.gcf()
figure.set_size_inches((15, 5))
plt.title('Binary Accuracy - Epoch Graphics')
plt.xlabel('Epoch')

```

```

plt.ylabel('Binary Accuracy')
plt.plot(range(1, len(hist.history['binary_accuracy']) + 1), hist.history['binary_accuracy'])
plt.plot(range(1, len(hist.history['val_binary_accuracy']) + 1),
hist.history['val_binary_accuracy']))
plt.legend(['Binary Accuracy', 'Validation Binary Accuracy'])
plt.show()

eval_result = model.evaluate(data_generator(df_test_dataset, 1, BATCH_SIZE))
for i in range(len(eval_result)):
    print(f'{model.metrics_names[i]} --> {eval_result[i]}')

predict_text = 'the movie was not good. There are many flaws. Players are act badly'
predict_words = re.findall("[a-zA-Z0-9]+", predict_text.lower())
predict_numbers = [word_dict[pw] for pw in predict_words]
predict_data = vectorize([predict_numbers], len(word_dict))
predict_result = model.predict(predict_data)

if predict_result[0][0] > 0.5:
    print('OLUMLU')
else:
    print('OLUMSUZ')

```

Burada dosyadan elde edilen DataFrame nesnesi önce %80'i eğitim %20'si test olmak üzere iki parçaya ayrılmıştır. Sonra da eğitim veri kümelerinin de %20'si sınıma amacıyla ayırtılmıştır. Böylece df_training_datset, df_validation_dataset ve df_test_dataset olmak üzere üç ayrı DataFrame nesnesi elde edilmiştir. data_generator isimli üretici fonksiyon DataFrame nesnesini (df) ve oluşturulacak parçaların büyüklüğünü (batch_size) parametre olarak alıp her defasında yield deyimi ile parça büyülüğu kadar veriyi oluşturup dışarıya vermektedir. Örneğimizdeki fit metodunun kullanımına dikkat ediniz:

```

hist = model.fit(data_generator(df_training_dataset, EPOCHS, BATCH_SIZE), epochs=EPOCHS,
steps_per_epoch=training_spe,
validation_data=data_generator(df_validation_dataset, EPOCHS, BATCH_SIZE),
validation_steps=validation_spe)

```

Burada biz eğitim için ve sınıma için üretici fonksiyonlardan faydalandık. Üretici fonksiyonun yield deyimi eğitim sırasında her epoch için steps_per_epoch kadar çağrılabilecek ve yield deyiminden elde edilen değerler de batch işleminde kullanılacaktır. Yukarıda belirttiğimiz gibi sınıma verileri her epoch için epoch sonunda toplanmaktadır. Örneğimizde sınıma verilerinin her epoch için 64'erli bir biçimde oluşturulduğuna dikkat ediniz. data_generator üretici fonksiyonunun nasıl yazıldığına dikkat ediniz:

```

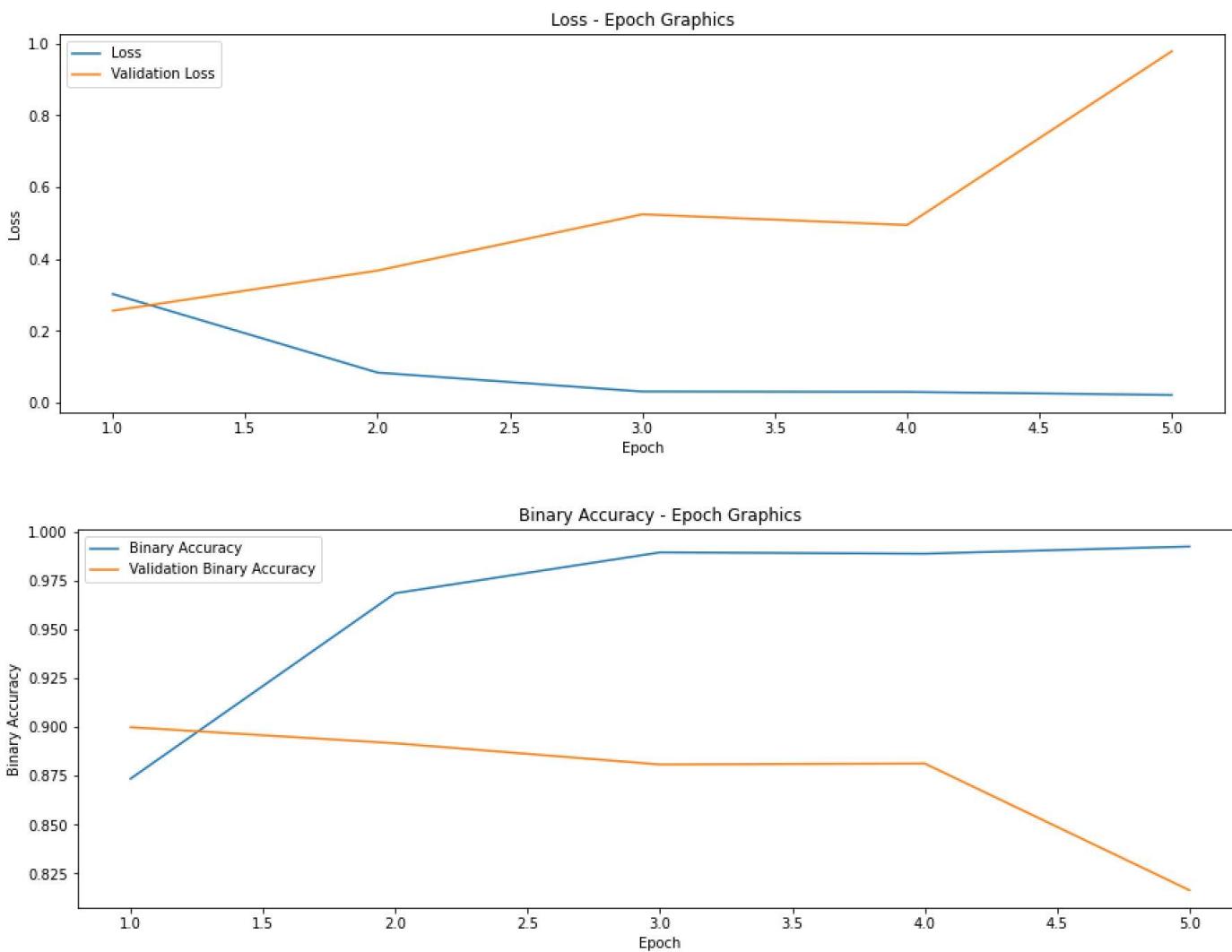
def data_generator(df, epochs, batch_size):
    for _ in range(epochs):
        for i in range(0, len(df) // batch_size):
            start = i * batch_size
            stop = (i + 1) * batch_size
            dataset_x = vectorize(df.iloc[start:stop, 0], len(word_dict))
            dataset_y = df.iloc[start:stop, 1].to_numpy(dtype=np.int8)

            yield dataset_x, dataset_y

```

Fonksiyonun birinci parametresi (df) DataFrame nesnesini ikinci parametresi (epochs) toplam epoch sayısını, üçüncü parametresi (batch_size) ise her batch işlemi için geri döndürülecek veri kümelerinin uzunluğunu (başka bir deyişle her batch işleminin ne kadar veri üzerinde gerçekleştirileceğini) belirtmektedir. Fonksiyon içerisinde iç içe döngülerle gereken sayıda yield işlemi uygulanmıştır. Burada bool vektörün bütünsel olarak oluşturulmadığına her batch işlem sırasında o anda oluşturulduğuna dikkat ediniz.

Yukarıdaki koddan elde edilen sonuçlar şunlardır:



Yine burada uygun epoch sayısının 1 olduğuna dikkat ediniz. Test işleminden şu sonuç elde edilmiştir:

```
loss: 0.9493581056594849
binary_accuracy: 0.8200120329856873
```

Keras'ta parçalı işlemle sınıflar yoluyla da yapılabilmektedir. Bunun için tensorflow.keras.utils.Sequence sınıfından bir sınıf türetilerek sınıf için `__getitem__` ve `__len__` metotları yazılır. `__len__` metodu bir epoch'un kaç batch işleminden oluşacağını belirlemek amacıyla kullanılmaktadır. Bir epoch `__len__` metodundan döndürülen sayıda batch işleminden oluşturulmaktadır. `__getitem__` metodu ise her batch işleminde fit metodu tarafından çağrılr. `__getitem__` metodunun bir index parametresi bulunmaktadır. Bu index parametresi epoch içerisindeki kaçinci batch verilerinin elde edileceğini belirtmektedir. Programcı `__getitem__` metodundan batch işleminde kullanılacak parçalı x ve y değerlerinden oluşan bir demet geri döndürmelidir. Sınıfın `on_epoch_end` metodu her epoch bittiğinde çağrılmaktadır. Programcılar tipik olarak epoch bittiğinde eğitim veri kümесinin karıştırılmasını bu metot içerisinde yaparlar.

Şimdi yukarıda üretici fonksiyonla yaptığımız "imdb.csv" örneğini şidi sınıf yoluyla gerçekleştirelim:

```
import numpy as np
import pandas as pd
import re

BATCH_SIZE = 64
EPOCHS = 5

class WordConverter:
    def __init__(self):
```

```

self.word_set = set()

def __call__(self, review):
    words = re.findall("[a-zA-Z0-9']+", review.lower())
    self.word_set.update(words)

    return np.array(words)

def get_word_dict(self):
    return {word: index for index, word in enumerate(self.word_set)}

wc = WordConverter()
df = pd.read_csv('imdb.csv', converters={0: wc})
word_dict = wc.get_word_dict()
rev_word_dict = {index: word for word, index in word_dict.items()}

for i in range(len(df)):
    df.iloc[i, 0] = np.array([word_dict[word] for word in df.iloc[i, 0]]))

"""
text = ' '.join([rev_word_dict[index] for index in df.iloc[0, 0]])
print(text)
"""

test_zone = int(len(df) * 0.8)
validation_zone = int(test_zone * 0.8)

df_training_dataset = df.iloc[:validation_zone]
df_validation_dataset = df.iloc[validation_zone:test_zone]
df_test_dataset = df.iloc[test_zone:]

df.iloc[df.iloc[:, 1] == 'positive', 1] = 1
df.iloc[df.iloc[:, 1] == 'negative', 1] = 0

def vectorize(iterable, colszie):
    result = np.zeros((len(iterable), colszie), dtype=np.int8)
    for index, values in enumerate(iterable):
        result[index, values] = 1

    return result

from tensorflow.keras.utils import Sequence

class DataGenerator(Sequence):
    def __init__(self, df, batch_size):
        self.df = df
        self.batch_size = batch_size
        self.indices = np.arange(len(self.df)) // self.batch_size, dtype=np.int32

    def __len__(self):
        return len(self.df) // self.batch_size

    def __getitem__(self, index):
        start = self.indices[index] * self.batch_size
        stop = (self.indices[index] + 1) * self.batch_size

        dataset_x = vectorize(self.df.iloc[start:stop, 0], len(word_dict))
        dataset_y = self.df.iloc[start:stop, 1].to_numpy().astype(np.int8)

        return dataset_x, dataset_y

    def on_epoch_end(self):

```

```

np.random.shuffle(self.indices)

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential(name='IMDB')
model.add(Dense(100, activation='relu', input_dim=len(word_dict), name='Hidden-1'))
model.add(Dense(100, activation='relu', name='Hidden-2'))
model.add(Dense(1, activation='sigmoid', name='Output'))
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['binary_accuracy'])
hist = model.fit(DataGenerator(df_training_dataset, BATCH_SIZE),
                  validation_data=DataGenerator(df_validation_dataset, BATCH_SIZE),
                  epochs=EPOCHS)

import matplotlib.pyplot as plt

figure = plt.gcf()
figure.set_size_inches((15, 5))
plt.title('Loss - Epoch Graphics')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.plot(range(1, len(hist.history['loss']) + 1), hist.history['loss'])
plt.plot(range(1, len(hist.history['val_loss']) + 1), hist.history['val_loss'])
plt.legend(['Loss', 'Validation Loss'])
plt.show()

figure = plt.gcf()
figure.set_size_inches((15, 5))
plt.title('Binary Accuracy - Epoch Graphics')
plt.xlabel('Epoch')
plt.ylabel('Binary Accuracy')
plt.plot(range(1, len(hist.history['binary_accuracy']) + 1), hist.history['binary_accuracy'])
plt.plot(range(1, len(hist.history['val_binary_accuracy']) + 1),
         hist.history['val_binary_accuracy'])
plt.legend(['Binary Accuracy', 'Validation Binary Accuracy'])
plt.show()

eval_result = model.evaluate(DataGenerator(df_test_dataset, BATCH_SIZE))
for i in range(len(eval_result)):
    print(f'{model.metrics_names[i]} --> {eval_result[i]}')

predict_text = 'the movie was not good. There are many flaws. Players are act badly'
predict_words = re.findall("[a-zA-Z0-9]+", predict_text.lower())
predict_numbers = [word_dict[pw] for pw in predict_words]
predict_data = vectorize([predict_numbers], len(word_dict))
predict_result = model.predict(predict_data)

if predict_result[0][0] > 0.5:
    print('OLUMLU')
else:
    print('OLUMSUZ')

```

Burada parçalı veri DataGenerator sınıfının `__getitem__` metodu tarafından oluşturulmaktadır. Bu mettotta index parametresinden hareketle verilerin batch_size kadar kısımları çekilerek geri döndürülmüştür. `__len__` metodunda bir epoch'taki batch sayısının geri döndürüldüğünü görüyorsunuz. Sınıfın `on_epoch_end` metodunda eğitim ve sınıma verileri karıştırılmıştır. Karıştırma işleminin dolaylı bir biçimde asıl veriler yoluyla değil indisler yoluyla yapıldığına dikkat ediniz. Sınıfta indices isimli bir numpy dizisi biçiminde bir örnek özniteliği bulundurulmuştur. Bu dizi `on_epoch_end` metodunda karıştırılmış ve batch verilerinin indeksleri bu diziden çekilerek kullanılmıştır.

Seyrek Matrislerle İşlemler

Elemanlarının önemli bir bölümü 0 olan küçük bölümü 0'dan farklı değerlere sahip olan matrislere seyrek matris (sparse matrix) denilmektedir. Örneğin IMDB ve Reuters örneklerinde biz yazıları her biri tüm sözcük sayılarının uzunluğu kadar sütuna sahip olan bool vektörler biçiminde kodladık. Bu biçimde elde ettiğimiz x verilerine ilişkin matrislerin çok büyük çoğunluğu 0 olan elemanlardan oluşmaktadır. Örneğin yukarıdaki "imdb.csv" dosyasından hareketle oluşturarak bool vektöre dönüştürüdüğümüz dataset_x matrisindeki 0'ların oranına bakınız:

```
In [135]: dataset_x.shape  
Out[135]: (50000, 121589)
```

```
In [136]: np.sum(dataset_x) / dataset_x.size  
Out[136]: 0.001156351808140539
```

```
In [137]: 1 - np.sum(dataset_x) / dataset_x.size  
Out[137]: 0.9988436481918594
```

Bu matriste elemanların kabaca %99.88'i 0'lardan oluşmaktadır. İşte bu biçimde elemanlarının büyük çoğunluğu 0'lardan oluşan matrislerin daha az yer kaplayacak biçimde temsil edilmeleri için bazı veri yapıları kullanılmaktadır.

Scipy kütüphanesinde seyrek matrisler üzerinde işlem yapabilen hazır birtakım sınıflar vardır. Ancak Numpy ya da Keras kütüphanelerinde seyrek matris işlemleri için özel sınıflar ya da fonksiyonlar bulundurulmamıştır. Biz burada önce bu veri yapıları üzerinde kısaca duracağız sonra da scipy kütüphanesindeki seyrek matris sınıflarını gözden geçireceğiz.

Seyrek matrislerin az yer kaplayacak biçimde temsil edilmesinde kullanılan veri yapılarından birine "DOK (Dictionary of Keys)" yöntemi denilmektedir. Bu yöntemde seyrek matris bir sözlük nesnesiyle temsil edilir. Sıfır olmayan elemanların satır ve sütun numaraları sözlüğün anahtarı o elemanların matristeki değerleri de anahtarlarla karşı gelen değerler olur. Örneğin aşağıdaki gibi bir seyrek matris olsun:

```
0  0  10  0  5  
0  2  0  0  0  
0  0  0  20  0  
4  0  0  0  0
```

Bu matris şöyle bir sözlükle saklanabilir:

```
dok_dict = {(0, 2): 10, (0, 4): 5, (1, 1): 2, (2, 3): 20, (3, 0): 4}
```

DOK matris aşağıdakine benzer bir sınıfta temsil edilebilir:

```
import numpy as np

class DokMatrix:  
    def __init__(self, shape):  
        if not isinstance(shape, tuple) or len(shape) != 2:  
            raise IndexError('shape must be 2 length tuple')  
        self.rowsize, self.colsize = shape  
        self.dict = {}  
  
    def __getitem__(self, index):  
        if not isinstance(index, tuple) or len(index) != 2:  
            raise IndexError('index must be two dimentional')  
  
        if index[0] >= self.rowsize or index[1] >= self.colsize:  
            raise IndexError('index out of range')  
  
        return self.dict.get(index, 0)  
  
    def __setitem__(self, index, value):  
        if not isinstance(index, tuple) or len(index) != 2:  
            raise IndexError('index must be two dimentional')
```

```

        raise IndexError('index must be two dimensional')

    if index[0] >= self.rowsize or index[1] >= self.colsize:
        raise IndexError('index out of range')

    self.dict[index] = value

def todense(self, dtype=np.float64):
    result = np.zeros((self.rowsize, self.colsize), dtype=dtype)
    for index, value in self.dict.items():
        result[index] = value

    return result

dm = DokMatrix((10, 10))

dm[3, 4] = 10
dm[5, 6] = 20
dm[1, 2] = 20

a = dm.todense()
print(a)

```

Scipy'da DOK matris veri yapısı `scipy.sparse` modülündeki `dok_matrix` isimli bir sınıfla temsil edilmiştir. Bu sınıf mevcut bir numpy dizisinden seyrek matris yaratıldığı gibi boş bir seyrek matris de yaratılmaktadır. `dok_matrix` yoluyla seyrek matris yaratıldıktan sonra onun istediğimiz bir elemanına değer atayabiliriz ya da onun istediğimiz bir elemanın değerini alabiliriz. Matris üzerinde dilimleme gibi işlemler de yapılabiliriz. Bu işlemlerden yine `dok_matrix` nesneleri elde edilmektedir. Seyrek matrisler üzerinde temel aritmetik işlemler de yapılabilmektedir. Bir seyrek matris istenirse ilgili sınıfların `todense` ya da `toarray` metodlarıyla iki boyutlu numpy dizilerine dönüştürülebilir.

Örneğin:

```

from scipy.sparse import dok_matrix

dm = dok_matrix((10, 10))

dm[0, 3] = 10
dm[1, 6] = 20
dm[7, 9] = 30
dm[3, 4] = 40
dm[5, 1] = 50

print(dm, end='\n\n')

dm2 = dm[1:4, 3:7]
print(dm2, end='\n\n')

dm3 = dm * 2
print(dm3, end='\n\n')

a = dm.toarray()
print(a)

```

Buradan şöyle bir çıktı elde edilmiştir:

```

(0, 3)      10.0
(1, 6)      20.0
(7, 9)      30.0
(3, 4)      40.0
(5, 1)      50.0

(0, 3)      20.0
(2, 1)      40.0

```

```

(0, 3)    20.0
(1, 6)    40.0
(7, 9)    60.0
(3, 4)    80.0
(5, 1)   100.0

[[ 0.  0.  0. 10.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0. 20.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0. 40.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0. 50.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0. 30.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.]]
```

Seyrek matrisleri temsil etmek için kullanılan diğer bir veri yapısı LIL (List of List) biçiminde isimlendirilmektedir. Bu veri yapısında satırlara ilişkin iki paralel liste dizisi tutulur. Dizilerin bir tanesinde satırlardaki değerler, diğerinde ise bu değerlere ilişkin sütun numaraları bulunmaktadır. Yine seyrek matrisimiz şöyle olsun:

```

0  0  10  0  5
0  2  0  0  0
0  0  0 20  0
4  0  0  0  0
```

Bu matris LIL formatı ile şöyle saklanabilir:

```

rows = [[10, 5], [1], [20], [4]]
cols = [[2, 3], [1], [3], [0]]
```

Burada rows listesinde satırlardaki değerler, cols listesinde de onların sırasıyla sütun indeksleri bulunmaktadır. Bu veri yapısında genellikle satırlardaki sütun indeksleri sıralı bir biçimde tutulmaktadır. Bu sayede satırdaki elemanlara ikili arama (binary search) yoluyla erişilebilmektedir.

LIL matrisler Scipy kütüphanesinde scipy.sparse modülündeki lil_matrix sınıfıyla temsil edilmiştir. Sınıfın genel kullanımı dok_matrix sınıfında olduğu gibidir. Örneğin:

```

from scipy.sparse import lil_matrix

lil = lil_matrix((10, 10))

lil[0, 3] = 10
lil[1, 6] = 20
lil[7, 9] = 30
lil[3, 4] = 40
lil[5, 1] = 50

print(lil, end='\n\n')

lil2 = lil[1:4, 3:7]
print(lil2, end='\n\n')

lil3 = lil * 2
print(lil3, end='\n\n')

a = lil.toarray()
print(a)
```

CSR (Compressed Sparse Row) ve CSC (Compress Sparce Column) diğer iki önemli seyrek matris veri yapısıdır. CSR formatında seyrek matris üç dizi ile ifade edilmektedir. Dizilerden biri satır temelinde sıfır olmayan elemanların değerlerini, ikincisi bu elemanların sütun numaralarını ve üçüncüsü de her satırdaki elemanların birinci dizideki başlangıç ve bitiş indeks numaralarını tutmaktadır. Aşağıdaki gibi bir seyrek matris olsun:

```
0  0  10  0  5  7  12  0
0  2  5  0  0  0  7  6
0  7  0  20  0  9  0  0
4  0  2  0  3  0  8  0
```

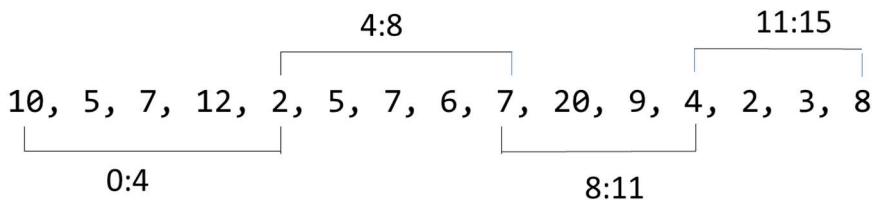
Bu matris CSR formatında aşağıdaki gibi üç dizi (liste) ile ifade edilir:

```
data = [10, 5, 7, 12, 2, 5, 7, 6, 7, 20, 9, 4, 2, 3, 8]
indices = [2, 4, 5, 6, 1, 2, 6, 7, 1, 3, 5, 0, 2, 4, 6]
indptr = [0, 4, 8, 11, 11, 15]
```

Buradaki data ve indices paralel iki dizidir. data dizisi satır temelli soldan sağa 0 olmayan elemanları, indices ise bu elemanların sütun indekslerini tutmaktadır. indptr belli satırdaki elemanların data listesindeki dilimlerini göstermektedir. Yani yukarıdaki örnekte 10 değeri 2'inci sütunda, 5 değeri 4'üncü sütunda, 7 değeri ise 5'inci sütundadır. Ancak bunların hangi satırlarda olduğu bilinmemektedir. İşte indptr aslında bu elemanların hangi satırda olduğuna ilişkin dilimlemelerden oluşmaktadır:

```
(0, 4) --> 0'inci satırdaki elemanlar
(4, 8) --> 1'inci satırdaki elemanlar
(8, 11) --> 2'inci satırdaki elemanlar
(11, 15) --> 3'üncü satırdaki elemanlar
```

indptr ile data listeleri arasındaki ilişkiyi şekilsel olarak da şöyle gösterebiliriz:



CSR matrlslere yeni bir eleman atanması DOK ve LIL matrlslere kıyasla daha fazla zaman gerektirmektedir. Böyle bir işlemde uyarı alırsanız şaşırmayınız. Ancak CSR matrlsler etkin biçimde birbirleriyle aritmetik işlemlere sokulabilmektedir. csr_matrix sınıfının data örnek özniteligi 0 olmayan matrls elemanlarını, indices örnek özniteligi bu elemanların sütun numaralarını ve indptr örnek özniteligi ise satır dilimlemelerini bize numpy dizisi olarak vermektedir. Örneğin:

```
import numpy as np
from scipy.sparse import csr_matrix

a = np.array([[0, 0, 10, 0, 5, 7, 12, 0],
              [0, 2, 5, 0, 0, 0, 7, 6],
              [0, 7, 0, 20, 0, 9, 0, 0],
              [0, 0, 0, 0, 0, 0, 0, 0],
              [4, 0, 2, 0, 3, 0, 8, 0]])

csr = csr_matrix(a, dtype=np.int32)
print(csr.toarray(), end='\n\n')
print(csr.data, end='\n\n')
print(csr.indices, end='\n\n')
print(csr.indptr, end='\n\n')
```

```

csr2 = csr * 2
print(csr2.toarray())

```

Şöyledir bir çıktı elde edilmiştir:

```

[[ 0  0 10  0  5  7 12  0]
 [ 0  2  5  0  0  0  7  6]
 [ 0  7  0 20  0  9  0  0]
 [ 0  0  0  0  0  0  0  0]
 [ 4  0  2  0  3  0  8  0]]]

[10  5  7 12  2  5  7  6  7 20  9  4  2  3  8]

[2 4 5 6 1 2 6 7 1 3 5 0 2 4 6]

[ 0  4  8 11 11 15]

[[ 0  0 20  0 10 14 24  0]
 [ 0  4 10  0  0  0 14 12]
 [ 0 14  0 40  0 18  0  0]
 [ 0  0  0  0  0  0  0  0]
 [ 8  0  4  0  6  0 16  0]]

```

csr_matrix sınıfının çeşitli işlemler yapan pek çok metodu vardır. Bunları scipy dokümanlarından inceleyebilirsiniz.

CSC matrisleri depolama biçimini olarak CSR matrisleri gibidir. Ancak eleman yerleri satır temelli değil sütun temelli olarak tutulmaktadır. CSR matris için de benzer örneği verebiliriz:

```

import numpy as np
from scipy.sparse import csc_matrix

a = np.array([[0, 0, 10, 0, 5, 7, 12, 0],
              [0, 2, 5, 0, 0, 0, 7, 6],
              [0, 7, 0, 20, 0, 9, 0, 0],
              [0, 0, 0, 0, 0, 0, 0, 0],
              [4, 0, 2, 0, 3, 0, 8, 0]])

csc = csc_matrix(a, dtype=np.int32)
print(csc.toarray(), end='\n\n')
print(csc.data, end='\n\n')
print(csc.indices, end='\n\n')
print(csc.indptr, end='\n\n')
csr2 = csr * 2
print(csr2.toarray())

```

Şöyledir bir çıktı elde edilmiştir:

```

[[ 0  0 10  0  5  7 12  0]
 [ 0  2  5  0  0  0  7  6]
 [ 0  7  0 20  0  9  0  0]
 [ 0  0  0  0  0  0  0  0]
 [ 4  0  2  0  3  0  8  0]]]

[ 4  2  7 10  5  2 20  5  3  7  9 12  7  8  6]

[4 1 2 0 1 4 2 0 4 0 2 0 1 4 1]

[ 0  1  3  6  7  9 11 14 15]

[[ 0  0 20  0 10 14 24  0]
 [ 0  4 10  0  0  0 14 12]
 [ 0 14  0 40  0 18  0  0]]

```

```
[ 0  0  0  0  0  0  0  0]
[ 8  0  4  0  6  0 16  0]]
```

Pekiyi burada ele aldığımız Scipy seyrek matris sınıflarının hangileri hangi durumlarda tercih edilmelidir? Bunları birkaç maddede özetleyebiliriz:

- DOK matriste elemanlara okuma ya da yazma amaçlı erişim hızlı bir biçimde gerçekleştirilmektedir. Ancak DOK matrisler matris işlemlerinde çok etkin değildir. DOK matrisler dilimleme de etkin değildir.
- LIL matrisler de okuma amaçlı eleman erişimlerinde ve satırsal dilimlemelerde hızlıdır. Ancak sütunsal dilimlemelerde ve matris işlemlerinde yavaşırlar. 0 olan elemanlara yazma amaçlı erişimlerde çok hızlı olmasalar da nispeten hızlıdır. Bu matrislerin matris işlemleri için CSR ve CSC formatlarına dönüştürülmesi uygundur ve bu dönüştürme hızlıdır.
- CSR matrisler satırsal dilimlemelerde CSC matrisler ise sütunsal dilimlemelerde hızlıdır. Ancak CSR sütunsal, CSC de satırsal dilimlemelerde yavaştır. Her iki matris de matris işlemlerinde hızlıdır. Bu matrislerde elemanların değerlerinin değiştirilmek (özellikle 0 olan elemanların) yavaştır.

Scipy kütüphanesinde burada ele aldığımızın dışında da başka seyrek matris sınıfları bulunmaktadır. Bu sınıfları Scipy dokümanlarından inceleyebilirsiniz.

Şimdi de bir seyrek matris uygulaması yapalım. Uygulamamızda yine "imdb.csv" verilerini kullanacağız. Önce bu dosyayı yine pandas.read_csv fonksiyonuyla bir DataFrame olarak okuyacağız. Sonra bu DataFrame nesnesini seyrek matris olarak okuyup seyrek matrisi parçalı biçimde bool vektöre dönüştüreceğiz:

```
import numpy as np
import pandas as pd
import re

BATCH_SIZE = 64
EPOCHS = 5

class WordConverter:
    def __init__(self):
        self.word_set = set()

    def __call__(self, review):
        words = re.findall("[a-zA-Z0-9]+", review.lower())
        self.word_set.update(words)

        return np.array(words)

    def get_word_dict(self):
        return {word: index for index, word in enumerate(self.word_set)}

wc = WordConverter()
df = pd.read_csv('imdb.csv', converters={0: wc})
word_dict = wc.get_word_dict()
rev_word_dict = {index: word for word, index in word_dict.items()}

for i in range(len(df)):
    df.iloc[i, 0] = np.array([word_dict[word] for word in df.iloc[i, 0]])

"""
text = ' '.join([rev_word_dict[index] for index in df.iloc[0, 0]])
print(text)
"""

test_zone = int(len(df) * 0.8)
validation_zone = int(test_zone * 0.8)
```

```

df_training_dataset = df.iloc[:validation_zone]
df_validation_dataset = df.iloc[validation_zone:test_zone]
df_test_dataset = df.iloc[test_zone:]

df.iloc[df.iloc[:, 1] == 'positive', 1] = 1
df.iloc[df.iloc[:, 1] == 'negative', 1] = 0

from scipy.sparse import lil_matrix

def vectorize(iterable, colszie):
    result = lil_matrix((len(iterable), colszie), dtype=np.int8)
    for index, values in enumerate(iterable):
        result[index, values] = 1

    return result

sparse_training_dataset_x = vectorize(df_training_dataset.iloc[:, 0], len(word_dict))
training_dataset_y = df_training_dataset.iloc[:, 1].values.astype(np.int8)

sparse_validation_dataset_x = vectorize(df_validation_dataset.iloc[:, 0], len(word_dict))
validation_dataset_y = df_validation_dataset.iloc[:, 1].values.astype(np.int8)

sparse_test_dataset_x = vectorize(df_test_dataset.iloc[:, 0], len(word_dict))
test_dataset_y = df_test_dataset.iloc[:, 1].values.astype(np.int8)

from tensorflow.keras.utils import Sequence

class DataGenerator(Sequence):
    def __init__(self, sparse_dataset_x, dataset_y, batch_size):
        self.sparse_dataset_x = sparse_dataset_x
        self.dataset_y = dataset_y
        self.batch_size = batch_size
        self.indices = np.arange(len(self))

    def __len__(self):
        return len(self.dataset_y) // self.batch_size

    def __getitem__(self, index):
        start = self.indices[index] * self.batch_size
        stop = (self.indices[index] + 1) * self.batch_size

        return self.sparse_dataset_x[start:stop, :].toarray(), self.dataset_y[start:stop]

    def on_epoch_end(self):
        np.random.shuffle(self.indices)

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential(name='IMDB')
model.add(Dense(100, activation='relu', input_dim=len(word_dict), name='Hidden-1'))
model.add(Dense(100, activation='relu', name='Hidden-2'))
model.add(Dense(1, activation='sigmoid', name='Output'))
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['binary_accuracy'])
hist = model.fit(DataGenerator(sparse_training_dataset_x, training_dataset_y, BATCH_SIZE),
                  validation_data=DataGenerator(sparse_validation_dataset_x, validation_dataset_y, BATCH_SIZE),
                  epochs=EPOCHS)

import matplotlib.pyplot as plt

figure = plt.gcf()

```

```

figure.set_size_inches((15, 5))
plt.title('Loss - Epoch Graphics')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.plot(range(1, len(hist.history['loss']) + 1), hist.history['loss'])
plt.plot(range(1, len(hist.history['val_loss']) + 1), hist.history['val_loss'])
plt.legend(['Loss', 'Validation Loss'])
plt.show()

figure = plt.gcf()
figure.set_size_inches((15, 5))
plt.title('Binary Accuracy - Epoch Graphics')
plt.xlabel('Epoch')
plt.ylabel('Binary Accuracy')
plt.plot(range(1, len(hist.history['binary_accuracy']) + 1), hist.history['binary_accuracy'])
plt.plot(range(1, len(hist.history['val_binary_accuracy']) + 1),
hist.history['val_binary_accuracy'])
plt.legend(['Binary Accuracy', 'Validation Binary Accuracy'])
plt.show()

eval_result = model.evaluate(DataGenerator(sparse_test_dataset_x, test_dataset_y, BATCH_SIZE))
for i in range(len(eval_result)):
    print(f'{model.metrics_names[i]} --> {eval_result[i]}')

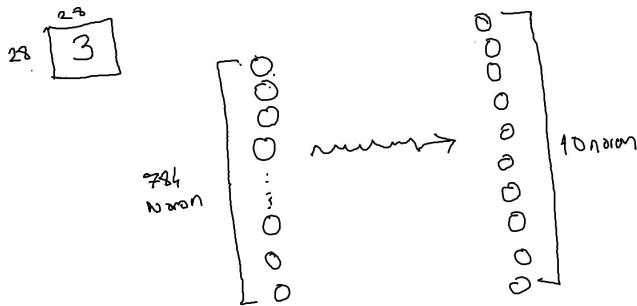
predict_text = 'the movie was not good. There are many flaws. Players are act badly'
predict_words = re.findall("[a-zA-Z0-9]+", predict_text.lower())
predict_numbers = [word_dict[pw] for pw in predict_words]
predict_data = vectorize([predict_numbers], len(word_dict))
predict_result = model.predict(predict_data)

if predict_result[0][0] > 0.5:
    print('OLUMLU')
else:
    print('OLUMSUZ')

```

Tek Etiketli Çok Sınıflı Lojistik Regresyon Modeli İçin "MNIST (Modified National Institute of Standards and Technology)" Örneği

MNIST makine öğrenmesinde deneme ve eğitim amacıyla en çok kullanılan veritabanlarından biridir. MNIST'in çok değişik veri kümeleri vardır. Bu veri kümelerinden biri Keras içerisinde de bulundurulmuştur. keras.mnist modülünde her biri 28x28'lik bir "gri tonlamalı (grayscale)" bitmap'ten oluşan pek çok resim bulunmaktadır. Bu resimlerde 0'dan 9'a kadar sayıların elle çizilmiş şekilleri vardır. Dolayısıyla modelin girdi katmanı $28 * 28 = 784$ nörondan çıktı katmanı da 10 nörondan oluşmaktadır.



Bir resim bilindiği gibi pixel'lerden oluşmaktadır. Her pixel'in de renk bileşenleri vardır. Pixel'le rin renk bileşenleri için en çok kullanılan format RGB (Red, Green, Blue) formatıdır. Bu formatta renk bileşenleri her biri 1 byte'tan oluşan Kırmızı (Red), Yeşil (Green) ve Maviniñ (Blue) tonal değerleriyle ifade edilmektedir. Pek çok görüntü birimi de aslında kendi içerisinde renkleri zaten bu renklerin tonal bileşimleriyle oluşturmaktadır. Bir byte ile ifade edilecek sayılar toplam 256 tane olduğuna göre 1 byte'lık RGB formatında her pixel $256 * 256 * 256 = 2^{24}$ renkten biri ile renklendirilebilmektedir. Aslında modern grafik kartlarında her pixel için bir byte da transparanlık (alpha channel)

bilgisi tutulmaktadır. Transparanlık arkadaki görüntünün görülebilirliği üzerinde etkili olmaktadır. Bu transparanlık 255 ise arka plan tam saydamsız, 0 ise tam saydam anlamına gelir.

Gri tonlamalı resim aslında grinin tonlarından oluşan resimdir. Gri tonlamalı resimlerde her bir pixel RGB ile değil tek bir değerle ifade edilir. Yani gri tonlamalı resim aslında $R = G = B$ pixel renklerinden oluşan resimdir. Böylece gri tonlamalı resimde her pixel üç byte'la değil bir byte'la kodlanabilmektedir. Gri tonlamalı resimleri siyah-beyaz (monochrome) resimlerle karıştırılmayınız. Siyah-beyaz (buradaki siyah ve beyaz başka renkler de olabilir) bir resimde her pixel 1 bit ile ifade edilmektedir. Ancak siyah-beyaz resimlerin çoğu insanlar tarafından anlamsız bir görüntü biçiminde algılanmaktadır. Siyah tonlamalı resimler insanlar tarafından çok daha iyi algılanmaktadır.

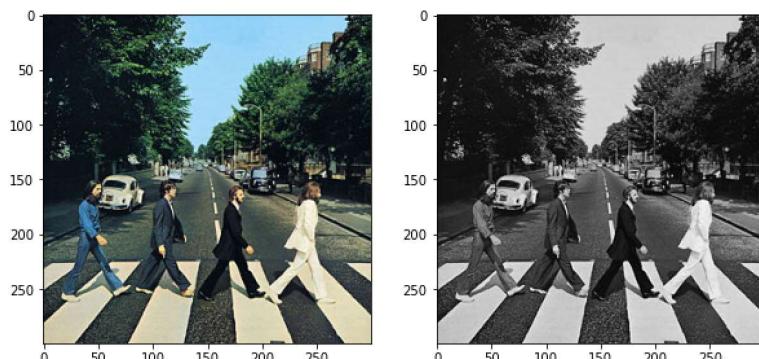
Peki normal RGB bir resim gri tonlamalı biçimde dönüştürülebilir mi? Yanıt evet. Bunun için kullanılan iki temel teknik vardır. Birinci teknikte her pixel'in RGB değerlerinin ortalaması alınır. Ancak bu yöntem bazı resimlerde çok tatmin edici sonuçlar vermemektedir. Bu yöntem için şöyle bir örnek verebiliriz:

```
import numpy as np
import matplotlib.pyplot as plt

img_data = plt.imread('AbbeyRoad.jpg')
gray_img_data = np.mean(img_data, axis=2)

figure = plt.gcf()
figure.set_size_inches((10, 10))
plt.subplot(1, 2, 1)
plt.imshow(img_data)
plt.subplot(1, 2, 2)
plt.imshow(gray_img_data, cmap='gray')
plt.show()
```

Elde edilen görüntüleri karşılaştırınız:



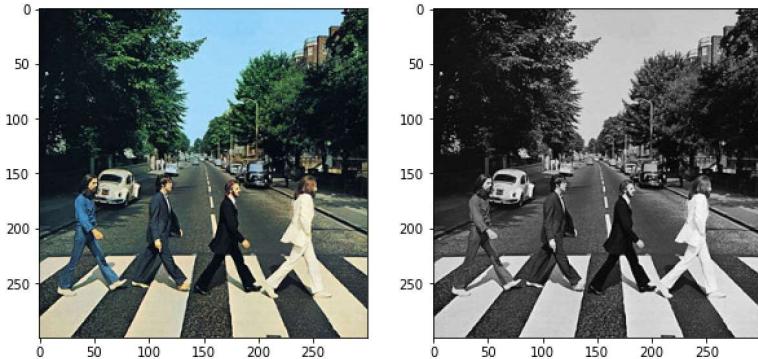
Diger yöntemde pixel'in RGB değerleri önce özel değerlerle çarpılıp sonra onların ortalamaları alınmaktadır. Uygulamada genellikle bu ağırlıklı ortalama yöntemi tercih edilmektedir. Yukarıdaki örneği bu yöntemle söyle uygulayabiliriz:

```
import numpy as np
import matplotlib.pyplot as plt

img_data = plt.imread('AbbeyRoad.jpg')
gray_img_data = np.average(img_data, weights=[0.3, 0.59, 0.11], axis=2)

figure = plt.gcf()
figure.set_size_inches((10, 10))
plt.subplot(1, 2, 1)
plt.imshow(img_data)
plt.subplot(1, 2, 2)
plt.imshow(gray_img_data, cmap='gray')
plt.show()
```

Burada R, B için ağırlık değerleri sırasıyla 0.3, 0.59, 0.11 alınmıştır. Elde edilen görüntü benzerdir:



MNIST örneğimizde işe önce veri kümesini yüklemekle başlayalım:

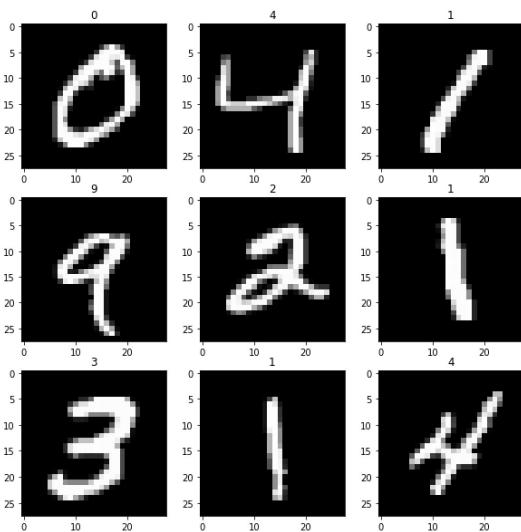
```
from tensorflow.keras.datasets import mnist  
(training_dataset_x, training_dataset_y), (test_dataset_x, test_dataset_y) = mnist.load_data()
```

Buradaki training_dataset_x her biri 28X28'lik matristen oluşan 60000'luk bir numpy dizisidir. Yani bu dizi 60000X28X28 boyutundadır. Benzer biçimde test_dataset_x de her biri 28X28'lik matristen oluşan 10000'luk bir ndarray dizisidir. Tabii bu dizilerin elemanları gri tonlamalı pixel belirttiğine göre 0 ile 255 arasında olmak zorundadır. training_dataset_y ve test_dataset_y dizileri ise tek boyutlu ndarray nesneleridir. Bu dizilerin her elemanı 0'dan 9'a kadar kategorik bir değer belirtmektedir. Bu değerin örneğin 3 olması demek oradaki 28x28 gri tonlamalı resmin üzerindeki çizimin 3 çizimi olması demektir.

Şimdi biz matplotlib.pyplot kullanarak buradaki birkaç resmi çizdirelim:

```
import matplotlib.pyplot as plt  
  
figure = plt.gcf()  
figure.set_size_inches(10, 10)  
for i in range(1, 10):  
    plt.subplot(3, 3, i)  
    axis = plt.gca()  
    axis.set_title(str(training_dataset_y[i]))  
    plt.imshow(training_dataset_x[i], cmap='gray')  
  
plt.show()
```

Çizim işleminden şöyle bir görüntü elde edilmiştir:



28x28'lik üç boyutlu resim matrisini yapay sinir ağına girdi yapmak için iki boyutlu hale getirelim:

```
training_dataset_x = training_dataset_x.reshape(-1, 28 * 28)
test_dataset_x = test_dataset_x.reshape(-1, 28 * 28)
```

Şimdi de `training_dataset_y` ve `test_dataset_y` verilerini düzenleyelim. Bizim ağımızın çıktı katmanında 10 tane nöron olacağına göre biz de ağıımızı eğitirken çıktı için 10 değer bulundurmalıyız. Anımsanacağı gibi çok sınıflı sınıflandırma problemlerinde çıktı katmanındaki aktivasyon fonksiyonları "softmax" alınıyordu. Bu "softmax" fonksiyonu çıktı nöronlarının toplam değerini 1'de tutuyordu. Ayrıca modelin çıktısı kategorik bir veri olduğuna göre eğitimden önce bu çıktı verilerini de one hot encoding biçiminde kodlamamız gereklidir. One hot encoding için önceki örneklerde genellikle scikit-learn.preprocessing modülündeki OneHotEncoder sınıfını kullanmıştık. Burada bir değişiklik yaparak işlemi sizleri alıstırmak için Keras'taki `to_categorical` fonksiyonu ile yapalım:

```
from tensorflow.keras.utils import to_categorical

training_dataset_y = to_categorical(training_dataset_y)
test_dataset_y = to_categorical(test_dataset_y)
```

Şimdi verilerimizi normalize edelim. Bunun için MinMax ölçeklendirmesi yöntemini kullanabiliriz:

```
training_dataset_x = training_dataset_x / 255
test_dataset_x = test_dataset_x / 255
```

Burada en düşük pixel değeri 0 ve en büyük pixel değeri 255 olduğu için MinMax ölçeklendirmesini pixel değerlerini doğrudan 255'e bölgerek yaptığımiza dikkat ediniz.

Şimdi modelimizi oluşturalım:

```
from tensorflow.keras.utils import to_categorical

training_dataset_y = to_categorical(training_dataset_y)
test_dataset_y = to_categorical(test_dataset_y)

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

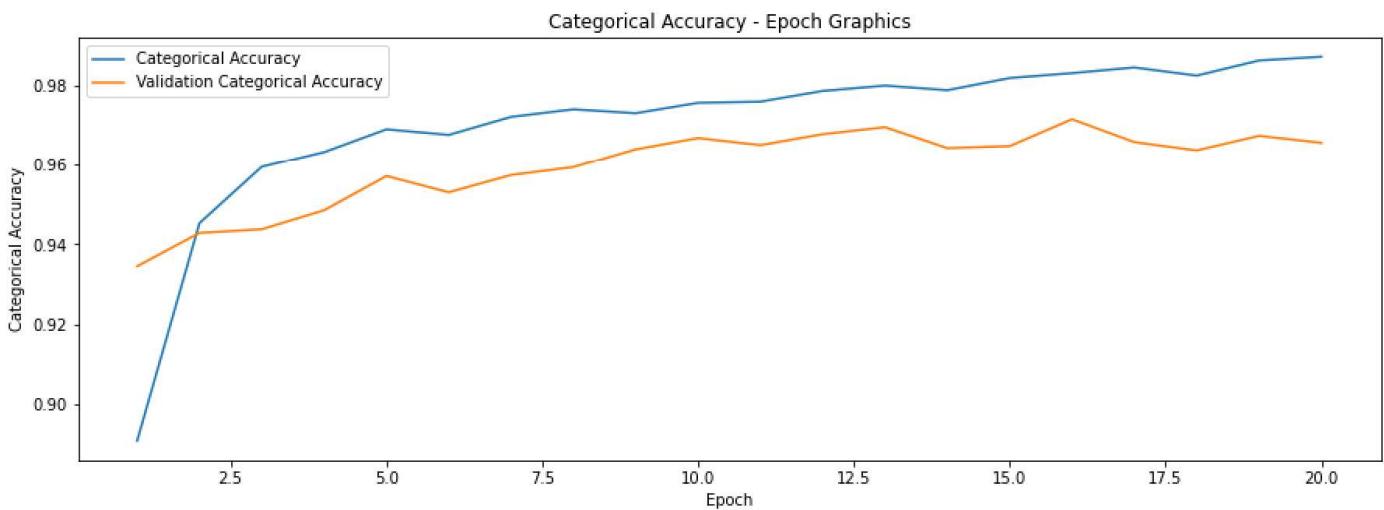
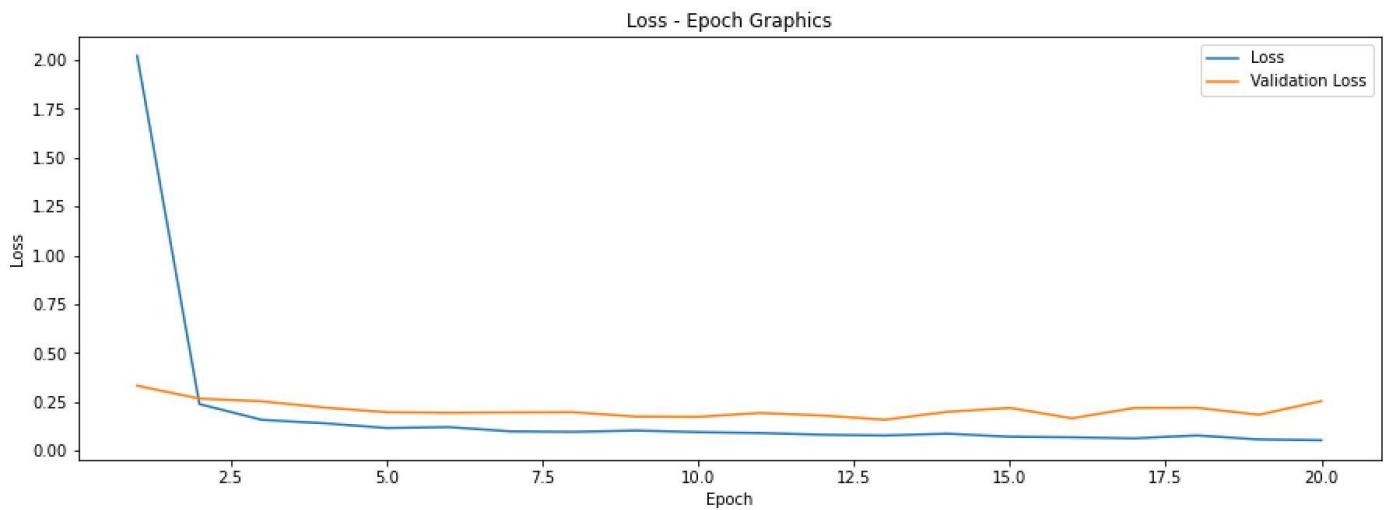
model = Sequential()
model.add(Dense(512, input_dim=28 * 28, activation='relu', name='Hidden-1'))
model.add(Dense(256, activation='relu', name='Hidden-2'))
model.add(Dense(10, activation='softmax', name='Output'))

model.compile('adam', loss='categorical_crossentropy', metrics=['categorical_accuracy'])
```

```
hist = model.fit(training_dataset_x, training_dataset_y, epochs=20, batch_size=64,  
validation_split=0.2)
```

Şimdi de grafiklerimizi çizelim:

```
import matplotlib.pyplot as plt  
  
figure = plt.gcf()  
figure.set_size_inches((15, 5))  
plt.title('Loss - Epoch Graphics')  
plt.xlabel('Epoch')  
plt.ylabel('Loss')  
plt.plot(range(1, len(hist.history['loss']) + 1), hist.history['loss'])  
plt.plot(range(1, len(hist.history['val_loss']) + 1), hist.history['val_loss'])  
plt.legend(['Loss', 'Validation Loss'])  
plt.show()  
  
figure = plt.gcf()  
figure.set_size_inches((15, 5))  
plt.title('Categorical Accuracy - Epoch Graphics')  
plt.xlabel('Epoch')  
plt.ylabel('Categorical Accuracy')  
plt.plot(range(1, len(hist.history['categorical_accuracy']) + 1),  
hist.history['categorical_accuracy'])  
plt.plot(range(1, len(hist.history['val_categorical_accuracy']) + 1),  
hist.history['val_categorical_accuracy'])  
plt.legend(['Categorical Accuracy', 'Validation Categorical Accuracy'])  
plt.show()
```



Bu grafiklere göre epoch sayısı arttıkça düşük düzeyli bir "overfit" şüphesi belirmektedir. Bu nedenle epoch sayısı 10 civarında tutulabilir.

Şimdi de modelimiz test edelim:

```
eval_result = model.evaluate(test_dataset_x, test_dataset_y)
for i in range(len(eval_result)):
    print(f'{model.metrics_names[i]} --> {eval_result[i]}')
```

Şu değerler elde edilmiştir:

```
loss --> 0.20867395401000977
categorical_accuracy --> 0.9699000120162964
```

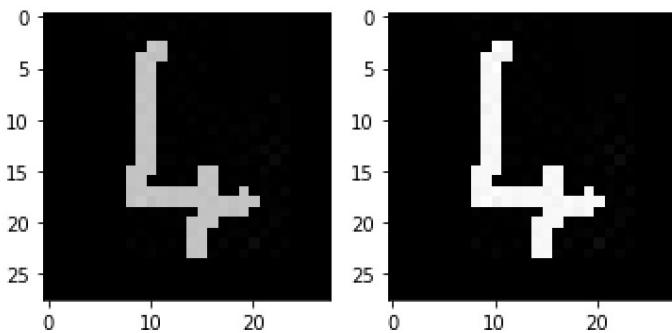
Buradaki categorical_accuracy değerine bakıldığında bu modelin %96.9 olasılıkla girilen rakamı doğru tanadığını söyleyebiliriz. Daha yüksek bir veri kümlesi üzerinde eğitim yapıldığında bu değer artacaktır.

Şimdi de predict işlemi yapmak isteyelim. Biz bu işlemi gerçek bir şekil üzerinde yapalım. Bunun için Paint gibi bir programda elle bir rakam çizip onu BMP ya da PNG formatında kaydedebiliriz. Sonra pyplot.imread kullanarak onun içerisindeki pixel verilerini okuyup gri tonlamalı hale dönüştürebiliriz.

```
import numpy as np

img_data = plt.imread('test.jpg')
gray_img_data = np.average(img_data, weights=[0.3, 0.59, 0.11], axis=2)
plt.subplot(1, 2, 1)
plt.imshow(img_data)
plt.subplot(1, 2, 2)
plt.imshow(gray_img_data, cmap='gray')
plt.show()
```

Biz bu örnekte şekli zaten siyah beyaz çizdiğimiz için asıl resimle gri tonlamalı resim arasında önemli farklar bulunmamaktadır:



Şimdi predict işlemi yapalım:

```
gray_img_data = gray_img_data / 255
gray_img_data = gray_img_data.reshape((1, 28 * 28))
predict_result = model.predict(gray_img_data)
number = np.argmax(predict_result[0])
print(number)
```

Şu sonuç elde edilmiştir:

4

Aşağıda MNIST örneğinin tüm kodunu veriyoruz. Buradaki "test.jpg" resmi Paint ile 28x28 boyutunda oluşturulmuştur:

```
from tensorflow.keras.datasets import mnist

(training_dataset_x, training_dataset_y), (test_dataset_x, test_dataset_y) = mnist.load_data()

import matplotlib.pyplot as plt

figure = plt.gcf()
figure.set_size_inches(10, 10)
for i in range(1, 10):
    plt.subplot(3, 3, i)
    axis = plt.gca()
    axis.set_title(str(training_dataset_y[i]))
    plt.imshow(training_dataset_x[i], cmap='gray')

plt.show()

training_dataset_x = training_dataset_x / 255
test_dataset_x = test_dataset_x / 255

training_dataset_x = training_dataset_x.reshape(-1, 28 * 28)
test_dataset_x = test_dataset_x.reshape(-1, 28 * 28)

from tensorflow.keras.utils import to_categorical

training_dataset_y = to_categorical(training_dataset_y)
test_dataset_y = to_categorical(test_dataset_y)

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential()
model.add(Dense(512, input_dim=28 * 28, activation='relu', name='Hidden-1'))
model.add(Dense(256, activation='relu', name='Hidden-2'))
model.add(Dense(10, activation='softmax', name='Output'))

model.compile('adam', loss='categorical_crossentropy', metrics=['categorical_accuracy'])
hist = model.fit(training_dataset_x, training_dataset_y, epochs=10, batch_size=64,
validation_split=0.2)

eval_result = model.evaluate(test_dataset_x, test_dataset_y)
for i in range(len(eval_result)):
    print(f'{model.metrics_names[i]} --> {eval_result[i]}')

import numpy as np

img_data = plt.imread('test.jpg')
gray_img_data = np.average(img_data, weights=[0.3, 0.59, 0.11], axis=2)
plt.subplot(1, 2, 1)
plt.imshow(img_data)
plt.subplot(1, 2, 2)
plt.imshow(gray_img_data, cmap='gray')
plt.show()

gray_img_data = gray_img_data / 255
gray_img_data = gray_img_data.reshape((1, 28 * 28))
predict_result = model.predict(gray_img_data)
number = np.argmax(predict_result[0])

print(number)
```

Keras içerisinde bulunan MNIST veri kümesi aslında CSV dosyası olarak da çeşitli yerlerden indirilebilir. Örneğin bunun için <https://www.kaggle.com/oddrationale/mnist-in-csv> adresini kullanabilirsiniz. Buradan "mnist_train.csv" ve "mnist_test.csv" dosyalarını elde edeceksiniz. Bu dosyalardan hareketle yukarıdaki örneği aşağıdaki gibi oluşturabilirsiniz:

```
import numpy as np

training_dataset = np.loadtxt('mnist_train.csv', delimiter=',', skiprows=1, dtype=np.uint8)

training_dataset_x = training_dataset[:, 1:]
training_dataset_y = training_dataset[:, 0]

test_dataset = np.loadtxt('mnist_test.csv', delimiter=',', skiprows=1, dtype=np.uint8)

test_dataset_x = test_dataset[:, 1:]
test_dataset_y = test_dataset[:, 0]

import matplotlib.pyplot as plt

figure = plt.gcf()
figure.set_size_inches(10, 10)
for i in range(1, 10):
    plt.subplot(3, 3, i)
    axis = plt.gca()
    axis.set_title(str(training_dataset_y[i]))
    plt.imshow(training_dataset_x[i].reshape(28, 28), cmap='gray')

plt.show()

training_dataset_x = training_dataset_x / 255
test_dataset_x = test_dataset_x / 255

from tensorflow.keras.utils import to_categorical

training_dataset_y = to_categorical(training_dataset_y)
test_dataset_y = to_categorical(test_dataset_y)

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential(name='MNIST')
model.add(Dense(256, input_dim=training_dataset_x.shape[1], activation='relu', name='Hidden-1'))
model.add(Dense(128, activation='relu', name='Hidden-2'))
model.add(Dense(10, activation='softmax', name='Output'))
model.compile(optimizer='rmsprop', loss='categorical_crossentropy',
metrics=['categorical_accuracy'])
hist = model.fit(training_dataset_x, training_dataset_y, batch_size=32, epochs=20,
validation_split=0.2)

import matplotlib.pyplot as plt

figure = plt.gcf()
figure.set_size_inches((15, 5))
plt.title('Loss - Epoch Graphics')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.plot(range(1, len(hist.history['loss']) + 1), hist.history['loss'])
plt.plot(range(1, len(hist.history['val_loss']) + 1), hist.history['val_loss'])
plt.legend(['Loss', 'Validation Loss'])
plt.show()

figure = plt.gcf()
```

```

figure.set_size_inches((15, 5))
plt.title('Categorical Accuracy - Epoch Graphics')
plt.xlabel('Epoch')
plt.ylabel('Categorical Accuracy')
plt.plot(range(1, len(hist.history['categorical_accuracy']) + 1),
hist.history['categorical_accuracy'])
plt.plot(range(1, len(hist.history['val_categorical_accuracy']) + 1),
hist.history['val_categorical_accuracy']))
plt.legend(['Categorical Accuracy', 'Validation Categorical Accuracy'])
plt.show()

eval_result = model.evaluate(test_dataset_x, test_dataset_y)
for i in range(len(eval_result)):
    print(f'{model.metrics_names[i]} ---> {eval_result[i]}')

```

Evrişimsel Sinir Ağları (Convolutional Neural Networks)

Sayısal sinyal işleme (digital signal processing) alanında evrişim (convolution) işlemleri ile çokça karşılaşılmaktadır. Yapay sinir ağlarında evrişim işlemi başta görsel veriler olmak üzere işitsel (audio) ve hareketsel (video) verilerine uygulanan bir tekniktir. Evrişimin görüntü verilerine uygulanması en yaygın kullanım biçimidir. Evrişim işlemi sayesinde görüntü yerel ögelere duyarlı hale getirilebilmektedir.

Evrişim işlemi evrişime sokulacak görüntü matrisinin filtre matrisi ya da "kernel" denilen küçük bir matrisle kaydırılmış biçimde işleme sokulmasıyla gerçekleştirilmektedir. Evrişim işlemi gri tonlu görüntü verileri üzerinde tipik olarak şöyle yürütülmektedir: Evrişime sokulacak orijinal görüntünün yanı sıra evrişim işleminde kullanılacak birfiltreleme (kernel) matrisi oluşturulur. Sonra bufiltreleme matrisi görüntünün üzerine bindirilerek işleme sokulur. Her işlem sonucunda tek bir değer elde edilmektedir. Sonra bufiltreleme matrisi sağa ve aşağıya kaydırılarak işlemler devam ettirilir. Filtreleme matrisi ile görüntünün ilgili kısmı üzerinde yapılan işlem genellikle "dot product" biçimindedir. Yanifiltreleme matrisindeki elemanlar asıl resmin çakıştırıldığı yerdeki elemanlarla çarpılarak toplanır ve bu toplamdan bir değer elde edilir. Örneğin asıl remin gri tonlu pixel'leri şöyle olsun:

```

a11 a12 a13 a14 a15 a16
a21 a22 a23 a24 a25 a26
a31 a32 a33 a34 a35 a36
a41 a42 a43 a44 a45 a46
a51 a52 a53 a54 a55 a56
a61 a62 a63 a64 a65 a66

```

Seçtiğimizfiltreleme matrisi de aşağıdaki gibi 3x3'lük olsun:

```

b11 b12 b13
b21 b22 b23
b31 b32 b33

```

Şimdi bufiltreleme matrisi asıl görüntü matrisinin sol-üst köşesine oturularak "dot-product" işlemi yapılır:

```

c11 = b11 * a11 + b12 * a12 + b13 * a13 + b21 * a21 + b22 * a22 + b23 * a23 + b31 * a31 + b32 * a32 + b33 * a33

```

Burada elde edilen değer evrişim işlemlerinin sonunda elde edilecek görüntü matrisinin c₁₁ elemanını oluşturacaktır. Sonra bufiltreleme matrisi asıl görüntü matrisi üzerinde sağa ve aşağıya kaydırılarak aynı işlemler yinelenir. Örneğinfiltreleme matrisini 1 sağa kaydırılmış olalım:

```

c12 = b11 * a12 + b12 * a12 + b13 * a14 + b21 * a22 + b22 * a23 + b23 * a24 + b31 * a32 + b32 * a34 + b33 * a34

```

Burada elde edilen değer görüntü matrisinin c₁₂'inci elemanı olacaktır. Bu işlemde böyle devam edildiğinde evrişilmiş hedef matrisin 4x4'lük olacağına dikkat ediniz. Bir yukarıdaki örnekte görüntü matrisini vefiltre matrisini 3X3'lük kare

matris olarak almış olsak da aslında görüntü matrisi ve filtre matrisi kare matris olmak zorunda değildir. Bu durumda evrişilmiş hedef matrisin boyutu için şu ifadeleri yazılabiliriz:

Evrişilmiş matrisin satır uzunluğu = Görüntü matrisinin satır uzunluğu - filtre matrisinin satır uzunluğu + 1

Evrişilmiş matrisin sütun uzunluğu = Görüntü matrisinin sütun uzunluğu - filtre matrisinin sütun uzunluğu + 1

Filtre matrisi gerhangi bir boyutta olabiliyorsa da genel olarak filtre matrisinin tek sayılı (3x3, 5x5, 7x7 gibi) bir kare matris olarak alındığını da belirtmek istiyoruz.

Filtre matrisini ana matris ile çalıştığımızda filtre matrisinin orta noktası için hedef matrisin hücresinin elde edildiğine dikkat ediniz. Bu nedenle evrişim işleminden daha küçük bir görüntü matrisi elde edilmektedir. Eğer evrişilmiş hedef matrisin görüntü matrisi ile aynı büyüklükte olması isteniyorsa görüntü matrisinin filtreleme matrisinin satır ve sütun uzunluğuna dayalı olarak iki taraftan uzatılması gereklidir. Bu uzatma işlemi çeşitli biçimlerde yapılmaktadır. Örneğin sol sütunun soluna, sağ sütunun sağına, üst satırın yukarısına, alt satırın aşağısına içi sıfırlarla dolu sütunlar ve satırlar eklenerek asıl şekil evrişim amaçlı büyütülebilir. Ya da sıfır eklemek yerine son satır ya da sütunlar çoğaltılmaktadır. Bu işleme genel olarak İngilizce "padding" denilmektedir.

Evrişim işleminde kaydırma birer birer yapılmak zorunda değildir. Bu kaydırma değerine İngilizce "stride" denilmektedir. Stride değerinin 1 olması kaydırmanın birer birer yapılacaka anlamına gelir. Eğer stride değeri yükseltilirse evrişilmiş matris küçülecektir. Aynı zamanda resimdeki yerel ilişkilerin uzaklıği da artırılmış olacaktır.

Aşağıda gri tonlamalı bir resim üzerinde evrişim işlemi yapan örnek bir Python programı verilmektedir:

```
import numpy as np

def conv(image, filter):
    image_height = image.shape[0]
    image_width = image.shape[1]

    filter_height = filter.shape[0]
    filter_width = filter.shape[1]

    conv_height = image_height - filter_height + 1
    conv_width = image_width - filter_width + 1

    conv_image = np.zeros((conv_height, conv_width))

    for row in range(conv_height):
        for col in range(conv_width):
            for i in range(filter_height):
                for j in range(filter_width):
                    conv_image[row, col] += image[row + i, col + j] * filter[i, j]

    conv_image[conv_image > 255] = 255
    conv_image[conv_image < 0] = 0

    return conv_image
```

Görüntü işleme uygulamalarında evrişim işlemi resmi filtrelemek için sık kullanılmaktadır. Bu bağlamda çeşitli filtre matrisleri resimleri farklı biçimlerde filtre etmek için kullanılmaktadır. Örneğin bir resmi iki farklı filtreye sokup evrişim işlemi yapalım:

```
import matplotlib.pyplot as plt
from matplotlib.image import imread

img_data = imread('AbbeyRoad.jpg')
gray_img_data = np.average(img_data, weights=[0.3, 0.59, 0.11], axis=2)
```

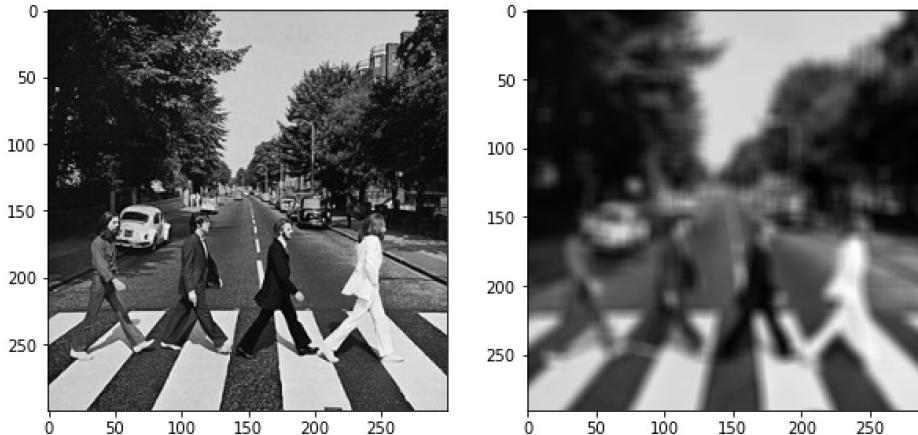
```

blur_filter = np.full((10, 10), 1 / 100)
img_conv = conv(gray_img_data, blur_filter)

figure = plt.gcf()
figure.set_size_inches((10, 10))
plt.subplot(1, 2, 1)
plt.imshow(gray_img_data, cmap='gray')
plt.subplot(1, 2, 2)
plt.imshow(img_conv, cmap='gray')
plt.show()

```

Bu filtrelemeden şu görüntüler elde edilmiştir:



Şimdi de başka bir filtre deneyelim:

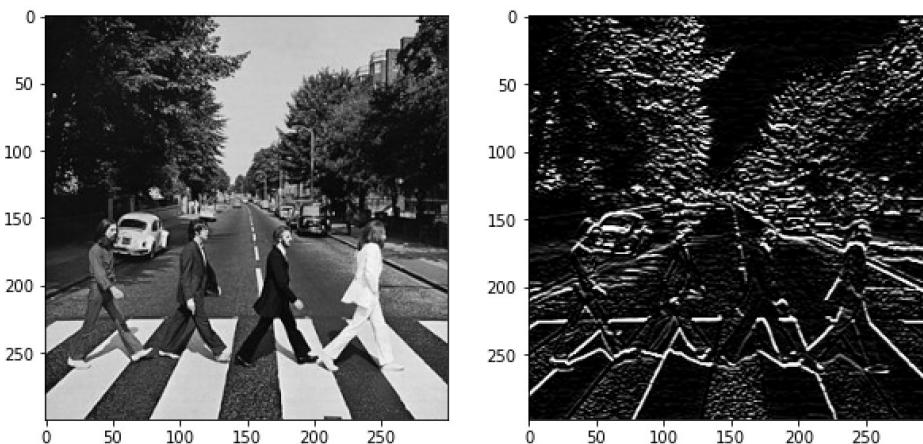
```

sobel_filter = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]])
img_conv = conv(gray_img_data, sobel_filter)

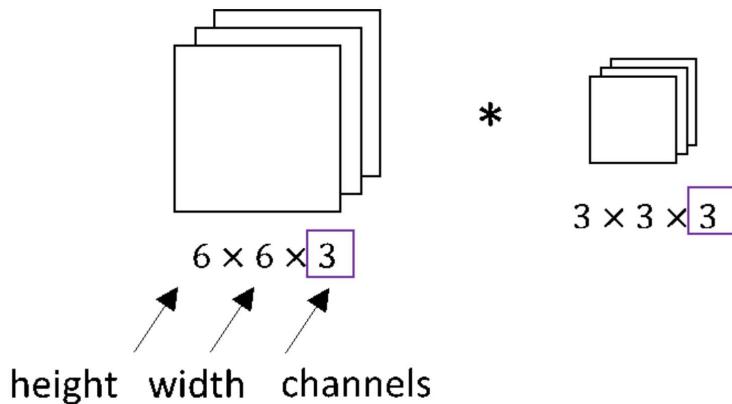
figure = plt.gcf()
figure.set_size_inches((10, 10))
plt.subplot(1, 2, 1)
plt.imshow(gray_img_data, cmap='gray')
plt.subplot(1, 2, 2)
plt.imshow(img_conv, cmap='gray')
plt.show()

```

Buradan da şu görüntüler elde edilmiştir:



Evrişim işlemi gri tonlamalı resimler yerine renkli RGB'li resimler üzerinde de yapılabilmektedir. Gri tonlamalı resimlerdeki NXN'lik filtreleme matrisleri renkli resimlerde NXNX3 biçiminde üç katmanlı olmaktadır. Örneğin gri tonlamalı resimlerdeki 3x3 boyutundaki filtreleme matrisleri renkli resimlerde 3X3X3 boyutunda olur. Bu matrisin her elemanı sırasıyla R, G ve B bileşenleri için kullanılacak 3X3'lük filtreleme matrislerini belirtir. Renkli resimlerde evrişim işlemi sinir ağları için genellikle iki biçimde yapılmaktadır. Birincisinde dot product işleminde sanki üç ayrı resim varmış gibi her renk bileşeni farklı bir filtre matrisiyle dot product işlemeye sokulur, buradan hedef pixel'i belirten 3 değer elde edilir. Örneğin:



Alıntı Notu: Görsel <http://datahacker.rs/convolution-rgb-image/> sitesinden alınmıştır.

İkincisinde yukarıdaki işlemler yapılır. Yani her renk bileşeni ile farklı bir filtre matrisi dot product işlemeye sokulur ancak buradan elde edilen 3 değer toplanarak teke indirilir. Yani bu yöntemde hedef resim RGB olmaktan çıkarılıp gri tonlamalı hale getirilir. Keras renkli resimler üzerinde evrişimi bu ikinci biçimde olduğu gibi yapmaktadır.

Pekiyi evrişim işlemi yapay sinir ağlarına nasıl uygulanmaktadır? Görüntü işlemede kişi belli filtreleri belirleyerek bunu resme uygulamaktadır. Halbuki yapay sinir ağlarında filtreleme matrisinin kendisini sinir ağı bulmaktadır. Yani işlemler tersten yapılmaktadır. Biz evrişim işlemini yaptığımızı varsayıarak bir ağ modeli oluştururuz. Sonra ağımızı eğitiriz. Bu eğitime göre ağ kendisi bir filtreleme matrisini oluşturmuş olur. Çünkü yapay sinir ağlarında biz aslında neyin filtreleneceğini bilmemekteyiz. Halbuki klasik görüntü işlemede görüntüyü işleyen kişinin bunu bir biçimde bildiği varsayılmaktadır. Yapay sinir ağlarında evrişim işleminde kullanılacak filtreleme matrisindeki değerlerin de ağırlık ve bias değerleri gibi tahmin edilecek değişken sayısına ekleneceğine dikkat ediniz.

Evrişimsel sinir ağlarında evrişim katmanlarında yapılan şey aslında her nöronu birbirine bağlamak (dense bağlantısı) yerine bazı nöronları birbirine bağlamaktır. Böylece biz her filtreleme matrisi ile evrişim işlemi yaptığımızda buradaki dot-product aslında nöron bağlantıları anlamına gelmektedir. Örneğin şeklimiz yine 6X6'lık olsun ve biz de yine 3x3'lük bir filtre kullanacak olalım:

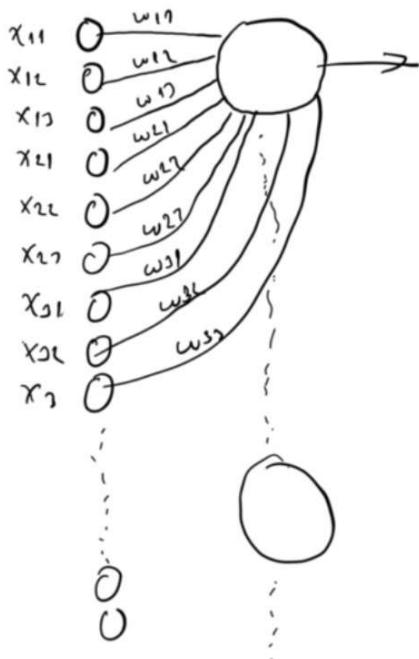
```
X11 X12 X13 X14 X15 X16
X21 X22 X23 X24 X25 X26
X31 X32 X33 X34 X35 X36
X41 X42 X43 X44 X45 X46
X51 X52 X53 X54 X55 X56
X61 X62 X63 X64 X65 X66
```

```
W11 W12 W13
W21 W22 W23
W31 W32 W33
```

Buradaki filtreleme işlemine dikkatlice bakalım. Örneğin sol-üst köşe hedef matris pixel'i için yapılacak dot-product işlemi şöyledir:

```
x11 * w11 + x12 * w12 + x13 * w13 + x21 * w21 + x22 * w22 + x23 * w23 + x31 * w31 + x32 * w32 +
x33 * w33
```

Bu işlem aslında bir nöron giriş toplam işlemi ile aynıdır. Örneğin:



Burada görüldüğü gibi aslında katmandaki nöron bağlantısı filtreleme matrisi ile asıl resmin ilgili kısımlarının dot-product yapılması ile eşdeğerdir. Başka bir deyişle aslında evrişim işlemi bir katman görevi yapar. Evrişim işlemini yapan katmana yapay sinir ağlarında "evrişim katmanı (convolution layer)" denilmektedir. Ancak bu evrişim katmanında bazı girdi nöronları bazı ağırlıklarla işleme girmektedir.

Pekiyyi yapay sinir ağlarındaki evrişim katmanlarındaki nöron sayıları ve tahmin edilecek parametre sayıları nasıl olacaktır? Evrişim katmanındaki nöron sayısı evrişim sonucunda elde edilecek görüntünün pixel sayısı kadar olacaktır. Çünkü evrişim sırasında filtre matrisini sağa ve aşağıya her kaydırduğumda aslında evrişim katmanında yeni bir nöron oluşturmuş gibi oluruz. Daha önce de belirttiğimiz gibi biz padding durumuna göre hedef görüntü matrisini asıl matrisle aynı boyutta ya da ondan daha küçük bir boyutta elde edebilmekteyiz. Yukarıdaki son örneğimizde ana resim 6x6 boyutunda filtre matrisi de 3x3 boyutundaydı. Bu durumda hedef görüntü matrisi padding yapılmışsa 6x6 boyutunda, padding yapılmamışsa 4x4 boyutunda olacaktır. Dolayısıyla evrişim katmanında da padding yapılmışsa 36 nöron, padding yapılmamışsa 16 nöron bulunacaktır. Öte yandan evrişim katmanında tahmin edilecek parametre sayısı filtre matrisindeki eleman sayısından 1 fazla olacaktır. Çünkü biz filtre matrisini sağa ve aşağı kaydırduğumda aslında filtre matrisindeki hep aynı değerlerle evrişim yapmış olmaktadır. Dolayısıyla yukarıdaki örnekte 3x3'lük filtre matrisi için padding yapılsın ya da yapılmamasın filtre matrisi nedeniyle tahmin edilecek parametre sayısı 9 olacaktır. Evrişim katmanında oluşan nöronların hepsinin bias değeri aynıdır. Yani nasıl her nöronda tahmin edilecek parametreler aslında aynıysa her nörondaki bias değeri de aynıdır. Bu nedenle evrişim katmanındaki toplam parametre sayısı filtre matrisinin eleman sayısından 1 fazla olmaktadır. (Nöronlarda bias değerlerinin kullanılıp kullanılmayacağıının da aslında katman sınıflarındaki `use_bias` parametresiyle ayarlandığını anımsayınız. Bu parametrenin default değeri True biçimindedir.)

Pekiyyi tek bir filtreleme işlemi şeklin daha iyi anlaşılmırması için yeterli olmakta mıdır? Genellikle hayır. Çünkü tek bir filtreleme şekli ancak bazı yönleriyle filtreleyemektedir. Filtreleme sayısını atrırırsa (örneğin 32 gibi) şekil başka yönlerden de tanınablecektir. Dolayısıyla evrişimsel sinir ağlarında evrişim katmanı genellikle tek bir filtre için değil birden fazla filtre için oluşturulmaktadır. Pekiyyi yukarıdaki 6x6'lık resme 32 tane filtre yerleştirirsek evrişimin katmanın parametrik yapısı nasıl olur? Bu durumda evrişim katmanındaki nöron sayısı padding yapılmışsa $36 * 32$ tane, padding yapılmamışsa $16 * 32$ tane olacaktır. Nöronlardaki toplam tahmin edilecek parametre sayısı da $9 * 32 + 32$ tane olacaktır.

Evrişimsel ağlarda evrişim katmanında yine dot-product sonucunun bir aktivasyon fonksiyonuna sokulmaktadır. Evrişim katmanları için de genellikle diğer saklı katmanlarda olduğu gibi "relu" aktivasyon fonksiyonu tercih edilmektedir.

Pekiyi evrişimsel sinir ağlarında evrişim katmanlarının sayısı birden fazla olabilir mi? İşte bir tane evrişimsel katman resimdeki küçük yerelikleri tanıyalıbmaktedir. Halbuki evrişimsel katmanların çıktılarını başka evrişimsel katmanlara bağladığımızda (yanifiltrelemenin üzerine yenidenfiltreleme uyguladığımızda) resmin tanınabildiği yerel bölgeyi büyütmiş oluruz. Uygulamada veri bilimcisi genellikle evrişimsel katmanlardaki nöron sayılarını ikiye katlayarak birden fazla katman oluşturmaktadır. (Yani ilk katman 32, sonrası 64, 128 biçiminde.)

Ayrıca görüntü tanımaya ilişkin ağlarda yalnızca evrişimsel katmanların kullanılması da uygun değildir. Çünkü bu durumda resmin bütününe ilişkin global özellikler gözden kaçırılmış olur. Bu nedenle bu tür ağların mimarilerinde genellikle önce birkaç evrişimsel katman daha sonra bir ya da birden fazla dense katman kullanılmaktadır. Böylece hem yerel öğeler hem de global özellikler ağa tarafından tanınabilmektedir.

Aslında evrişim işlemi tek boyutlu veriler üzerinde de uygulanabilmektedir. Tabii bu durumdafiltreleme matrisi de matris değil tek boyutlu bir dizi olacaktır. Örneğin evrişim işlemine sokacağımız dizi şöyle olsun:

2 3 5 1 4 6

Kullanacağımız filtre dizisi de şöyledir:

1 2 3

Bu filtre dizisini aynı biçimde asıl dizi üzerinde kaydırarak dot product işlemi yaparsak şöyledir:

23 15 19 27

Keras'ta Evrişimsel Sinir Ağlarının Oluşturulması

Keras yüksek seviyeli bir kütüphane olduğu için evrişim işlemi de yüksek seviyeli bir biçimde gerçekleştirilmektedir. Programcı yalnızca filtre matrislerinin boyutunu, sayısını, padding yapılmış yapılmayacağı ve stride değerini belirtmektedir. Tabii aslında bu değerlerin bazıları default argüman almış durumdadır. Evrişim işlemi tek boyutlu, iki boyutlu ya da üç boyutlu olarak da yapılabilmektedir. Biz burada iki boyutlu evrişim işlemi üzerinde duracağız.

Keras'ta iki boyutlu evrişim işlemleri için Conv2D sınıfı kullanılmaktadır. Yani tipki Dense nesnesi gibi programcı Conv2D nesnelerini yaratarak modeline ekler. Conv2D sınıfının `__init__` metodunun parametrik yapısı şöyledir:

```
tf.keras.layers.Conv2D(  
    filters,  
    kernel_size,  
    strides=(1, 1),  
    padding='valid',  
    data_format=None,  
    dilation_rate=(1, 1),  
    groups=1,  
    activation=None,  
    use_bias=True,  
    kernel_initializer='glorot_uniform',  
    bias_initializer='zeros',  
    kernel_regularizer=None,  
    bias_regularizer=None,  
    activity_regularizer=None,  
    kernel_constraint=None,  
    bias_constraint=None,  
    **kwargs  
)
```

Fonksiyonun birinci parametresi kullanılacak filtre sayısını belirtmektedir. İkinci parametre filtre matrisinin boyutlarını belirtir. Bu parametre demet olarak girilmelidir. Fonksiyonun stride parametresi stride miktarını yatay ve düşey olarak bir demet biçiminde alır. Bu parametrenin default değerinin (1, 1) olduğuna dikkat ediniz. padding parametresi ya 'valid' değerini alabilir ya da 'same' değerini alabilir. Bu parametrede 'valid' değeri padding yapılmayacağını, 'same'

değeri ise ise padding yapılacağını belirtmektedir. Ayrıca fonksiyonda bir input_shape parametresinin olduğunu da görürorsunuz. Bu parametre girdi katmanın boyutunu belirlemekte kullanılmaktadır. Girdi katmanı Conv2D katmanında matrisel biçimde üç boyutlu olarak girilmelidir. Bu üç boyutun birinci ve ikinci boyutları resmin genişlik ve yüksekliğini, üçüncü boyut ise resmin kanal sayısını (yani resimdeki pixel'lerin kaç rengin bileşimi ile ifade edildiğini) belirtmektedir. Örneğin gri tonlamalı MNIST resimleri için input_shape = (28, 28, 1) biçiminde olmalıdır.

Conv2D katmanından elde edilen çıktı üç boyutlu olmaktadır. Bu nedenle iki Conv2D katmanı birbirine bağlanabilir. Dense katmanın tek boyutlu girdi beklediğini anımsayınız. Bu nedenle Conv2D katmanın çıktı Dense katmana bağlanacaksça üç boyutlu çıktının tek boyuta dönüştürülmesi gereklidir. Bu dönüştürme Flatten isimli katmanla yapılmaktadır. Flatten sınıfının __init__ metodunun parametrik yapısı şöyledir:

```
tf.keras.layers.Flatten(  
    data_format=None,  
    **kwargs  
)
```

Metot tipik olarak parametresiz kullanılmaktadır. Flatten katmanı çok boyutlu girdiyi tek boyuta dönüştürmekten başka bir şey yapmaz. Yani bir çeşit adaptör katman görevindedir.

Şimdi öğrendiklerimizden hareketle MNIST örneğinin evrişimsel modelini oluşturmaya çalışalım. Önce modeli yükleyelim:

```
from tensorflow.keras.datasets import mnist  
  
(training_set_x, training_set_y), (test_set_x, test_set_y) = mnist.load_data()
```

Burada training_dataset_x (60000, 28, 28) boyutundadır. Conv2D katman nesnesini kullanabilmemiz için bizim bu matrisi (60000, 28, 28, 1) haline getirmemiz gereklidir. Çünkü Conv2D katmanı üç boyutlu bir matrisi girdi olarak istemektedir. Aynı işlemi test_dataset_x için de yapmalıyız:

```
training_dataset_x = training_dataset_x.reshape(-1, 28, 28, 1)  
test_dataset_x = test_dataset_x.reshape(-1, 28, 28, 1)
```

Şimdi Min-Max Ölçeklendirmesini yapalım:

```
training_dataset_x = training_dataset_x / 255  
test_dataset_x = test_dataset_x / 255
```

Şimdi de kategorik y verileri üzerinde one hot encoding işlemini uygulayalım:

```
from tensorflow.keras.utils import to_categorical  
  
training_dataset_y = to_categorical(training_dataset_y)  
test_dataset_y = to_categorical(test_dataset_y)
```

Artık modelimizi oluşturabiliriz:

```
from tensorflow.keras.models import Sequential  
from tensorflow.keras.layers import Conv2D, Dense, Flatten  
  
model = Sequential()  
model.add(Conv2D(32, kernel_size=(3, 3), padding='same', input_shape=(28, 28, 1),  
activation='relu', name='Convolution-1'))  
model.add(Conv2D(64, kernel_size=(3, 3), padding='same', activation='relu', name='Convolution-2'))  
model.add(Flatten(name='Flatten'))  
model.add(Dense(128, activation='relu', name='Hidden-1'))  
model.add(Dense(128, activation='relu', name='Hidden-2'))  
model.add(Dense(10, activation='softmax', name='Output'))
```

```
model.summary()
```

Buradan elde edilen özet şöyledir:

Layer (type)	Output Shape	Param #
=====		
Convolution-1 (Conv2D)	(None, 28, 28, 32)	320
Convolution-2 (Conv2D)	(None, 28, 28, 64)	18496
Flatten (Flatten)	(None, 50176)	0
Hidden-1 (Dense)	(None, 128)	6422656
Hidden-2 (Dense)	(None, 128)	16512
Output (Dense)	(None, 10)	1290
=====		
Total params:	6,459,274	
Trainable params:	6,459,274	
Non-trainable params:	0	

Pekiyi buradaki parametre değerleri nereden geliyor? Birinci evrişimsel katmanda 3×3 'luk bir filtre matrisi kullandık. Bu katmandaki nöronların hepsi için ortak bir bias değeri de olacağına göre bu katmanda tahmin edilecek parametre sayısı $32 * 9 + 32 = 320$ tane olacaktır. Evrişimsel katmandaki padding parametresinin 'same' biçiminde girildiğine dikkat ediniz. Bu durumda bu katmanın çıkışında $(28, 28, 1)$ boyutlarına sahip 32 nöron bulunacaktır. O halde ikinci evrişimsel katmanın girdisi $(28, 28, 32)$ biçiminde bir matris olacaktır. İkinci katmanda 3×3 'luk 64 filtre matrisinin kullanıldığını görürsünüz. Birinci katmanın çıkışında 28×28 'lik toplam 32 resim vardır. Bu resimlerden her biri farklı bir filtre matrisiyle işleme sokulacağına göre ve bu katmanda da 64 filtre kullanılacağına göre bu katmanda tahmin edilecek parametre sayısı $64 * 32 * 9 + 64 = 18496$ olacaktır. Pekiyi ikinci evrişim katmanın çıkışında kaç nöron vardır? Bu katmanın girişinde 28×28 'lik resimlerden 32 tane bulunmaktadır. Ancak evrişim katmanlarının girdilerinin hepsine aynı filtreler uygulanmaktadır. Dolayısıyla aslında bu 32 resmin hepsine aynı filtreler uygulanacaktır. Bu ikinci katmanda 64 tane filtre olduğuna göre bu katmanın çıkışında toplam $28 * 28 * 64 = 50176$ nöron bulunur. İkinci evrişim katmanından sonra Flatten katmanı ile bu katmandaki nöronların tek boyutlu hale getirildiğini görürsünüz. Bundan sonra modelde 128 nöronluğ bir Dense katman kullanılmıştır. Böylece bu Dense katmanda tahmin edilecek parametrelerin sayısı da $50176 * 128 + 128 = 6422656$ olur. İlk Dense katmanın çıkışında 128 nöronun bulunduğuna dikkat ediniz. Bu durumda ikinci Dense katmanda tahmin edilecek parametrelerin sayısı $128 * 128 + 128 = 16512$ tanedir. İkinci Dense katmanın çıkışında yine 128 nöron olduğuna göre ve bu 128 nöron çıktı katmanındaki nöronlarla dense bağlılığına göre çıktı katmanında tahmin edilecek parametrelerin sayısı $128 * 10 + 10 = 1290$ 'dır.

Şimdi modelimizi eğitelim:

```
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['categorical_accuracy'])
hist = model.fit(training_dataset_x, training_dataset_y, epochs=10, batch_size=64,
validation_split=0.2)
```

Epoch grafiklerini çizdirelim:

```
import matplotlib.pyplot as plt

figure = plt.gcf()
figure.set_size_inches((15, 5))
plt.title('Loss - Epoch Graphics')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.plot(range(1, len(hist.history['loss']) + 1), hist.history['loss'])
plt.plot(range(1, len(hist.history['val_loss']) + 1), hist.history['val_loss'])
plt.legend(['Loss', 'Validation Loss'])
```

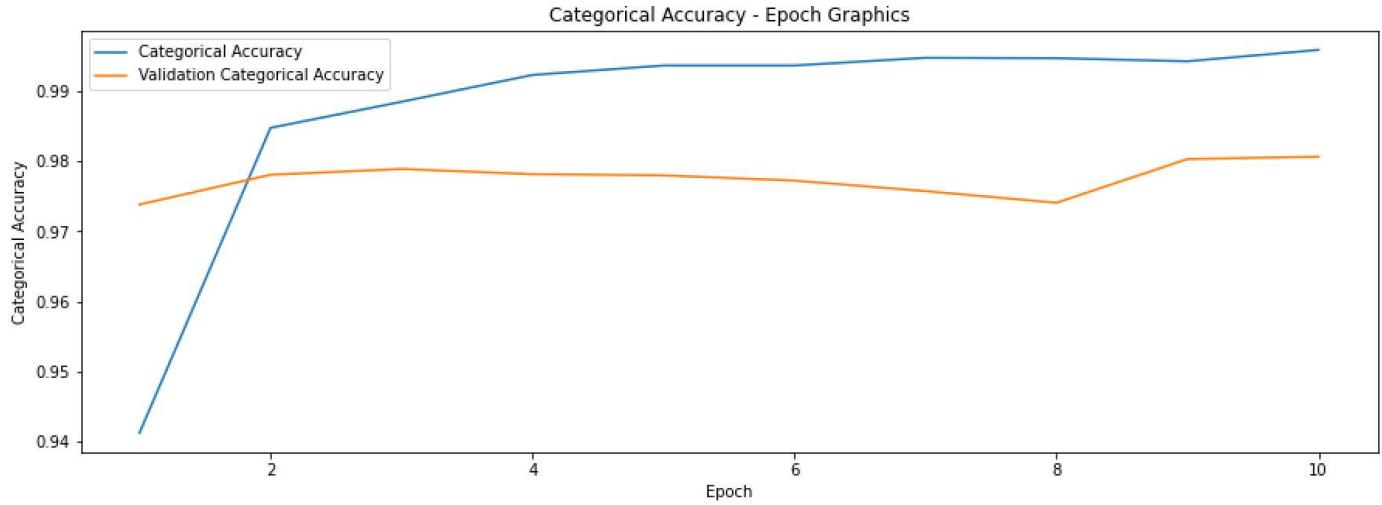
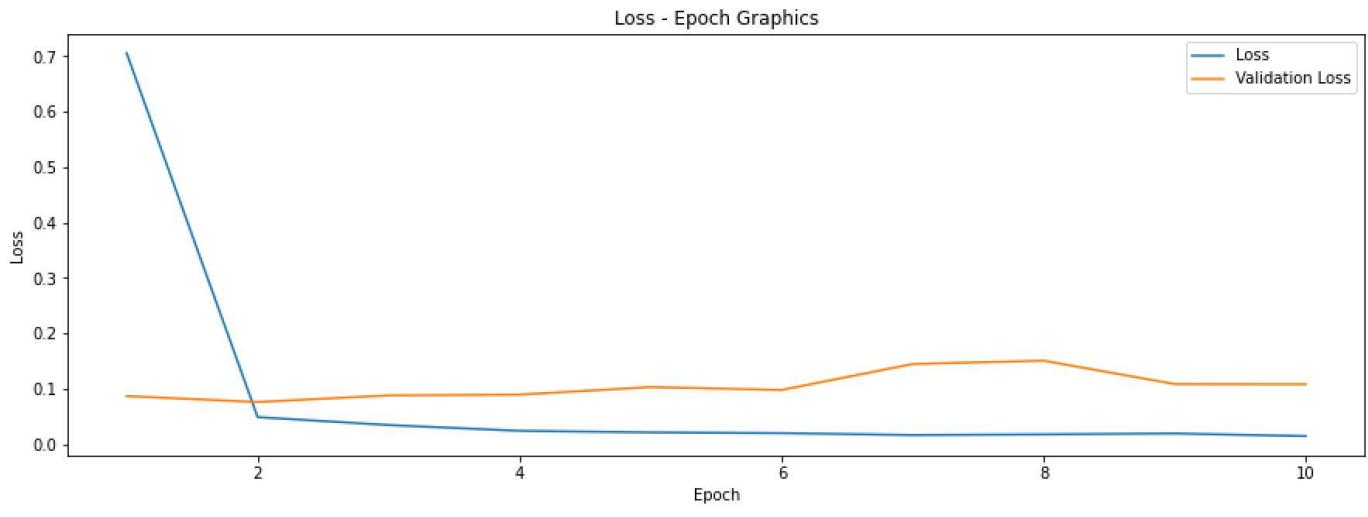
```

plt.show()

figure = plt.gcf()
figure.set_size_inches((15, 5))
plt.title('Categorical Accuracy - Epoch Graphics')
plt.xlabel('Epoch')
plt.ylabel('Categorical Accuracy')
plt.plot(range(1, len(hist.history['categorical_accuracy']) + 1),
hist.history['categorical_accuracy'])
plt.plot(range(1, len(hist.history['val_categorical_accuracy']) + 1),
hist.history['val_categorical_accuracy'])
plt.legend(['Categorical Accuracy', 'Validation Categorical Accuracy'])
plt.show()

```

Epoch-Accuracy grafiğini de çizelim:



Grafiklerden eğitim için 2 epoch işleminin yeterli olduğu görülmektedir. Modelimizi bu haliyle test edelim:

```

eval_result = model.evaluate(test_dataset_x, test_dataset_y)
for i in range(len(eval_result)):
    print(f'{model.metrics_names[i]} --> {eval_result[i]}')

```

Şu sonuçlar elde edilmiştir:

```

loss --> 0.11871606111526489
categorical_accuracy --> 0.9775999784469604

```

Yukarıdaki örneğin kaynak kodlarını bütün halinde söyle verebiliriz:

```
from tensorflow.keras.datasets import mnist

(training_dataset_x, training_dataset_y), (test_dataset_x, test_dataset_y) = mnist.load_data()

training_dataset_x = training_dataset_x.reshape(-1, 28, 28, 1)
test_dataset_x = test_dataset_x.reshape(-1, 28, 28, 1)

from tensorflow.keras.utils import to_categorical

training_dataset_y = to_categorical(training_dataset_y)
test_dataset_y = to_categorical(test_dataset_y)

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, Dense, Flatten

model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), padding='same', input_shape=(28, 28, 1),
activation='relu', name='Convolution-1'))
model.add(Conv2D(64, kernel_size=(3, 3), padding='same', activation='relu', name='Convolution-2'))
model.add(Flatten(name='Flatten'))
model.add(Dense(128, activation='relu', name='Hidden-1'))
model.add(Dense(128, activation='relu', name='Hidden-2'))
model.add(Dense(10, activation='softmax', name='Output'))

model.summary()

model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['categorical_accuracy'])
hist = model.fit(training_dataset_x, training_dataset_y, epochs=10, batch_size=64,
validation_split=0.2)

import matplotlib.pyplot as plt

figure = plt.gcf()
figure.set_size_inches((15, 5))
plt.title('Loss - Epoch Graphics')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.plot(range(1, len(hist.history['loss']) + 1), hist.history['loss'])
plt.plot(range(1, len(hist.history['val_loss']) + 1), hist.history['val_loss'])
plt.legend(['Loss', 'Validation Loss'])
plt.show()

figure = plt.gcf()
figure.set_size_inches((15, 5))
plt.title('Categorical Accuracy - Epoch Graphics')
plt.xlabel('Epoch')
plt.ylabel('Categorical Accuracy')
plt.plot(range(1, len(hist.history['categorical_accuracy']) + 1),
hist.history['categorical_accuracy'])
plt.plot(range(1, len(hist.history['val_categorical_accuracy']) + 1),
hist.history['val_categorical_accuracy'])
plt.legend(['Categorical Accuracy', 'Validation Categorical Accuracy'])
plt.show()

eval_result = model.evaluate(test_dataset_x, test_dataset_y)
for i in range(len(eval_result)):
    print(f'{model.metrics_names[i]} --> {eval_result[i]}'")
```

Evrişim işlemi tek boyutlu veriler üzerinde de yapılabilmektedir. Tabii bu durumda evrişime sokulacak olan veri ve evrişimde kullanılacak filtre de tek boyutlu olur. Örneğin evrişime sokulacak vektör şöyle olsun:

```
[4, 6, 2, 8, 5, 3, 9]
```

Filtre vektörünün de şöyle olduğunu varsayıyalım:

```
[1, 2, 3]
```

Bu durumda padding yapılmadığı durumda stride=1 için evrişimden elde edilecek vektör de şöyle olacaktır:

```
[22, 34, 33, 27, 40]
```

Evrişim işlemi zamansal (temporal) veriler üzerinde de uygulanabilmektedir. Burada zamansallık söz konusu değerler arasında öncelik sonralık ilişkisinin olması anlamına gelmektedir. Örneğin yazılarındaki sözcükler, videolardaki görüntüyü oluşturan frame'ler, EEG, EKG verileri zamansal verilere örnek olarak verilebilir.

Keras'ta parçalı (zamansal) vektörler üzerinde evrişim işlemi yapmak için de Conv1D isimli bir katman sınıfı bulundurulmuştur. Conv1D sınıfının `__init__` metodunun parametrik yapısı şöyledir:

```
tensorflow.keras.layers.Conv1D(  
    filters,  
    kernel_size,  
    strides=1,  
    padding='valid',  
    data_format='channels_last',  
    dilation_rate=1,  
    groups=1,  
    activation=None,  
    use_bias=True,  
    kernel_initializer='glorot_uniform',  
    bias_initializer='zeros',  
    kernel_regularizer=None,  
    bias_regularizer=None,  
    activity_regularizer=None,  
    kernel_constraint=None,  
    bias_constraint=None,  
    **kwargs  
)
```

Metodun birinci parametresi kullanılacak toplam filtre sayısını, ikinci parametresi ise filtre genişliğini belirtmektedir. stride parametresi filtrenin kaydırma miktarını belirlemekte kullanılır. Yine padding parametresi 'valid' ya da 'same' biçiminde girilebilmektedir.

Conv1D katmanının girdisinin matrisin olması gereklidir. Ancak evrişim Conv2D katmanında olduğu gibi yapılmaz. Matrisin satır ve sütunları üzerinde yapılmaktadır. Bu nedenle Conv1D katmanı tipik olarak parçalı (zamansal) girdiler üzerinde uygulanmaktadır. Şimdi Conv1D katmanının yaptığı evrişim işlemini açıklayalım.

Conv1D katmanı için zamansal bir verinin 20 satır ve 10 sütundan oluştuğunu varsayıyalım. Verilerin her satırı yazındaki bir sözcük, videodaki bir frame gibi zamansal bir girdiyi temsil ediyor olsun. Bu verinin görünümü aşağıdakine benzer olacaktır:

```
[[116, 103, 32, 110, 78, 145, 250, 41, 8, 13],  
 [ 59, 61, 225, 25, 73, 224, 43, 46, 180, 208],  
 [232, 66, 76, 243, 233, 98, 191, 249, 209, 101],  
 [156, 190, 77, 58, 161, 246, 54, 250, 52, 105],  
 [ 13, 226, 25, 82, 74, 165, 181, 104, 17, 203],  
 [124, 241, 42, 163, 230, 97, 157, 13, 155, 207],  
 [ 96, 45, 137, 134, 189, 238, 132, 34, 162, 176],
```

```
.....
[235, 197, 185, 49, 164, 6, 229, 232, 8, 162],
[ 11, 192, 110, 73, 132, 130, 92, 234, 58, 198],
[174, 21, 213, 178, 237, 237, 159, 226, 165, 220]])
```

Buradaki satırlar aslında zamansal verileri temsil etmektedir. Dolayısıyla aslında ardışık satırlar veride peşi sıra gelen değerlere ilişkiliidir. O halde padding kullanılmayan evrişim işleminde satırlar üçer üçer işleme sokulacaktır. Yani ilk evrişimde ilk üç satır işleme sokulur:

```
[116, 103, 32, 110, 78, 145, 250, 41, 8, 13]
[ 59, 61, 225, 25, 73, 224, 43, 46, 180, 208]
[232, 66, 76, 243, 233, 98, 191, 249, 209, 101]
```

Sonra strides = 1 olduğuna göre 1 satır kaydırılarak sonraki 3 satır işleme sokulur:

```
[ 59, 61, 225, 25, 73, 224, 43, 46, 180, 208],
[232, 66, 76, 243, 233, 98, 191, 249, 209, 101],
[156, 190, 77, 58, 161, 246, 54, 250, 52, 105],
```

Bu işlem böyle devam ettirilir. Burada evrişime sokulacak verilerin 3×10 'luk bir matris olduğunu görüyorsunuz. Uygulamada bu matrisin her bir elemanı ayrı bir değerle dot product yapılmaktadır. Yani her ne kadar filtre uzunluğu 3 biçiminde belirtiliyorsa da aslında filtre 3×10 'luk gibi düşünülmeliidir.

Şimdi Keras'ta filtre genişliğinin 3, strides değerinin 1 olduğu padding yapılmış Conv1D katmanını kullanarak özet bilgileri görüntüleyelim. Model aşağıdaki gibi oluşturulmuş olsun:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D

model = Sequential(name='Conv1D-Test')
model.add(Conv1D(1, 3, padding='same', activation='relu', input_shape=(100, 10),
name='Conv1D'))
model.summary()
```

Şöyledir bir summary çıktısı elde edilmişdir:

Model: "Conv1D-Test"

Layer (type)	Output Shape	Param #
<hr/>		
Conv1D (Conv1D)	(None, 100, 1)	31
<hr/>		
Total params: 31		
Trainable params: 31		
Non-trainable params: 0		

Burada 100 satırlık her bir satırı 10 sütundan oluşan zamansal veriler söz konusudur. Bu durumda kullanılacak filtrenin eleman sayısı aslında $3 * 10 = 30$ tanedir. Dolayısıyla bias değerini debuna eklersek bu katmanda tahmin edilecek parametre sayısı 31 olacaktır. Şimdi aşağıdaki modele bakınız:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, Flatten, Dense

model = Sequential(name='Conv1D-Test')
model.add(Conv1D(1, 3, padding='same', activation='relu', input_shape=(100, 10),
name='Conv1D'))
model.add(Flatten(name='Flatten'))
model.add(Dense(128, activation='relu', name='Dense'))
```

```
model.add(Dense(1, activation='sigmoid', name='Output'))  
model.summary()
```

Buradan şu summary bilgileri elde edilmiştir:

Model: "Conv1D-Test"

Layer (type)	Output Shape	Param #
<hr/>		
Conv1D (Conv1D)	(None, 100, 1)	31
Flatten (Flatten)	(None, 100)	0
Dense (Dense)	(None, 128)	12928
Output (Dense)	(None, 1)	129
<hr/>		
Total params: 13,088		
Trainable params: 13,088		
Non-trainable params: 0		

Burada yine aynı Conv1D katmanı kullanılmıştır. Pekiyi Conv1D katmanın çıktılarında kaç nöron vardır? İşte aslında her 3'lü kaydırma bir nöronla temsil edildiğine göre ve padding='same' alındığına göre katmanın çıktılarında 100 nöron bulunmaktadır. Ancak Conv1D katmanın çıktısının tek boyutlu değil matrisel olduğuna da dikkat ediniz. Bu durumda biz Conv1D katmanın çıktısını Dense bir katmana doğrudan bağlayamayız. Önce bir Flatten işlemini yapmamız gereklidir.

Conv1D katmanında tahmin edilecek parametre sayısının filtre genişliği ile zamansal girdilerin sütun uzunlıklarından bir fazla olduğuna dikkat ediniz. Katmanın çıktı nöronlarının sayısı da eğer padding='same' yapılmışsa zamansal verilerdeki satır sayısı kadardır.

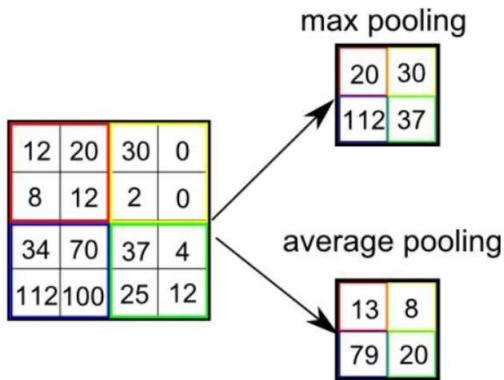
Evrişimsel Sinir Ağlarında Nöron Sayılarının Azaltılması (Downsampling)

Yukarıda da belirttiğimiz gibi evrişimsel ağlar özellikle görüntü tanımda çok kullanılmaktadır. Görüntü verileri de büyük olma eğilimindedir. (Her ne kadar MNIST görselleri 28x28'lik gri tonlamalı olsa da aslında gerçek yaşam uygulamalarında çoğu görsel daha yüksek çözünürlükte ve RGB biçimindedir.) Örneğin önceki bölümdeki MNIST modelinde oluşturduğumuz evrişimsel ağda 6 milyonun üzerinde ağırlık değerleri tahmin edilmeye çalışılmaktadır. Modelde çok nöronun olmasının şunun dezavantajları söz konusudur:

- Modelin eğitilmesi fazla zaman alır.
- Model verilerinin saklanması sırasında daha fazla disk alanına gereksinim duyulur.
- Modeldeki parametre sayılarının artması da "overfitting" kaynağı olabilmektedir.

İşte bu nedenlerden dolayı modeldeki nöron sayılarını azaltmak isteyebiliriz. Bunun için ilk akla gelen yöntem evrişimsel katmalardaki stride değerinin yükseltilmesidir. (Biz yukarıdaki örnekte default stride değerini 1 almıştık.) Ancak stride yönteminden daha etkin olduğu düşünülen "pooling" isimli bir yöntem vardır. Pooling yöntemi genellikle 2x2'lik bir matrisle ve stride = 2 olarak uygulanmaktadır.

Pooling işlemleri iki temel yöntemle yapılmaktadır: Max-Pooling ve Average-Pooling. Max-Pooling yönteminde pooling büyüğü kadarki (örneğin 2x2) matristeki en büyük eleman bulunur ve pooling büyüğündeki tüm değerler bu en büyük elemanla temsil edilir. Average-Pooling yönteminde ise pooling büyüğü kadarki elemanların ortalaması alınarak pooling büyüğündeki elemanlar bu ortalama değerle temsil edilmektedir. Max-Pooling yöntemi Average-Pooling yöntemine göre daha yaygın kullanılmaktadır. Örneğin:



Aıntı Notu: Görüel <https://medium.com/pursuitnotes/day-41-deep-learning-6-cnn-2-d38b355bd0ba> adresinden alınmıştır.

İki boyutlu matrisler üzerinde Max-Pooling ve Average-Pooling işlemleri için tensorflow.keras.layers modülünde MaxPooling2D ve AveragePooling2D isimli iki katman sınıfı bulunmaktadır. Bu sınıfların `__init__` metodlarının parametrik yapısı şöyledir:

```
tf.keras.layers.MaxPool2D(
    pool_size=(2, 2),
    strides=None,
    padding='valid',
    data_format=None,
    **kwargs
)

tf.keras.layers.AveragePooling2D(
    pool_size=(2, 2),
    strides=None,
    padding='valid',
    data_format=None,
    **kwargs
)
```

Metotların `pool_size` parametreleri pooling işleminde kullanılacak matrisin büyülüüğünü belirtmektedir. Bu parametrenin default değerinin `(2, 2)` olduğunu görürsünüz. `stride` parametreleri kaydırma değerlerini belirtmektedir. Buradaki `None` değeri default `stride` değerinin `2` olduğu anlamına gelmektedir. `padding` parametresi yine padding yapılmış yapılmayacağı belirtir. Bu parametrededeki '`valid`' değeri padding yapılmayacağı, '`same`' değeri padding yapılacak anlamına gelmektedir.

Evrişimsel sinir ağlarında pooling katmanları tipik olarak evrişim katmanlarından hemen sonra bulundurulmaktadır. O halde tipik bir evrişimsel sinir ağı modeli şöyle oluşturulmaktadır:



Şimdi yukarıdaki evrişimsel MNIST modelini pooling katmanlarını ekleyerek yeniden oluşturalım:

```
from tensorflow.keras.datasets import mnist

(training_dataset_x, training_dataset_y), (test_dataset_x, test_dataset_y) = mnist.load_data()

training_dataset_x = training_dataset_x.reshape(-1, 28, 28, 1)
test_dataset_x = test_dataset_x.reshape(-1, 28, 28, 1)

from tensorflow.keras.utils import to_categorical

training_dataset_y = to_categorical(training_dataset_y)
test_dataset_y = to_categorical(test_dataset_y)
```

```

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Flatten

model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), padding='same', input_shape=(28, 28, 1),
activation='relu', name='Convolution-1'))
model.add(MaxPooling2D(name='MaxPooling2D-1'))
model.add(Conv2D(64, kernel_size=(3, 3), padding='same', activation='relu', name='Convolution-2'))
model.add(MaxPooling2D(name='MaxPooling2D-2'))
model.add(Flatten(name='Flatten'))
model.add(Dense(128, activation='relu', name='Hidden-1'))
model.add(Dense(128, activation='relu', name='Hidden-2'))
model.add(Dense(10, activation='softmax', name='Output'))

model.summary()

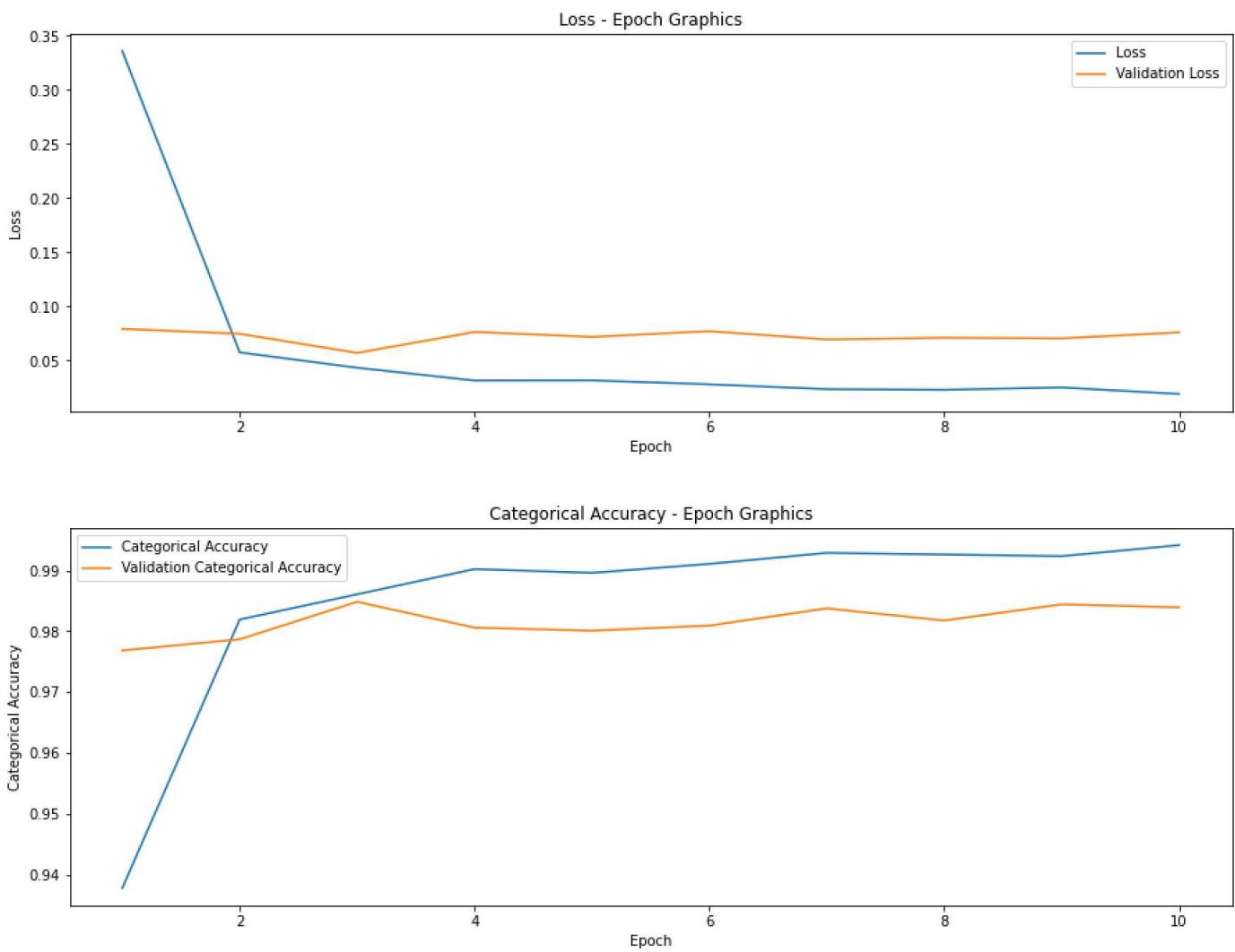
```

summary fonksiyonundan elde edilen çıktı şöyledir:

Layer (type)	Output Shape	Param #
<hr/>		
Convolution-1 (Conv2D)	(None, 28, 28, 32)	320
<hr/>		
MaxPooling2D-1 (MaxPooling2D)	(None, 14, 14, 32)	0
<hr/>		
Convolution-2 (Conv2D)	(None, 14, 14, 64)	18496
<hr/>		
MaxPooling2D-2 (MaxPooling2D)	(None, 7, 7, 64)	0
<hr/>		
Flatten (Flatten)	(None, 3136)	0
<hr/>		
Hidden-1 (Dense)	(None, 128)	401536
<hr/>		
Hidden-2 (Dense)	(None, 128)	16512
<hr/>		
Output (Dense)	(None, 10)	1290
<hr/>		
Total params: 438,154		
Trainable params: 438,154		
Non-trainable params: 0		

Bu özet bilgiden evrişim işlemi sonucunda elde edilen 28x28'luk matrislerin önce 14x14'e sonra da 7x7'ye indirgendiğini görüyorsunuz. Pooling işlemlerinden sonra modelin 6 milyon civarındaki parametre sayısı 400 bin civarına düşürülmüştür.

Şimdi modeli 10 epoch kullanarak eğitelim ve epoch grafiklerine bakalım:



Grafiklere göre eğitim 3 epoch'ta sonlandırılabilir. Şimdi modelimiz test edelim:

```
loss --> 0.0667557567358017
categorical_accuracy --> 0.984499990940094
```

Göründüğü gibi pooling işlemlerinden sonra hem model parametreleri azaltılmış hem de model biraz daha iyileştirilmiştir. Aşağıda örneğin tüm kodlarını yine bütünsel olarak veriyoruz:

```
from tensorflow.keras.datasets import mnist

(training_dataset_x, training_dataset_y), (test_dataset_x, test_dataset_y) = mnist.load_data()

training_dataset_x = training_dataset_x.reshape(-1, 28, 28, 1)
test_dataset_x = test_dataset_x.reshape(-1, 28, 28, 1)

from tensorflow.keras.utils import to_categorical

training_dataset_y = to_categorical(training_dataset_y)
test_dataset_y = to_categorical(test_dataset_y)

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Flatten

model = Sequential()
model.add(
    Conv2D(32, kernel_size=(3, 3), padding='same', input_shape=(28, 28, 1), activation='relu',
    name='Convolution-1'))
model.add(MaxPooling2D(name='MaxPooling2D-1'))
```

```

model.add(Conv2D(64, kernel_size=(3, 3), padding='same', activation='relu', name='Convolution-2'))
model.add(MaxPooling2D(name='MaxPooling2D-2'))
model.add(Flatten(name='Flatten'))
model.add(Dense(128, activation='relu', name='Hidden-1'))
model.add(Dense(128, activation='relu', name='Hidden-2'))
model.add(Dense(10, activation='softmax', name='Output'))

model.summary()

model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['categorical_accuracy'])
hist = model.fit(training_dataset_x, training_dataset_y, epochs=10, batch_size=64,
validation_split=0.2)

import matplotlib.pyplot as plt

figure = plt.gcf()
figure.set_size_inches((15, 5))
plt.title('Loss - Epoch Graphics')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.plot(range(1, len(hist.history['loss']) + 1), hist.history['loss'])
plt.plot(range(1, len(hist.history['val_loss']) + 1), hist.history['val_loss'])
plt.legend(['Loss', 'Validation Loss'])
plt.show()

figure = plt.gcf()
figure.set_size_inches((15, 5))
plt.title('Categorical Accuracy - Epoch Graphics')
plt.xlabel('Epoch')
plt.ylabel('Categorical Accuracy')
plt.plot(range(1, len(hist.history['categorical_accuracy']) + 1),
hist.history['categorical_accuracy'])
plt.plot(range(1, len(hist.history['val_categorical_accuracy']) + 1),
hist.history['val_categorical_accuracy'])
plt.legend(['Categorical Accuracy', 'Validation Categorical Accuracy'])
plt.show()

eval_result = model.evaluate(test_dataset_x, test_dataset_y)
for i in range(len(eval_result)):
    print(f'{model.metrics_names[i]} --> {eval_result[i]}')

```

Pooling işlemleri iki boyutlu görüntü verilerinin dışında tek boyutlu veriler üzerinde de uygulanabilmektedir. Tabii bu durumda pooling büyülüğu iki boyutlu bir matris yerine tek boyutlu bir değer olacaktır. Örneğin aşağıdaki gibi tek boyutlu 10 elemanlı bir vektör söz konusu olsun:

1, 4, 6, 3, 2, 9, 4, 6, 10, 3

Bu vektör üzerinde MaxPooling1D işlemini strides=2 için yaptığımda 5 elemanlı şı vektörü elde ederiz:

4, 6, 9, 6, 10

Keras'ta tek boyutlu pooling işlemleri için MaxPooling1D ve AveragePooling1D sınıfları bulundurulmuştur. Bu sınıfların __init__ metodlarının parametrik yapısı aşağıdaki gibidir:

```

tensorflow.keras.layers.MaxPooling1D(
    pool_size=2,
    strides=None,
    padding='valid',
    data_format='channels_last',

```

```

        **kwargs
    )

tensorflow.keras.layers.AveragePooling1D(
    pool_size=2,
    strides=None,
    padding='valid',
    data_format='channels_last',
    **kwargs
)

```

Metotların birinci parametreleri kullanılacak pool genişliğini belirlemekte kullanılmaktadır. Bu parametrenin default değerinin 2 olduğunu görüyorsunuz. Metotların strides parametreleri kaydırma miktarını belirtmektedir. None değer kaydırmanın pool genişliği kadar yapılacağını belirtir. padding parametreleri yine 'valid' ya da 'same' değerini alabilmektedir.

Renkli Resimlerin Sınıflandırılmasına İlişkin CIFAR-10 Örneği

CIFAR-10 Keras'in içerisinde de var olan eğitimde yaygın kullanılan bir görsel veri kümelerinden biridir. CIFAR-10 içerisinde 60000 tane her biri 32X32 boyutlarında bitmap resimler vardır. Bu resimlerdeki her pixel 3 byte olup Red, Green, Blue tonal bileşenlerini içermektedir. CIFAR-10 çok sınıfı regresyon problemleri için hazırlanmış bir veri kümeleridir. Burdaki her resim 10 sınıf içerisinde birine ilişkindir. Cifar-10'un 100 sınıf içeren CIFAR-100 biçiminde başka bir uyarlaması da bulunmaktadır.

CIFAR-10 içerisinde resimlerin ilişkin olabileceği sınıflar şunlardır:

```
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']
```

Şimdi CIFAR-10 için adım adım evrişimsel sinir ağı modelini oluşturalım. Önce veri kümесini yükleyelim:

```
from tensorflow.keras.datasets import cifar10

(training_dataset_x, training_dataset_y), (test_dataset_x, test_dataset_y) =
cifar10.load_data()

class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']
```

Okuduğumuz verilerin boyutsal özelliklerini yazdıralım:

```
print(f'training_dataset_x.shape: {training_dataset_x.shape}')
print(f'training_dataset_y.shape: {training_dataset_y.shape}')
print(f'test_dataset_x.shape: {test_dataset_x.shape}')
print(f'test_dataset_y.shape: {test_dataset_y.shape}')
```

Elde edilen çıktı şöyledir:

```
training_dataset_x.shape: (50000, 32, 32, 3)
training_dataset_y.shape: (50000, 1)
test_dataset_x.shape: (10000, 32, 32, 3)
test_dataset_y.shape: (10000, 1)
```

Sınıflar yine 0'dan 9'a kadar bir tamsayı ile ifade edilmiştir. Şimdi ilk 9 resmi deneme amaçlı çizdirelim:

```
import matplotlib.pyplot as plt

figure = plt.gcf()
figure.set_size_inches(10, 10)
for i in range(1, 10):
```

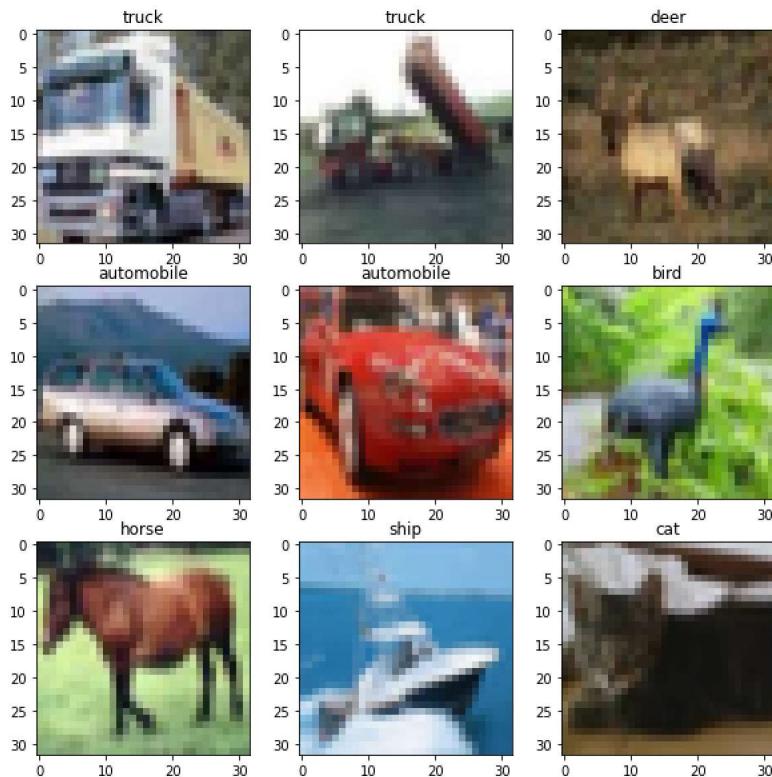
```

plt.subplot(3, 3, i)
axis = plt.gca()
axis.set_title(class_names[training_dataset_y[i, 0]])
plt.imshow(training_dataset_x[i], cmap='gray')

plt.show()

```

Elde edilen görüntü şöyledir:



Şimdi x verilerini Min-Max ölçeklendirmesine sokalım:

Min-max ölçeklendirimesini uygulayalım:

```

training_dataset_x = training_dataset_x / 255
test_dataset_x = test_dataset_x / 255

```

Şimdi training_dataset_y ve test_dataset_y değerleri üzerinde one hot encoding işlemi yapalım:

```

from tensorflow.keras.utils import to_categorical

training_dataset_y = to_categorical(training_dataset_y)
test_dataset_y = to_categorical(test_dataset_y)

```

Artık modelimizi oluşturabiliriz:

```

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, Dense, Flatten, MaxPooling2D

model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), input_shape=(32, 32, 3), activation='relu',
name='Convolution-1'))
model.add(MaxPooling2D(name='MaxPooling-1'))
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu', name='Convolution-2'))
model.add(MaxPooling2D(name='MaxPooling-2'))
model.add(Flatten(name='Flatten'))
model.add(Dense(128, activation='relu', name='Hidden-1'))

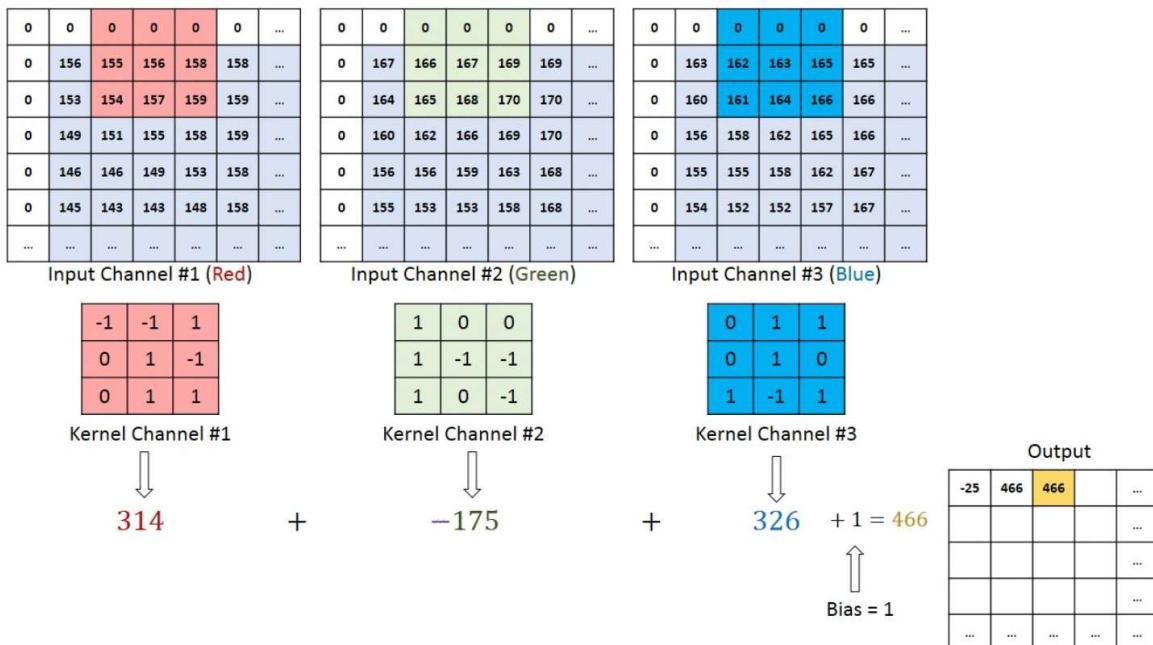
```

```
model.add(Dense(10, activation='softmax', name='Output'))  
model.summary()
```

Buradan elde edilen özet bilgiler şöyledir:

Layer (type)	Output Shape	Param #
<hr/>		
Convolution-1 (Conv2D)	(None, 32, 32, 32)	896
<hr/>		
Pooling-1 (MaxPooling2D)	(None, 16, 16, 32)	0
<hr/>		
Convolution-2 (Conv2D)	(None, 16, 16, 64)	18496
<hr/>		
Pooling-2 (MaxPooling2D)	(None, 8, 8, 64)	0
<hr/>		
Flatten (Flatten)	(None, 4096)	0
<hr/>		
Hidden-1 (Dense)	(None, 128)	524416
<hr/>		
Hidden-2 (Dense)	(None, 128)	16512
<hr/>		
Output (Dense)	(None, 10)	1290
<hr/>		
Total params: 561,610		
Trainable params: 561,610		
Non-trainable params: 0		

Öncelikle önce Keras'in Conv2D katmanının RGB resimler üzerinde nasıl evrişim işlemi uyguladığı hakkında bir açıklama yapmak istiyoruz. Keras RGB resimlerin her bir kanalı için ayrı filtreler kullanmakla birlikte bu ayrı filtrelerden elde ettiği dot product değerlerini toplayıp tek bir değer elde etmektedir. Yapılan işlemi aşağıdaki şekilde özetleyebiliriz:



Alıntı Notu: Görsel <https://stackoverflow.com/questions/43306323/keras-conv2d-and-input-channels> adresinden elde edilmiştir.

Yani Keras adeta RGB resimlerin evrişiminden sanki gri tonlamalı bir resim elde ediyor gibi davranmaktadır.

Şimdi de model özet bilgilerinde belirtilen değerlerin nereden geldiğine ilişkin bazı açıklamalar yapmak istiyoruz. Birinci evrişim katmanının girdisi 32x32'lik 3 kanallı resimdir. Bu katmanda 3x3'lük filtre matrisi kullanılmıştır. Her kanal için ayrı bir filtre oluşturulacağına göre ve toplam 32 farklı filtre olacağına göre bu katmanda tahmin edilecek parametrelerin sayısı $9 * 3 * 32 + 32 = 896$ olacaktır. Yukarıda da belirttiğimiz gibi evrişim işleminden 32X32X1'lik 32 matris oluşmaktadır. Toplamdaki 32 değeri katmanın çıktısından oluşturulacak 32 resim için gereken bias değerinden gelmektedir. Pooling-1 katmanında MaxPooling2D işlemi 2x2'lik bir matris için uygulanmıştır. Böylece 32x32x32'lik matris 16x16x32'ye düşürülmüştür. Sonra 32 resim Convolution-2 katmanına sokulmuştur. Bu katmanda tahmin edilecek parametre sayısı $32 * 9 * 64 + 64 = 18496$ olacaktır. Bu katmandan 16x16x1'lik 64 resim elde edilecektir. Bu resimler Pooling-2 katmanına sokulduğunda 8x8'lik 64 resme dönüştürülür. Sonra bu 8x8'lik 64 resim Flatten katmanıyla tek boyutlu hale getirilmiştir. Buradan toplam $64 * 64 = 4096$ tane nöron elde edilmektedir. Sonra bu 4096 nöron Dense-1 katmanına bağlanmıştır. Bu katmanda tahmin edilecek parametre sayısı $4096 * 128 + 128 = 524416$ olacaktır. Dense-2 katmanın 128 girişi olacağına göre o katmandaki tahmin edilecek parametre sayısı da $128 * 128 + 128 = 16512$ 'dir. Nihayet Output katmanının girdi sayısı 128 olduğuna göre orada tahmin edilecek parametre sayısı da $128 * 10 + 10 = 1290$ olacaktır.

Artık modelimizi derleyip eğitebiliriz:

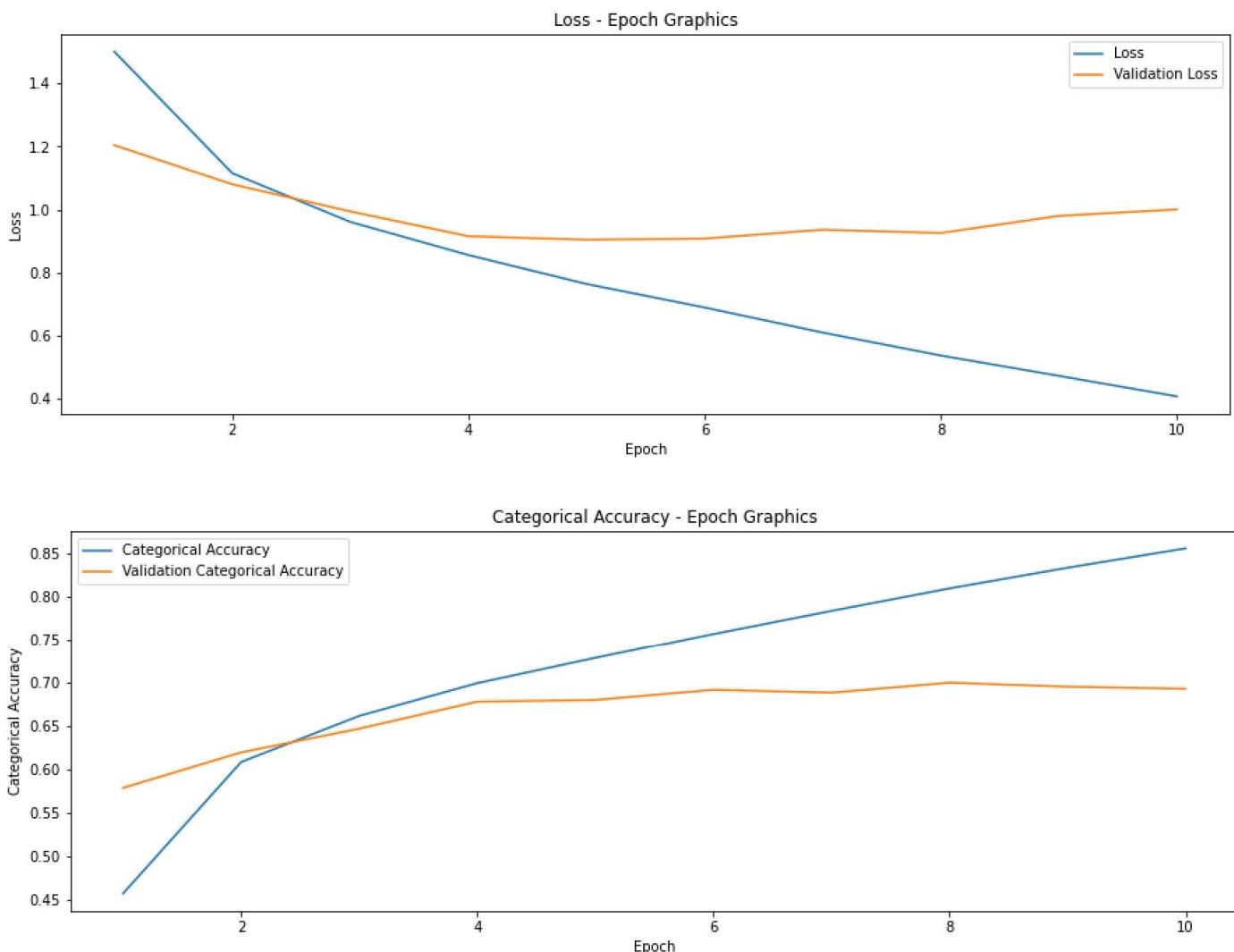
```
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['categorical_accuracy'])
hist = model.fit(training_dataset_x, training_dataset_y, epochs=10, batch_size=64,
validation_split=0.2)
```

Şimdi de Epoch grafiklerini çizdirelim:

```
import matplotlib.pyplot as plt

figure = plt.gcf()
figure.set_size_inches((15, 5))
plt.title('Loss - Epoch Graphics')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.plot(range(1, len(hist.history['loss']) + 1), hist.history['loss'])
plt.plot(range(1, len(hist.history['val_loss']) + 1), hist.history['val_loss'])
plt.legend(['Loss', 'Validation Loss'])
plt.show()

figure = plt.gcf()
figure.set_size_inches((15, 5))
plt.title('Categorical Accuracy - Epoch Graphics')
plt.xlabel('Epoch')
plt.ylabel('Categorical Accuracy')
plt.plot(range(1, len(hist.history['categorical_accuracy']) + 1),
hist.history['categorical_accuracy'])
plt.plot(range(1, len(hist.history['val_categorical_accuracy']) + 1),
hist.history['val_categorical_accuracy'])
plt.legend(['Categorical Accuracy', 'Validation Categorical Accuracy'])
plt.show()
```



Şimdi de modelimizi test edelim:

```
eval_result = model.evaluate(test_dataset_x, test_dataset_y)
for i in range(len(eval_result)):
    print(f'{model.metrics_names[i]} --> {eval_result[i]}')
```

Şöyledir bir sonuç elde edilmiş:

```
loss --> 1.016248106956482
categorical_accuracy --> 0.6923999786376953
```

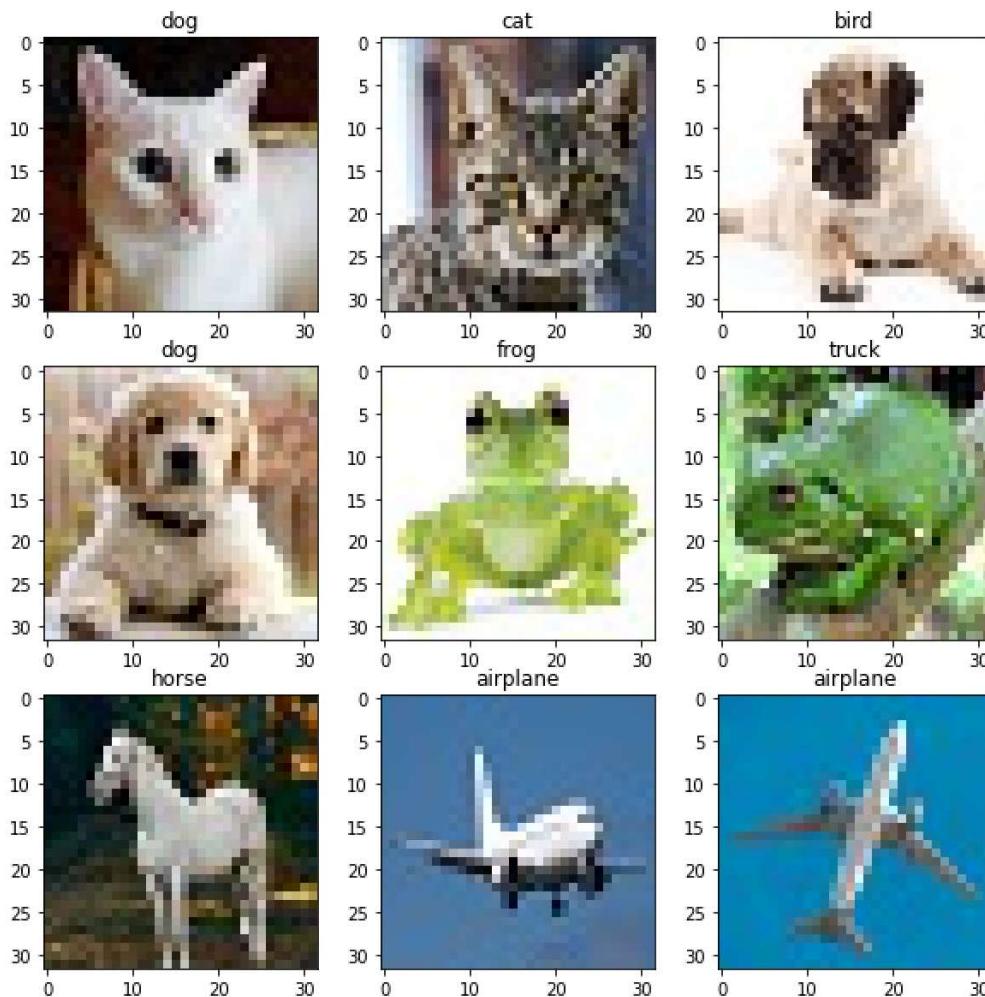
Şimdi de kendi bulduğumuz 32x32x3'lük 9 fotoğrafı sınıflandırmaya çalışalım:

```
import numpy as np
import itertools
import glob

figure = plt.gcf()
figure.set_size_inches((10, 10))
for index, path in enumerate(itertools.islice(glob.glob('*.*jpg'), 9)):
    img_data = plt.imread(path)
    scaled_img_data = img_data / 255
    result = model.predict(scaled_img_data.reshape(1, 32, 32, 3))
    number = np.argmax(result)
    plt.subplot(3, 3, index + 1)
    axis = plt.gca()
    axis.set_title(class_names[number])
```

```
plt.imshow(img_data)
plt.show()
```

Şöyledir bir sonuç elde edilmişdir:



CIFAR-10 uygulamasının kodlarını bütün olarak veriyoruz:

```
from tensorflow.keras.datasets import cifar10

(training_dataset_x, training_dataset_y), (test_dataset_x, test_dataset_y) =
cifar10.load_data()

class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship',
'truck']

print(f'training_dataset_x.shape: {training_dataset_x.shape}')
print(f'training_dataset_y.shape: {training_dataset_y.shape}')
print(f'test_dataset_x.shape: {test_dataset_x.shape}')
print(f'test_dataset_y.shape: {test_dataset_y.shape}')

import matplotlib.pyplot as plt

figure = plt.gcf()
figure.set_size_inches(10, 10)
for i in range(1, 10):
    plt.subplot(3, 3, i)
    axis = plt.gca()
    axis.set_title(class_names[training_dataset_y[i, 0]])
    plt.imshow(training_dataset_x[i], cmap='gray')
```

```

plt.show()

training_dataset_x = training_dataset_x / 255
test_dataset_x = test_dataset_x / 255

from tensorflow.keras.utils import to_categorical

training_dataset_y = to_categorical(training_dataset_y)
test_dataset_y = to_categorical(test_dataset_y)

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), padding='same', input_shape=(32, 32, 3),
activation='relu', name='Convolution-1'))
model.add(MaxPooling2D(name='Pooling-1'))
model.add(Conv2D(64, kernel_size=(3, 3), padding='same', activation='relu', name='Convolution-2'))
model.add(MaxPooling2D(name='Pooling-2'))
model.add(Flatten(name='Flatten'))
model.add(Dense(128, activation='relu', name='Hidden-1'))
model.add(Dense(128, activation='relu', name='Hidden-2'))
model.add(Dense(10, activation='softmax', name='Output'))

model.summary()

model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['categorical_accuracy'])
hist = model.fit(training_dataset_x, training_dataset_y, epochs=10, batch_size=64,
validation_split=0.2)

import matplotlib.pyplot as plt

figure = plt.gcf()
figure.set_size_inches((15, 5))
plt.title('Loss - Epoch Graphics')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.plot(range(1, len(hist.history['loss']) + 1), hist.history['loss'])
plt.plot(range(1, len(hist.history['val_loss']) + 1), hist.history['val_loss'])
plt.legend(['Loss', 'Validation Loss'])
plt.show()

figure = plt.gcf()
figure.set_size_inches((15, 5))
plt.title('Categorical Accuracy - Epoch Graphics')
plt.xlabel('Epoch')
plt.ylabel('Categorical Accuracy')
plt.plot(range(1, len(hist.history['categorical_accuracy']) + 1),
hist.history['categorical_accuracy'])
plt.plot(range(1, len(hist.history['val_categorical_accuracy']) + 1),
hist.history['val_categorical_accuracy'])
plt.legend(['Categorical Accuracy', 'Validation Categorical Accuracy'])
plt.show()

eval_result = model.evaluate(test_dataset_x, test_dataset_y)
for i in range(len(eval_result)):
    print(f'{model.metrics_names[i]} --> {eval_result[i]}')

import numpy as np
import glob

```

```

for path in glob.glob('*.*'):
    img_data = plt.imread(path)
    scaled_img_data = img_data / 255
    result = model.predict(scaled_img_data.reshape(1, 32, 32, 3))
    number = np.argmax(result)
    plt.title(class_names[number], fontsize=16)
    plt.imshow(img_data)
    plt.show()

```

Her ne kadar CIFAR-10 veri kümesi Keras'ın içinde hazır bulunuyorsa da CSV dosyası biçiminde de elde edilebilmektedir. CIFAR-10 veri kümesine ilişkin CSV dosyalarını ZIP dosyası olarak <https://www.kaggle.com/fedesoriano/cifar10-python-in-csv> adresinden indirebilirsiniz. İndirdiğiniz ZIP dosyasını açınca "train.csv" ve "test.csv" dosyalarını elde edeceksiniz. Bu dosyaların ilk satırında yine bir başlık kısmı vardır. Her satırındaki $3 * 32 * 32$ değer ilgili resmin RGB pixel değerlerini, son değer ise ilgili resmin sınıfını belirtmektedir. Burada dikkat edilmesi gereken resmin pixel verilerinin CSV içerisinde $32 \times 32 \times 3$ olarak değil de $3 \times 32 \times 32$ biçiminde oluşturulduğudur. Bu nedenle numpy.loadtxt fonksiyonu ile dosya okunduktan sonra transpose işleminin uygulanması gereklidir. Şimdi dosyaların çalışma dizininde olduğunu varsayılmış numpy.loadtxt fonksiyonu ile okumayı şöyle yapabiliriz:

```

import numpy as np

training_dataset = np.loadtxt('train.csv', delimiter=',', skiprows=1, dtype=np.uint8)

training_dataset_x = training_dataset[:, :-1].reshape(-1, 3, 32, 32)
training_dataset_x = np.transpose(training_dataset_x, (0, 2, 3, 1))
training_dataset_y = training_dataset[:, -1]

test_dataset = np.loadtxt('test.csv', delimiter=',', skiprows=1, dtype=np.uint8)

test_dataset_x = test_dataset[:, :].reshape(-1, 3, 32, 32)
test_dataset_x = np.transpose(test_dataset_x, (0, 2, 3, 1))
test_dataset_y = test_dataset[:, :].reshape(-1, 3, 32, 32)

class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']

import matplotlib.pyplot as plt

figure = plt.gcf()
figure.set_size_inches(10, 10)
for i in range(1, 10):
    plt.subplot(3, 3, i)
    axis = plt.gca()
    axis.set_title(str(training_dataset_y[i]))
    plt.imshow(training_dataset_x[i])

plt.show()

```

Kodun geri kalan kısmı yukarıdakini aynısıdır.

Renkli Resimlerin Sınıflandırılmasına İlişkin CIFAR-100 Örneği

CIFAR-100 veri kümesi CIFAR-10 veri kümesinin 100 sınıf içeren genişletilmiş bir biçimidir. Bu veri kümesinde 100 sınıfından her birine ilişkin $3 \times 32 \times 32$ 'lik 600 renkli resim bulunmaktadır. CIFAR-100 veri kümesinin organizasyonu 100 tane sınıfa sahip olmasının dışında tamamen CIFAR-10 veri kümesi ile aynıdır. CIFAR-100 veri kümesi içerisindeki resimlerin sınıfları şöyledir:

```

class_names = [
    'apple', 'aquarium_fish', 'baby', 'bear', 'beaver', 'bed', 'bee', 'beetle',
    'bicycle', 'bottle', 'bowl', 'boy', 'bridge', 'bus', 'butterfly', 'camel',

```

```

'can', 'castle', 'caterpillar', 'cattle', 'chair', 'chimpanzee', 'clock',
'cloud', 'cockroach', 'couch', 'crab', 'crocodile', 'cup', 'dinosaur',
'dolphin', 'elephant', 'flatfish', 'forest', 'fox', 'girl', 'hamster',
'house', 'kangaroo', 'keyboard', 'lamp', 'lawn_mower', 'leopard', 'lion',
'lizard', 'lobster', 'man', 'maple_tree', 'motorcycle', 'mountain', 'mouse',
'mushroom', 'oak_tree', 'orange', 'orchid', 'otter', 'palm_tree', 'pear',
'pickup_truck', 'pine_tree', 'plain', 'plate', 'poppy', 'porcupine',
'possum', 'rabbit', 'raccoon', 'ray', 'road', 'rocket', 'rose',
'sea', 'seal', 'shark', 'shrew', 'skunk', 'skyscraper', 'snail', 'snake',
'spider', 'squirrel', 'streetcar', 'sunflower', 'sweet_pepper', 'table',
'tank', 'telephone', 'television', 'tiger', 'tractor', 'train', 'trout',
'tulip', 'turtle', 'wardrobe', 'whale', 'willow_tree', 'wolf', 'woman',
>worm'
]

```

CIFAR-100 veri kümesi de tensorflow.keras.datasets.cifar100 modülü içerisinde hazır biçimde bulundurulmaktadır.
Veri kümesini biliriz

```

from tensorflow.keras.datasets import cifar100

(training_dataset_x, training_dataset_y), (test_dataset_x, test_dataset_y) =
cifar100.load_data()

class_names = [
    'apple', 'aquarium_fish', 'baby', 'beaver', 'bed', 'bee', 'beetle',
    'bicycle', 'bottle', 'bowl', 'boy', 'bridge', 'bus', 'butterfly', 'camel',
    'can', 'castle', 'caterpillar', 'cattle', 'chair', 'chimpanzee', 'clock',
    'cloud', 'cockroach', 'couch', 'crab', 'crocodile', 'cup', 'dinosaur',
    'dolphin', 'elephant', 'flatfish', 'forest', 'fox', 'girl', 'hamster',
    'house', 'kangaroo', 'keyboard', 'lamp', 'lawn_mower', 'leopard', 'lion',
    'lizard', 'lobster', 'man', 'maple_tree', 'motorcycle', 'mountain', 'mouse',
    'mushroom', 'oak_tree', 'orange', 'orchid', 'otter', 'palm_tree', 'pear',
    'pickup_truck', 'pine_tree', 'plain', 'plate', 'poppy', 'porcupine',
    'possum', 'rabbit', 'raccoon', 'ray', 'road', 'rocket', 'rose',
    'sea', 'seal', 'shark', 'shrew', 'skunk', 'skyscraper', 'snail', 'snake',
    'spider', 'squirrel', 'streetcar', 'sunflower', 'sweet_pepper', 'table',
    'tank', 'telephone', 'television', 'tiger', 'tractor', 'train', 'trout',
    'tulip', 'turtle', 'wardrobe', 'whale', 'willow_tree', 'wolf', 'woman',
    'worm'
]

print(f'training_dataset_x.shape: {training_dataset_x.shape}')
print(f'training_dataset_y.shape: {training_dataset_y.shape}')
print(f'test_dataset_x.shape: {test_dataset_x.shape}')
print(f'test_dataset_y.shape: {test_dataset_y.shape}')

```

Şu çıktı elde edilmiştir:

```

training_dataset_x.shape: (50000, 32, 32, 3)
training_dataset_y.shape: (50000, 1)
test_dataset_x.shape: (10000, 32, 32, 3)
test_dataset_y.shape: (10000, 1)

```

Veri kümesindeki resimlerden 9 tanesini görüntüleyelim:

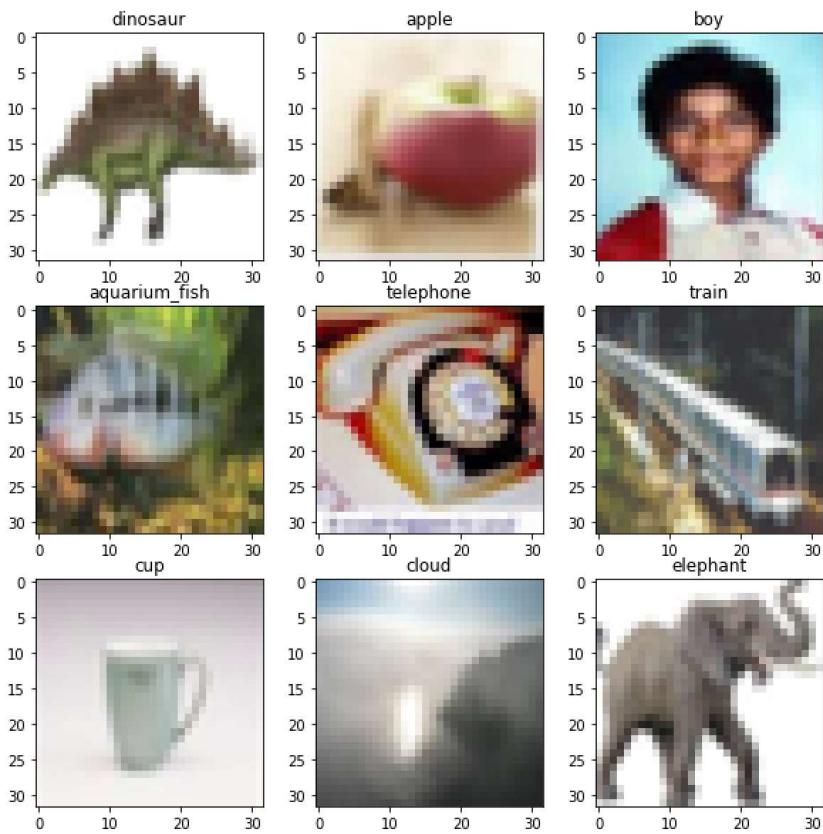
```

figure = plt.gcf()
figure.set_size_inches(10, 10)
for i in range(1, 10):
    plt.subplot(3, 3, i)
    axis = plt.gca()
    axis.set_title(class_names[training_dataset_y[i, 0]])

```

```
plt.imshow(training_dataset_x[i], cmap='gray')
plt.show()
```

Şöyledir bir görüntü elde edilmiştir:



Min-Max ölçeklendirmesini ve one hot encoding işlemlerini yapalım:

```
training_dataset_x = training_dataset_x / 255
test_dataset_x = test_dataset_x / 255

from tensorflow.keras.utils import to_categorical

training_dataset_y = to_categorical(training_dataset_y)
test_dataset_y = to_categorical(test_dataset_y)
```

Şimdi modelimizi oluşturalım. Modelimizde yine iki evrişim katmanı olsun. Evrişim katmanlarından sonra pooling katmanlarıyla nöron sayısını azaltalım. Bunları iki saklı katman ve çıkış katmanı izlesin:

```
from tensorflow.keras.utils import to_categorical

training_dataset_y = to_categorical(training_dataset_y)
test_dataset_y = to_categorical(test_dataset_y)

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

model = Sequential(name='CIFAR-100')
model.add(Conv2D(32, kernel_size=(3, 3), padding='same', input_shape=(32, 32, 3),
activation='relu', name='Convolution-1'))
model.add(MaxPooling2D(name='Pooling-1'))
model.add(Conv2D(64, kernel_size=(3, 3), padding='same', activation='relu', name='Convolution-2'))
model.add(MaxPooling2D(name='Pooling-2'))
model.add(Conv2D(64, kernel_size=(3, 3), padding='same', activation='relu', name='Convolution-3'))
```

```

3'))
model.add(MaxPooling2D(name='Pooling-3'))
model.add(Flatten(name='Flatten'))
model.add(Dense(256, activation='relu', name='Hidden-1'))
model.add(Dense(128, activation='relu', name='Hidden-2'))
model.add(Dense(100, activation='softmax', name='Output'))

model.summary()

```

Modelin özetini söylemek:

Layer (type)	Output Shape	Param #
<hr/>		
Convolution-1 (Conv2D)	(None, 32, 32, 32)	896
Pooling-1 (MaxPooling2D)	(None, 16, 16, 32)	0
Convolution-2 (Conv2D)	(None, 16, 16, 64)	18496
Pooling-2 (MaxPooling2D)	(None, 8, 8, 64)	0
Convolution-3 (Conv2D)	(None, 8, 8, 64)	36928
Pooling-3 (MaxPooling2D)	(None, 4, 4, 64)	0
Flatten (Flatten)	(None, 1024)	0
Hidden-1 (Dense)	(None, 256)	262400
Hidden-2 (Dense)	(None, 128)	32896
Output (Dense)	(None, 100)	12900
<hr/>		
Total params: 364,516		
Trainable params: 364,516		
Non-trainable params: 0		

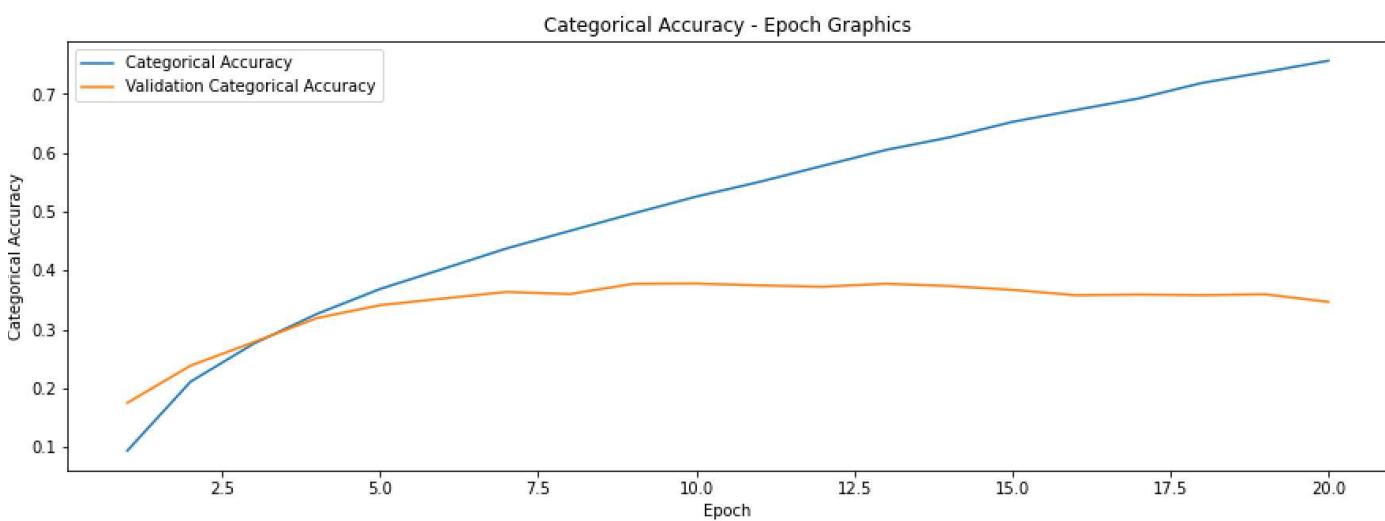
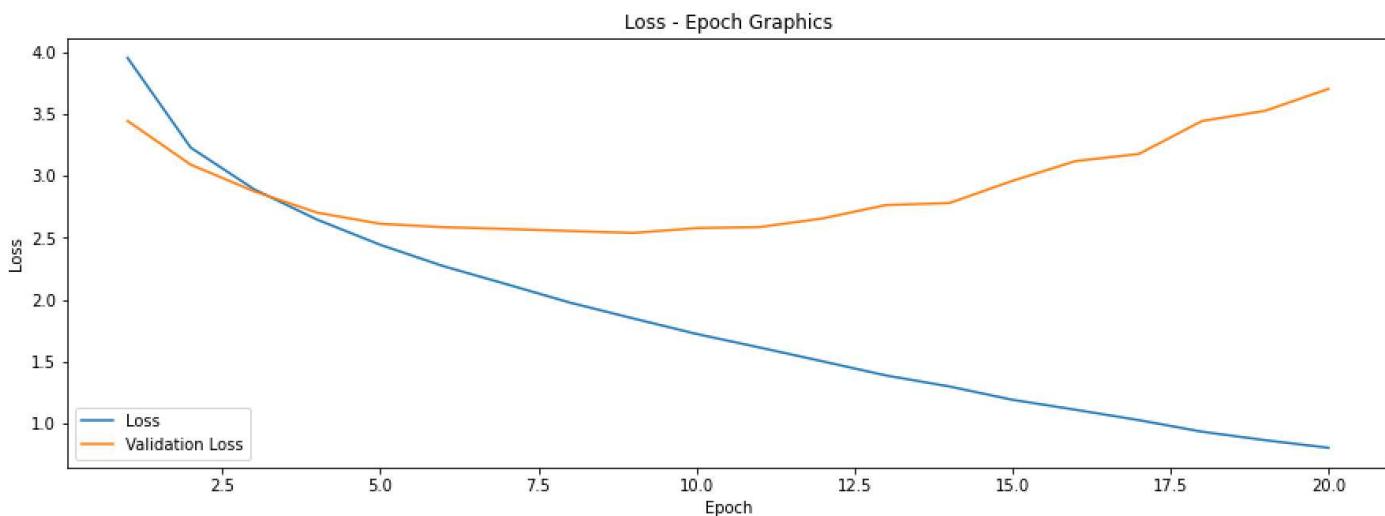
Modelimizi derleyelim ve eğitelim:

```

model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['categorical_accuracy'])
hist = model.fit(training_dataset_x, training_dataset_y, epochs=10, batch_size=64,
validation_split=0.2)

```

Epoch grafiklerini çizdirelim:



Burada 7 civarında bir epoch uygun gözükmemektedir. Şimdi de modelimizi test edelim:

```
eval_result = model.evaluate(test_dataset_x, test_dataset_y)
for i in range(len(eval_result)):
    print(f'{model.metrics_names[i]} --> {eval_result[i]}')
```

Şu sonuçlar elde edilmiştir:

```
loss --> 3.6690242290496826
categorical_accuracy --> 0.35409998893737793
```

Şimdi modeldeki 100 sınıfın bazlarına ilişkin resimleri bulup bunları 32x32x3 boyutuna getirdiğimizi varsayıyalım. Kestirim işlemini CIFAR-10 örneğindekine benzer biçimde yapabiliriz:

```
import numpy as np
import glob

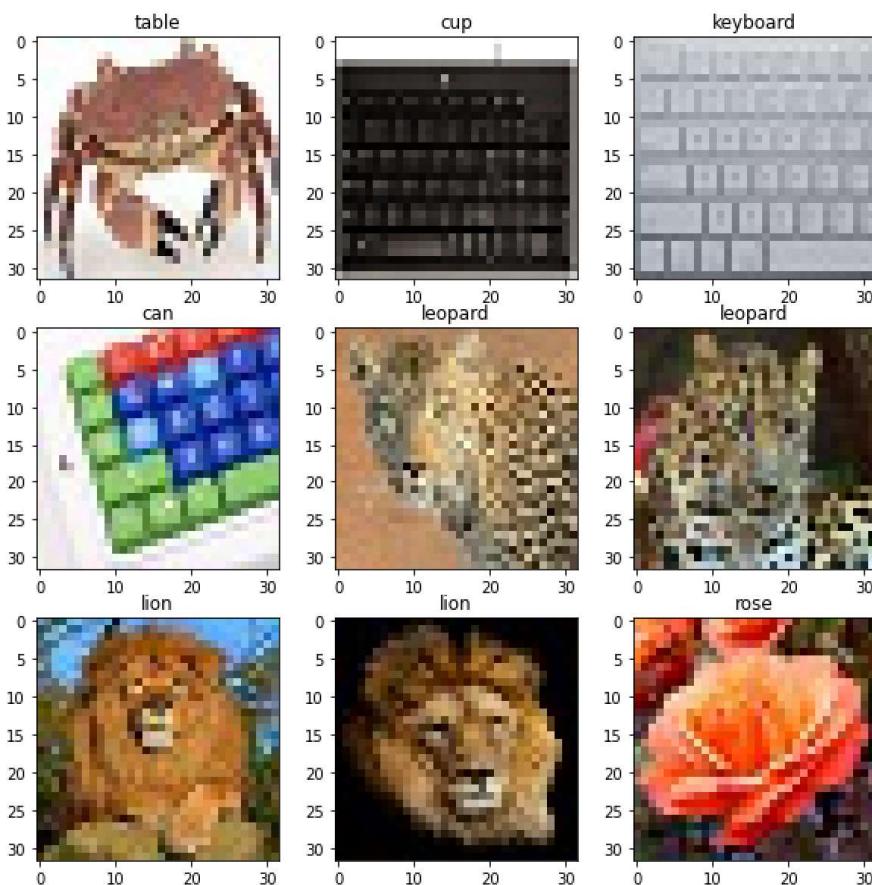
for path in glob.glob('*.*'):
    img_data = plt.imread(path)
    scaled_img_data = img_data / 255
    result = model.predict(scaled_img_data.reshape(1, 32, 32, 3))
    number = np.argmax(result)
    plt.title(class_names[number], fontsize=16)
    plt.imshow(img_data)
    plt.show()
```

Şimdi 9 tane resim için kestirim sonuçlarına bakalım:

```
import numpy as np
import itertools
import glob

figure = plt.gcf()
figure.set_size_inches((10, 10))
for index, path in enumerate(itertools.islice(glob.glob('*.*jpg'), 9)):
    img_data = plt.imread(path)
    scaled_img_data = img_data / 255
    result = model.predict(scaled_img_data.reshape(1, 32, 32, 3))
    number = np.argmax(result)
    plt.subplot(3, 3, index + 1)
    axis = plt.gca()
    axis.set_title(class_names[number])
    plt.imshow(img_data)
plt.show()
```

Şu sonuçlar elde edilmiştir:



Aşağıda CIFAR-100 örneğinin tüm kodlarını bütün olarak veriyoruz:

```
from tensorflow.keras.datasets import cifar100

(training_dataset_x, training_dataset_y), (test_dataset_x, test_dataset_y) =
cifar100.load_data()

class_names = [
    'apple', 'aquarium_fish', 'baby', 'bear', 'beaver', 'bed', 'bee', 'beetle',
    'bicycle', 'bottle', 'bowl', 'boy', 'bridge', 'bus', 'butterfly', 'camel',
    'can', 'castle', 'caterpillar', 'cattle', 'chair', 'chimpanzee', 'clock',
    'cloud', 'cockroach', 'couch', 'crab', 'crocodile', 'cup', 'dinosaur',
    'dive_bird', 'eagle', 'elephant', 'flatfish', 'garlic', 'giraffe', 'goldfish',
    'house_tea_kettle', 'kangaroo', 'lawn_mower', 'motorcycle', 'ostrich', 'palm',
    'panda', 'pear', 'raindeer', 'red_woodpecker', 'shark', 'sheep', 'skunk',
    'snail', 'spider', 'starfish', 'tiger', 'toad', 'zebra']
```

```

'dolphin', 'elephant', 'flatfish', 'forest', 'fox', 'girl', 'hamster',
'house', 'kangaroo', 'keyboard', 'lamp', 'lawn_mower', 'leopard', 'lion',
'lizard', 'lobster', 'man', 'maple_tree', 'motorcycle', 'mountain', 'mouse',
'mushroom', 'oak_tree', 'orange', 'orchid', 'otter', 'palm_tree', 'pear',
'pickup_truck', 'pine_tree', 'plain', 'plate', 'poppy', 'porcupine',
'possum', 'rabbit', 'raccoon', 'ray', 'road', 'rocket', 'rose',
'sea', 'seal', 'shark', 'shrew', 'skunk', 'skyscraper', 'snail', 'snake',
'spider', 'squirrel', 'streetcar', 'sunflower', 'sweet_pepper', 'table',
'tank', 'telephone', 'television', 'tiger', 'tractor', 'train', 'trout',
'tulip', 'turtle', 'wardrobe', 'whale', 'willow_tree', 'wolf', 'woman',
'worm'
]

import matplotlib.pyplot as plt

figure = plt.gcf()
figure.set_size_inches((10, 10))
for i in range(9):
    plt.subplot(3, 3, i + 1)
    axis = plt.gca()
    axis.set_title(class_names[training_dataset_y[i, 0]])
    plt.imshow(training_dataset_x[i])
plt.show()

index = class_names.index('oak_tree')
image_data = training_dataset_x[training_dataset_y[:, 0] == index][:20]
for i in range(20):
    plt.imshow(image_data[i])
    plt.show()

training_dataset_x = training_dataset_x / 255
test_dataset_x = test_dataset_x / 255

from tensorflow.keras.utils import to_categorical

training_dataset_y = to_categorical(training_dataset_y)
test_dataset_y = to_categorical(test_dataset_y)

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

model = Sequential(name='CIFAR-10')
model.add(
    Conv2D(32, kernel_size=(3, 3), padding='same', activation='relu', input_shape=(32, 32, 3),
name='Convolution-1'))
model.add(MaxPooling2D(name='Pooling-1'))
model.add(Conv2D(64, kernel_size=(3, 3), padding='same', activation='relu', name='Convolution-2'))
model.add(MaxPooling2D(name='Pooling-2'))
model.add(Conv2D(128, kernel_size=(3, 3), padding='same', activation='relu', name='Convolution-3'))
model.add(MaxPooling2D(name='Pooling-3'))
model.add(Flatten(name='Flatten'))
model.add(Dense(256, activation='relu', name='Dense-1'))
model.add(Dense(128, activation='relu', name='Dense-2'))
model.add(Dense(100, activation='softmax', name='Output'))

model.summary()

model.compile(optimizer='rmsprop', loss='categorical_crossentropy',
metrics=['categorical_accuracy'])
hist = model.fit(training_dataset_x, training_dataset_y, epochs=20, batch_size=64,
validation_split=0.2)

```

```

import matplotlib.pyplot as plt

figure = plt.gcf()
figure.set_size_inches((15, 5))
plt.title('Loss - Epoch Graphics')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.plot(range(1, len(hist.history['loss']) + 1), hist.history['loss'])
plt.plot(range(1, len(hist.history['val_loss']) + 1), hist.history['val_loss'])
plt.legend(['Loss', 'Validation Loss'])
plt.show()

figure = plt.gcf()
figure.set_size_inches((15, 5))
plt.title('Categorical Accuracy - Epoch Graphics')
plt.xlabel('Epoch')
plt.ylabel('Categorical Accuracy')
plt.plot(range(1, len(hist.history['categorical_accuracy']) + 1),
hist.history['categorical_accuracy'])
plt.plot(range(1, len(hist.history['val_categorical_accuracy']) + 1),
hist.history['val_categorical_accuracy'])
plt.legend(['Categorical Accuracy', 'Validation Categorical Accuracy'])
plt.show()

eval_result = model.evaluate(test_dataset_x, test_dataset_y)
for i in range(len(eval_result)):
    print(f'{model.metrics_names[i]} ---> {eval_result[i]}')

import numpy as np
import glob

for path in glob.glob('*.*'):
    img_data = plt.imread(path)
    scaled_img_data = img_data / 255
    result = model.predict(scaled_img_data.reshape(1, 32, 32, 3))
    number = np.argmax(result)
    plt.title(class_names[number], fontsize=16)
    plt.imshow(img_data)
    plt.show()

```

CIFAR-100 veri kümesi de veri dosyalarından hareketle indirilip yüklenebilir. Aşağıdaki adresten CIFAR-100 verilerini indirebilirisiniz:

<https://www.kaggle.com/fedesoriano/cifar100>

Buradan "train" ve "test" isimli iki dosya elde edeceksiniz. Bu dosyalar CSV formatında değildir. Bu dosyalar Python standart kütüphanesindeki pickle modülü ile seri hale getirilmiş sözlük nesnesi içermektedir. Dolayısıyla bu dosyaları aynı modülü kullanarak açmalısınız:

```

import pickle

with open('train', 'rb') as f:
    training_dataset_dict = pickle.load(f, encoding='bytes')

with open('test', 'rb') as f:
    test_dataset_dict = pickle.load(f, encoding='bytes')

```

Deserialize işleminden bir sözlük nesnesi elde edilmiştir. Bu sözlük nesnesinin anahtarları şöyledir:

Burada her anahtara karşılık değer olarak ilgili veriler bulunmaktadır. Resimlerdeki pixel'ler de yine CIFAR-10 örneğinde olduğu gibi 3x32x32 biçimindedir. Dolayısıyla bunların da yine 32x32x3 biçimine dönüştürülmesi gereklidir:

```
import numpy as np

training_dataset_x = training_dataset_dict[b'data']
training_dataset_y = np.array(training_dataset_dict[b'fine_labels'], dtype=np.uint32)

training_dataset_x = training_dataset_x.reshape(-1, 3, 32, 32)
training_dataset_x = np.transpose(training_dataset_x, (0, 2, 3, 1))

test_dataset_x = test_dataset_dict[b'data']
test_dataset_y = np.array(test_dataset_dict[b'fine_labels'], dtype=np.uint32)

test_dataset_x = test_dataset_x.reshape(-1, 3, 32, 32)
test_dataset_x = np.transpose(test_dataset_x, (0, 2, 3, 1))
```

Kodun geri kalan kısmı yukarıdaki örneğin aynısıdır.

Keras'ta EarlyStopping ve ModelCheckpoint Callback Sınıfları

Keras'taki bazı callback sınıflarını daha önce incelemiştik. Burada daha önce ele almadığımız iki callback sınıfını inceleyeceğiz. EarlyStopping eğitimi durdurmak için kullanılan bir callback sınıfıdır. Yani eğitim sırasında epoch'lardaki sinama (validation) değerleri belli koşulları sağladığında (örneğin artık daha fazla iyileşmediği ya da gerilediği zaman) eğitimi durdurabiliyoruz. Çünkü overfit durumlarında eğitimin devam ettirilmesi toplamda daha kötü sonuçlara yol açabilmektedir. Şüphesiz programcı önce yüksek bir epoch'la denemeyi yapıp sonra grafiklere bakarak uygun epoch miktarını belirleyebilmektedir. (Biz de şimdide kadar hep böyle yaptık.) İşte EarlyStopping callback sınıfı bu işlemi daha pratik hale getirmek amacıyla düşünülmüştür. EarlyStopping sınıfının `__init__` metodunun parametrik yapısı şöyledir:

```
tensorflow.keras.callbacks.EarlyStopping(
    monitor='val_loss',
    min_delta=0,
    patience=0,
    verbose=0,
    mode='auto',
    baseline=None,
    restore_best_weights=False
)
```

EarlyStopping callback sınıfı her zaman epoch'lardan sonra devreye girmektedir. Metodun birinci parametresi hangi metrik değerin izleneceğini belirtmektedir. Bu parametrenin default değerinin 'val_loss' biçiminde olduğuna dikkat ediniz. Yani default durumda "validation loss" değerine bakılarak erken sonlandırma yapılacaktır. Biz bu parametreye kullandığımız metrik isimlerini girebiliriz. Sinama verileri için bu metrik isimlerinin başına "val_" önekinin getirilmesi gereklidir. patience parametresine epoch sayısı girilir. Eğitim burada girilen epoch sayısı kadar epoch işleminde ilgili değerde olumlu bir değişiklik yoksa sonlandırılır. Örneğin bu parametrenin 5 girildiğini varsayıyalım. Artık 5 epoch'ta bir farklılık oluşmuyorsa işlem sonlandırılabilir. patience parametresindeki duyarlılık `min_delta` parametresiyle ayarlanmaktadır. `min_delta` parametresinin default değeri 0'dır. Örneğin biz `min_delta` parametresi için 0.2 değeri girmış olalım. Bu durumda patience değeri 5 ise, 5 epoch süresince validation loss değerinde 0.2'den daha yüksek bir iyileşme olmamışsa eğitim sonlandırılacaktır. Metodun mode parametresi üç değer alabilir: "min", "max" ve "auto". Bu parametre monitor parametresine göre belirlenmelidir. Örneğin monitor parametresi "val_loss" ise biz bu değerin düşmesine yönelik bir sonlandırma yapmak isteriz. Bu durumda mode parametresi 'min' girilebilir. Ancak örneğin monitor parametresi "val_accuracy" ise bu durumda biz bu değerin yükselmesi yönünde bir ilgi içerisinde oluruz. O halde mode parametresinin bu durumda max girilmesi gereklidir. "auto" ise monitor parametresinin durumuna göre bunu otomatik yapmaktadır. Bu parametrenin default değerinin "auto" olduğunu görüyorsunuz. Fonksiyonun baseline parametresi ise sonlandırmaın monitor edilen özellik belli bir değere eriştiğinde yapılmasını sağlamak için kullanılmaktadır. `restore_best_weights` parametresi True girilirse (default değer False biçimdedir) bu durumda model sonlandırılmadan önce ağıın ağırlık değerleri o zamana kadarki en iyi monitor değeri ile set edilmektedir.

Örneğin en son üzerinde çalıştığımız CIFAR-100 veri kümesi için aşağıdaki gibi bir EarlyStopping callback oluşturmuş olalım:

```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

from tensorflow.keras.callbacks import EarlyStopping
esc = EarlyStopping(monitor='val_acc', patience=2, verbose=1, restore_best_weights=True)

history = model.fit(training_set_x, training_set_y, epochs=200, batch_size=512,
validation_split=0.2, callbacks=[esc])
```

Burada patience değerinin 5 olarak girildiğine dikkat ediniz. İşte model gitgide iyileştirilirken 3 epoch süresince elde edilen değer iyileştirilmeme işlem sonlandırılacaktır. Bu örnekte eğitimin çıktısı aşağıdaki gibi elde edilmiştir:

```
Epoch 1/20
625/625 [=====] - 51s 79ms/step - loss: 3.9653 - categorical_accuracy: 0.0923 - val_loss: 3.4964 - val_categorical_accuracy: 0.1603
Epoch 2/20
625/625 [=====] - 50s 80ms/step - loss: 3.2196 - categorical_accuracy: 0.2155 - val_loss: 3.0754 - val_categorical_accuracy: 0.2438
Epoch 3/20
625/625 [=====] - 51s 82ms/step - loss: 2.8290 - categorical_accuracy: 0.2871 - val_loss: 2.8634 - val_categorical_accuracy: 0.2870
Epoch 4/20
625/625 [=====] - 46s 74ms/step - loss: 2.5657 - categorical_accuracy: 0.3434 - val_loss: 2.7120 - val_categorical_accuracy: 0.3177
Epoch 5/20
625/625 [=====] - 29s 46ms/step - loss: 2.3601 - categorical_accuracy: 0.3830 - val_loss: 2.5911 - val_categorical_accuracy: 0.3433
Epoch 6/20
625/625 [=====] - 44s 71ms/step - loss: 2.1778 - categorical_accuracy: 0.4248 - val_loss: 2.5534 - val_categorical_accuracy: 0.3543
Epoch 7/20
625/625 [=====] - 35s 56ms/step - loss: 2.0178 - categorical_accuracy: 0.4629 - val_loss: 2.5052 - val_categorical_accuracy: 0.3718
Epoch 8/20
625/625 [=====] - 52s 82ms/step - loss: 1.8759 - categorical_accuracy: 0.4910 - val_loss: 2.6125 - val_categorical_accuracy: 0.3645
Epoch 9/20
625/625 [=====] - 43s 69ms/step - loss: 1.7417 - categorical_accuracy: 0.5210 - val_loss: 2.5440 - val_categorical_accuracy: 0.3772
Epoch 10/20
625/625 [=====] - 36s 58ms/step - loss: 1.6143 - categorical_accuracy: 0.5505 - val_loss: 2.6472 - val_categorical_accuracy: 0.3705
Epoch 11/20
625/625 [=====] - 38s 60ms/step - loss: 1.4937 - categorical_accuracy: 0.5800 - val_loss: 2.6714 - val_categorical_accuracy: 0.3842
Epoch 12/20
625/625 [=====] - 37s 60ms/step - loss: 1.3696 - categorical_accuracy: 0.6090 - val_loss: 2.7079 - val_categorical_accuracy: 0.3797
Epoch 13/20
625/625 [=====] - 38s 61ms/step - loss: 1.2629 - categorical_accuracy: 0.6347 - val_loss: 2.8481 - val_categorical_accuracy: 0.3788
Restoring model weights from the end of the best epoch.
Epoch 00013: early stopping
```

Burada 11'inci epoch'tan itibaren değerler bu epoch'taki değerleri 2 kez üst üste geçmemiştir. İşte restore_best_weights parametresi True geçildiği için artık model son halinin ağırlıklarıyla değil en iyi halinin ağırlıklarıyla yüklenecaktır.

ModelCheckpoint callback sınıfı modeli belli bir noktada saklamak için kullanılmaktadır. Sınıfın `__init__` metodu şöyledir:

```
tf.keras.callbacks.ModelCheckpoint(  
    filepath,  
    monitor='val_loss',  
    verbose=0,  
    save_best_only=False,  
    save_weights_only=False,  
    mode='auto',  
    save_freq='epoch',  
    options=None,  
    **kwargs  
)
```

Metodun birinci parametresi modelin saklanacağı dosyanın yol ifadesini belirtir. Bu yol ifadesi aşağıdaki örnekte olduğu gibi formatlı da girilebilmektedir. monitor parametresi yine izlenecek değeri belirtir. `save_weights_only` parametresi False girilirse (default durum) tüm model saklanır, True girilirse yalnızca model ağırlık değerleri saklanır. edilir. Eğer `save_best_only` parametresi True girilirse ancak daha iyi bir değer elde edildiğinde saklama işlemi yapılır. `period` parametresi (default değerinin 1 olduğuna dikkat ediniz) kaç epoch aralıklarla kontrollerin yapılacağını belirtir.

Fonksiyonun birinci parametresi formatsız girilirse saklama işlemi o dosyaya yapılır. Ancak birinci parametre formatlı girilirse bu durumda saklama işlemi formatta belirtilen farklı isimlerdeki dosyalara yapılır. Örneğin:

```
mcp = ModelCheckpoint('model.hdf5', save_best_only=False, verbose=1)
```

Burada her epoch'ta model "model.hdf5" dosyasında saklanır. Böylece dosyada son epoch'taki model bilgileri kalacaktır. Örneğin:

```
mcp = ModelCheckpoint('model.hdf5', save_best_only=True, verbose=1)
```

Burada ancak o epoch'a kadar en iyi olan monitor değeri "model.hdf5" dosyasında saklanır. Örneğin:

```
mcp = ModelCheckpoint('model.{epoch:02d}.hdf5', save_best_only=False, verbose=1)
```

Burada her epoch sonrasında model farklı bir dosya ismiyle saklanacaktır. Örneğin:

```
mcp = ModelCheckpoint('model.{epoch:02d}.hdf5', save_best_only=True, verbose=1)
```

Burada ancak o epoch'a kadar en iyi modeller farklı dosya isimleriyle saklanacaktır.

Metinsel Uygulamalar İçin Word Embedding Katmanları

Önceki konularda da belirttiğimiz gibi metne dayalı modellerde kestirim yapmak istendiğinde metinleri oluşturan öğeler girdi olarak sinir ağına verilmektedir. Genellikle bu girdiler sözcükler olur. Fakat bu amaçla karakterler de kullanılabilir. Metni oluşturan sözcüklerin nümerik girdilere dönüştürülmesi IMDB ve Reuters örneklerinde olduğu gibi sözcük indeksleme yoluyla yapılmaktadır. Yani tüm sözcükler (vocabulary) bir listede toplanır. Sonra metin içerisinde geçen sözcükler bu listedeki indeksler haline getirilir. Anımsayacağınız gibi biz IMDB ve Reuters örneklerinde metin içerisindeki tüm sözcükleri tek bir vektör olarak ele almış ve buradaki sözcük indekslerinden kurtulmak için binary encoding uygulamıştık. Böylece ağımızın girdi vektörü de tüm sözcüklerin (vocabulary) sayısı kadar uzunlukta olmuştur. Binary encoding işleminde metnin içerisindeki belli bir sözcük geçmişse biz o indeksli elemanı 1 yapıyorduk. Böylece girdi vektörümüz bazı elemanları 1 olan fakat pek çok elemanı 0 olan bir vektör biçimine dönüştürülmüş oluyordu.

Her sözcüğün one hot encoding biçiminde bir vektörle oluşturulmasının en önemli dezavantajlarından biri şüphesiz metin için oluşturulacak toplam girdi matrisinin çok büyük olmasıdır. Ayrıca bu yöntemin diğer bir dezavantajı da

sözcükler arasında semantik bir bağlantının oluşamamasıdır. Örneğin "güzel" ve "iyi" arasında bağlam bakımından bir yakınlık vardır. Halbuki model bu yakınlığı belirleyememektedir.

İşte "word embedding" denilen yöntemde metni oluşturan her bir sözcük büyük bir one hot encoding vektörü ile değil, küçük bir gerçek sayı vektörüyle ifade edilir. Örneğin metnimizdeki bir sözcük aşağıdaki gibi bir one hot encoding vektörüyle temsil edilmiş olsun:

```
[0, 0, ..., 0, 0, 1, 0, 0, ..., 0, 0, 0]
```

Bunun gerçek sayılarından oluşan karşılığı şöyle olabilir:

```
[0.78, 0.12, ..., 0.34, 0.67]
```

Metindeki her sözcüğün uzun bir one hot encoding vektörü ile değil de nispeten kısa bir gerçek sayı vektörüyle ifade edilmesinin şu avantajları vardır:

- 1) Bu sayede girdi kümesi azaltılmış olur. Yani uzun bir one hot encoding matrisinden kurtulmuş oluruz.
- 2) Bu sayede sözcükler arasında semantik bir bağlantı da kurulur. Çünkü bu dönüştürme işlemi aslında öğrenme modelinin bir parçasıdır ve söz konusu vektör ağ eğitildikçe daha iyi hale gelmektedir. Sonuçta iki sözcük arasındaki vektörel uzaklık ne kadar azsa o sözcüklerin eğitiminde kullanılan metinler içerisindeki anlamsal benzerlikler o kadar fazla olur.

Burada önemli bir noktayı vurgulamak istiyoruz. Word embedding sonucunda elde edilecek vektörler aslında modelin eğitilmesi sonucunda ağ tarafından oluşturulmaktadır. Yani word embedding işleminde sözcüklere karşı gelen vektör değerlerinin belirlenmesi aslında eğitim sırasında konumlandırılan birtakım eğitim parametreleri tarafından sağlanmaktadır. Peki "word embedding" katmanında vektör değerlerinin oluşturulması için hangi eğitim algoritmaları kullanılmaktadır? İşte word embedding eğitimi için kullanılan algoritmalar çeşitlidir. En popülerleri şunlardır:

Word2Vect (Google)
GloVe (Stanford)
fastText (Facebook)

Keras'taki Embedding katmanı Word2Vect temeline dayandırılmıştır. Biz burada bu algoritmalar üzerinde durmayacağız. Ancak bu katmanın Keras'taki kullanımı üzerinde duracağız.

Keras'ta Embedding katmanı Embedding isimli bir sınıfı temsil edilmiştir. Sınıfın `__init__` metodunun parametrik yapısı şöyledir:

```
tf.keras.layers.Embedding(  
    input_dim,  
    output_dim,  
    embeddings_initializer='uniform',  
    embeddings_regularizer=None,  
    activity_regularizer=None,  
    embeddings_constraint=None,  
    mask_zero=False,  
    input_length=None,  
    **kwargs  
)
```

Metodun `input_dim` isimli birinci parametresi sözcük indekslerinden oluşturulacak one hot encoding vektörünün uzunluğunu belirtmektedir. Yani bu parametre MNIST ve Reuters örnekleri için tüm sözcüklerin (vocabulary) sayısı olarak girilmelidir. Yukarıda da belirtildiği gibi Embedding katmanı için kullanılacak yazılarla ilişkin sözcük indekslerinden oluşan vektörler aşağıdaki görünümde olacaktır:

Bu sayılar metin içerisinde geçen sözcüklerin indeks numaraları olduğuna dikkat ediniz. (Tabii aslında Embedding katmanı kendi içerisinde bu vektördeki dizi elemanlarını "one hot encoding"e dönüştürerek işleme sokmaktadır. Girdinin kullanıcıdan bu biçimde istenmesi kullanıcıya kolaylık sağlamaktadır.) Ancak burada bir sorun vardır. Eğitimde kullanılacak indekslerden oluşan girdi vektörlerinin Keras tasarımdan kaynaklanan biçimde aynı uzunluk olması gereklidir. Oysa metinlerdeki sözcükler farklı sayıarda olabilmektedir. Pekiyi bu sorun nasıl çözülecektir? Kullanılacak en makul yöntem tüm metinlerin sözcük uzunluğunu aynı yapmaktadır. Bunun için tensorflow.keras.preprocessing modülü içerisindeki sequence.pad_sequences fonksiyonundan faydalanailebilir. Bu fonksiyon eğer dizi kısaysa onun başını sıfırlarla doldurur, eğer dizi uzunsa sonundan kırpmaya yapar. pad_sequences fonksiyonunun parametrik yapısı şöyledir:

```
tensorflow.keras.preprocessing.sequence.pad_sequences(
    sequences,
    maxlen=None,
    dtype='int32',
    padding='pre',
    truncating='pre',
    value=0.0
)
```

Foksiyonun birinci parametresi padding yapılacak iki boyutlu matrisi belirtmektedir. İkinci parametre padding sonrasında elde edilecek vektörün uzunluğunu belirtir. Fonksiyonun padding parametresi padding sırasında doldurma değerlerinin vektörün başa mı yoksa sonuna mı yapılacağını belirtmektedir. Bu parametrenin default değerinin 'pre' biçiminde olduğunu görüyorsunuz. Bu durumda padding işlemi baş tarafa yapılır. Padding işleminin vektörün sonuna yapılması isteniyorsa bu parametre 'post' biçiminde girilmelidir. Fonksiyonun value parametresi ise padding sırasında doldurulacak değeri belirtmektedir. Bu parametrenin default değerinin 0 olduğunu görüyorsunuz.

Pekiyi tüm metinler için aynı olacak bu uzunluk nasıl tespit edilecektir? Aslında belli bir uzunluk sezgisel olarak belirlenebilir. Ya da kırılma durumu oluşmasın isteniyorsa en uzun metnin uzunluğu referans alınabilir.

Metodun output_dim isimli ikinci parametresi çıktı vektörünün uzunluğunu belirtmektedir. Aslında Embedding katmanın çıktısı her biri bu parametrede belirtilen uzunlukta sütundan oluşan bir matristir. Örneğin biz Embedding nesnesini şöyle yaratmış olalım:

```
Embedding(10000, 32)
```

Buradaki input_dim parametresi için girilen 10000 değeri bizim tüm metinlerimizdeki toplam farklı sözcük sayısını (vocabulary) belirtmektedir. Burada output_dim parametresi için girilmiş olan 32 değeri ise her sözcüğün temsil edildiği vektörün uzunluğudur. Yani her sözcük one hot encoding yerine 32'lük bir gerçek sayı vektörye temsil edilmektedir. Bu parametredeki vektör uzunluğu için genellikle 16, 32, 64 gibi değerler tercih edilmektedir. Ancak metin içerisindeki sözcüklerin toplam sayısı çok fazlaysa bu değerin de büyütülmesi olumlu sonuçlar doğurmaktadır. Yukarıda Embedding katmanın girdilerinin yazılıdaki sözcüklerin indekslerinden oluştuğunu söylemiştık. Pekiyi Embedding sınıfı girdi olarak kullanılacak bu sözcük indekslerine ilişkin vektörlerin uzunluğunu nasıl anlamaktadır? İşte metodun input_length isimli parametresi bu uzunluğu belirlemekte kullanılabilmektedir. Ancak bu parametre girilmediye fonksiyon otomatik olarak bu belirlemeyi eğitim sırasında girilen diziye bakarak yapmaktadır. Örneğin:

```
Embedding(10000, 32, input_length=100)
```

Burada tüm metinlerdeki farklı sözcüklerin sayısı (vocabulary) 10000'dir. Metinlerin hepsi 100 sözcük uzunluğundadır ve her metin için 32 uzunlığında bir gerçek sayı vektörü oluşturulacaktır.

Embedding katmanın çıktısı her zaman "input_length X output_dim" biçiminde iki boyutlu bir matristir. Eğer programcı çıktıının tek boyutlu olmasını istiyorsa (örneğin Embedding katmanın çıktısını Dense katmana bağlamak isteyebilir) bu durumda Flatten katmanından faydalanailebilir. Aşağıdaki gibi bir metin söz konusu olsun:

```
"film çok güzeldi"
```

Bu metinde 3 sözcük vardır. Eğer metinlerdeki tüm yazıların 100 sözcükten oluştuğunu kabul edersek bu 3 sözcüklü metin 100 sözcüğe padding yapılacaktır. Ağın eğitilmesiyle bu yazı 32 elemanlı gerçek sayılarından oluşan bir vektöre dönüştürülecektir.

IMDB Örneğinde WordEmbedding Katmanının Kullanılması

Şimdi word embedding işlemini daha önce üzerinde çalışmış olduğumuz IMDB verileri üzerinde uygulayalım. Önce verileri Keras içerisindeki imdb modülünü kullanarak okuyalım:

```
from tensorflow.keras.datasets import imdb

VOCAB_SIZE = 30000
TEXT_SIZE = 300

(training_dataset_x, training_dataset_y), (test_dataset_x, test_dataset_y) =
imdb.load_data(num_words=VOCAB_SIZE)

word_dict = imdb.get_word_index()

from tensorflow.keras.preprocessing.sequence import pad_sequences

training_dataset_x = pad_sequences(training_dataset_x, maxlen=TEXT_SIZE, padding='post')
test_dataset_x = pad_sequences(test_dataset_x, maxlen=TEXT_SIZE, padding='post')
```

Şimdi modelimizi kuralım. Modelimizde önce bir WordEmbedding katmanı kullanacağız. Bu katmanın çıktısı matrisel olduğu için onu Flatten katmanı ile düzeltceğiz:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Flatten, Dense

model = Sequential(name='IMDB')
model.add(Embedding(VOCAB_SIZE, 64, input_length=TEXT_SIZE, name='Embedding'))
model.add(Flatten(name='Flatten'))
model.add(Dense(128, activation='relu', name='Dense-1'))
model.add(Dense(128, activation='relu', name='Dense-2'))
model.add(Dense(1, activation='sigmoid', name='Output'))
model.summary()
```

Şimdi modeli derleyip eğitelim. Eğitim işleminde üst üste 5 kez "val_loss" değeri iyileştirilemezse eğitimi sonlandıracagız ve modele zamana kadarki en iyi ağırlık değerlerini yükleyeceğiz:

```
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['binary_accuracy'])

from tensorflow.keras.callbacks import EarlyStopping

esc = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True, verbose=1)

hist = model.fit(training_dataset_x, training_dataset_y, epochs=20, batch_size=32,
validation_split=0.2, callbacks=[esc])
```

Epoch grafiğini çizdirelim:

```
import matplotlib.pyplot as plt

figure = plt.gcf()
figure.set_size_inches((15, 5))
plt.title('Loss - Epoch Graphics')
plt.xlabel('Epoch')
plt.ylabel('Loss')
```

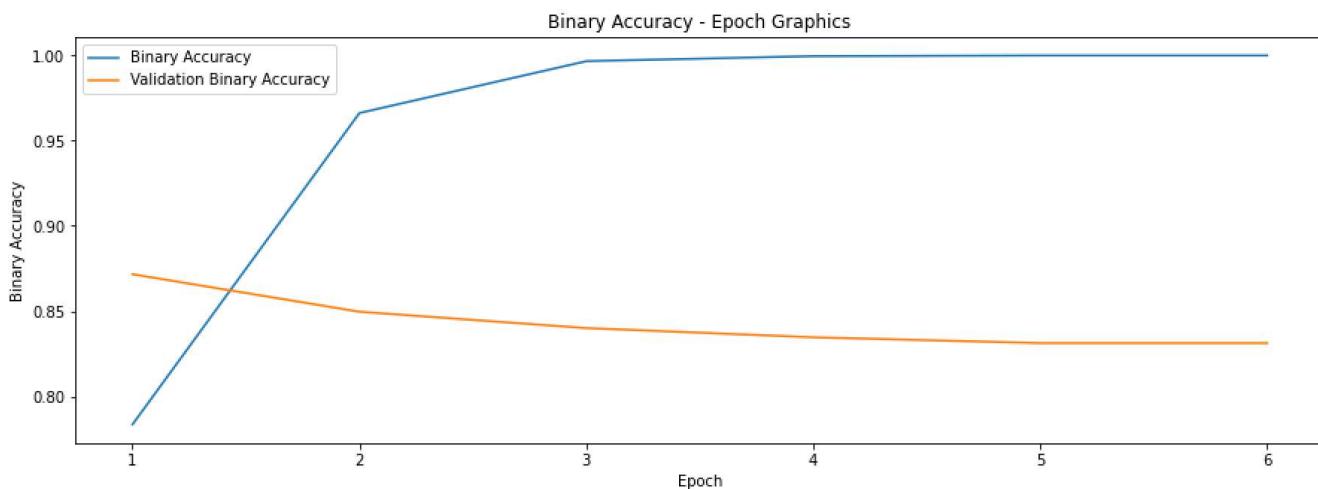
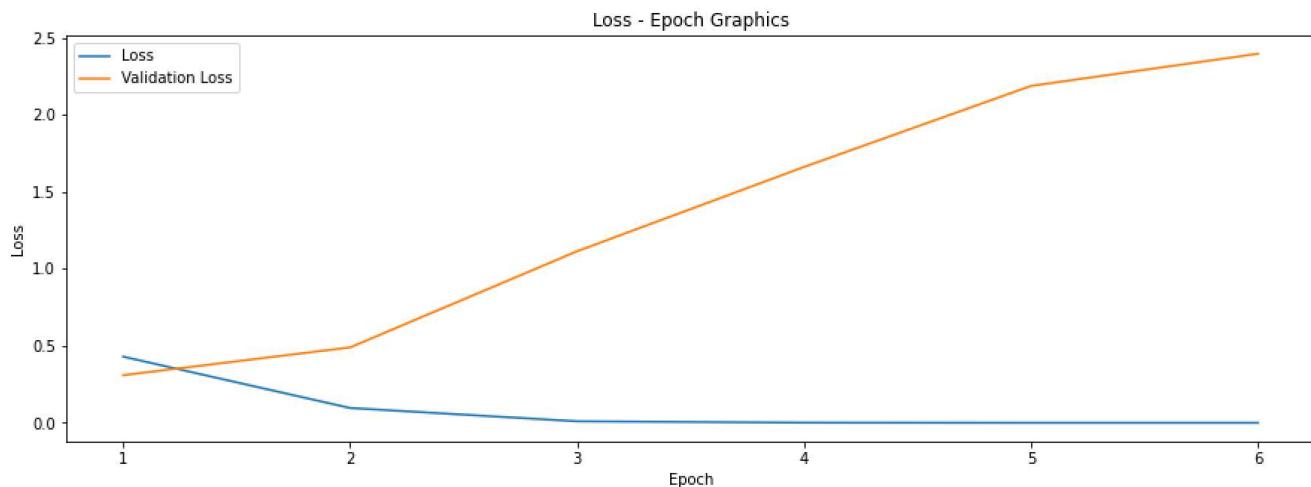
```

plt.plot(range(1, len(hist.history['loss']) + 1), hist.history['loss'])
plt.plot(range(1, len(hist.history['val_loss']) + 1), hist.history['val_loss'])
plt.legend(['Loss', 'Validation Loss'])
plt.show()

figure = plt.gcf()
figure.set_size_inches((15, 5))
plt.title('Binary Accuracy - Epoch Graphics')
plt.xlabel('Epoch')
plt.ylabel('Binary Accuracy')
plt.plot(range(1, len(hist.history['binary_accuracy']) + 1), hist.history['binary_accuracy'])
plt.plot(range(1, len(hist.history['val_binary_accuracy']) + 1),
hist.history['val_binary_accuracy'])
plt.legend(['Binary Accuracy', 'Validation Binary Accuracy'])
plt.show()

```

Şu grafikler elde edilmiştir:



Modelimiz test edelim:

```

eval_result = model.evaluate(test_dataset_x, test_dataset_y)
for i in range(len(eval_result)):
    print(f'{model.metrics_names[i]} --> {eval_result[i]}')

```

Şu sonuçlar elde edilmiştir:

```
loss --> 0.31072360277175903
```

```
binary_accuracy --> 0.8683199882507324
```

Word Embedding İşleminin Anlamı ve Görsel Bir Açıklama

Önceki bölümde word embedding işlemiyle birbirlerine semantik bakımdan yakın sözcüklerin birbirlerine yakın sayısal vektörlerle ifade edilebildiğini belirtmiştir. Bu bölümde bu durumu açıklayıp görsel bir örnek vereceğiz.

Analitik düzlemede (x_1, y_1) ve (x_2, y_2) noktaları arasındaki Öklit uzaklığının $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$ biçiminde hesaplandığını biliyorsunuz. Benzer biçimde N boyutlu uzayda da Öklit uzaklığı $\sqrt{\sum_{i=1}^n (p_i - q_i)^2}$ biçiminde hesaplanmaktadır. İşte word embedding eğitimi sonucunda semantik bakımdan benzer sözcüklerin arasındaki Öklit uzaklıklar daha az benzer olmayan sözcükler arasındaki Öklit uzaklıkları daha fazla olacaktır. Bu durumu IMDB örneği üzerinde iki boyutlu düzlemede basit bir örnekle gösterebiliriz. Örneğimizde her sözcük için düzlemede görüntüleme kolay olsun diye iki elemanlı bir word embedding vektörü üreteceğiz. Bunun için modeli şöyle oluşturabiliriz:

```
from tensorflow.keras.datasets import imdb

VOCAB_SIZE = 30000
TEXT_SIZE = 300

(training_dataset_x, training_dataset_y), (test_dataset_x, test_dataset_y) =
imdb.load_data(num_words=VOCAB_SIZE)

word_dict = imdb.get_word_index()

from tensorflow.keras.preprocessing.sequence import pad_sequences

training_dataset_x = pad_sequences(training_dataset_x, maxlen=TEXT_SIZE, padding='post')
test_dataset_x = pad_sequences(test_dataset_x, maxlen=TEXT_SIZE, padding='post')

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Flatten, Dense

model = Sequential(name='IMDB')
model.add(Embedding(VOCAB_SIZE, 2, input_length=TEXT_SIZE, name='Embedding'))
model.add(Flatten(name='Flatten'))
model.add(Dense(128, activation='relu', name='Dense-1'))
model.add(Dense(128, activation='relu', name='Dense-2'))
model.add(Dense(1, activation='sigmoid', name='Output'))
model.summary()

model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['binary_accuracy'])

from tensorflow.keras.callbacks import EarlyStopping

esc = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True, verbose=1)

hist = model.fit(training_dataset_x, training_dataset_y, epochs=20, batch_size=32,
validation_split=0.2, callbacks=[esc])
```

Burada daha önce yaptığımız gibi IMDB modelimizi eğittik. Şimdi bir yazımı ağımıza girdi yapıp WordEmbedding katmanındaki çıktıyı elde edelim:

```
import numpy as np
import re

predict_text = 'great brilliant terrible bad fantastic movie wonderful horrible'
predict_words = re.findall("[a-zA-Z0-9]+", predict_text.lower())
```

```
predict_numbers = np.array([word_dict[pw] + 3 for pw in predict_words]).reshape(1, -1)
predict_data = pad_sequences(predict_numbers, TEXT_SIZE, padding='post')
```

Burada yazıyı eğitimde kullandığımız biçimde bir sayı dizisine dönüştürdük. Şimdi girişe bu yazıyı uygulayıp WordEmbedding katmanındaki çıktıyı elde edelim:

```
from tensorflow.keras.backend import function
get_embedding_out = function(model.layers[0].input, model.layers[0].output)
result_data = get_embedding_out(predict_data)[0]
```

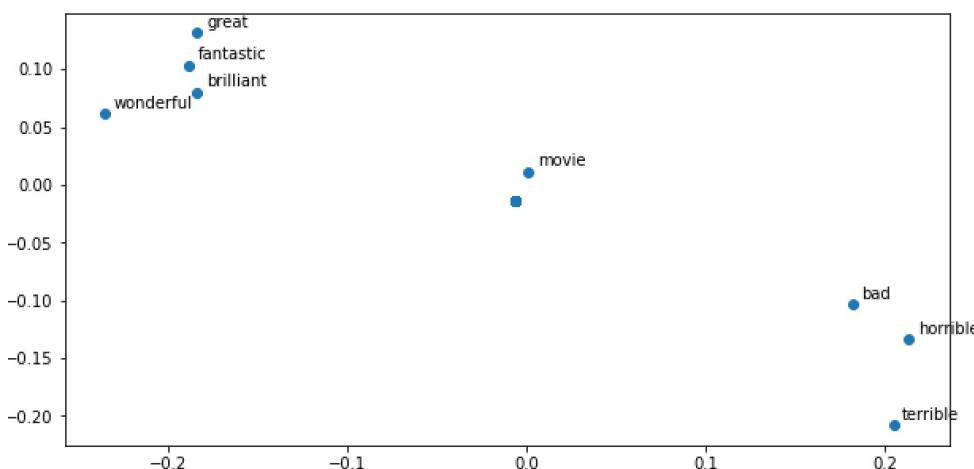
Burada result_data uyguladığımız girdiye karşılık elde ettiğimiz word embedding vektöründür. Nesnenin boyutuna dikkat ediniz:

```
In [140]: result_data.shape
Out[140]: (300, 2)
```

result_data 300 sözcüklük bir yazındaki her sözcüğün iki elemanlı word embedding vektörünü belirtmektedir. Bu matrisin ilk elemanları bizim yazında girdiğimiz sözcüklere ilişkin vektörler diğerleri de padding'ten dolayı sıfırlanmış değerleri ilişkin vektörlerdir. Şimdi sözcüklere karşı gelen noktaları düzleme gösterelim:

```
import matplotlib.pyplot as plt
figure = plt.gcf()
figure.set_size_inches((10, 5))
plt.scatter(result_data[:, 0], result_data[:, 1])
for i, txt in enumerate(predict_words):
    plt.annotate(txt, (result_data[i, 0] + 0.005, result_data[i, 1] + 0.005))
plt.show()
```

Şöyledir bir grafik elde edilmiştir:



Gördüğünüz gibi semantik olarak birbirlerine benzer sözcükler arasındaki Öklit uzaklıkları daha yakındır. Yukarıdaki kodu bir bütün olarak aşağıda veriyoruz:

```
from tensorflow.keras.datasets import imdb
VOCAB_SIZE = 30000
TEXT_SIZE = 300
(training_dataset_x, training_dataset_y), (test_dataset_x, test_dataset_y) =
```

```

imdb.load_data(num_words=VOCAB_SIZE)

word_dict = imdb.get_word_index()

from tensorflow.keras.preprocessing.sequence import pad_sequences

training_dataset_x = pad_sequences(training_dataset_x, maxlen=TEXT_SIZE, padding='post')
test_dataset_x = pad_sequences(test_dataset_x, maxlen=TEXT_SIZE, padding='post')

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Flatten, Dense

model = Sequential(name='IMDB')
model.add(Embedding(VOCAB_SIZE, 2, input_length=TEXT_SIZE, name='Embedding'))
model.add(Flatten(name='Flatten'))
model.add(Dense(128, activation='relu', name='Dense-1'))
model.add(Dense(128, activation='relu', name='Dense-2'))
model.add(Dense(1, activation='sigmoid', name='Output'))
model.summary()

model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['binary_accuracy'])

from tensorflow.keras.callbacks import EarlyStopping

esc = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True, verbose=1)

hist = model.fit(training_dataset_x, training_dataset_y, epochs=20, batch_size=32,
validation_split=0.2,
callbacks=[esc])

import numpy as np
import re

predict_text = 'great brilliant terrible bad fantastic movie wonderful horrible'

predict_words = re.findall("[a-zA-Z0-9]+", predict_text.lower())
predict_numbers = np.array([word_dict[pw] + 3 for pw in predict_words]).reshape(1, -1)
predict_data = pad_sequences(predict_numbers, TEXT_SIZE, padding='post')

from tensorflow.keras.backend import function

get_embedding_out = function(model.layers[0].input, model.layers[0].output)

result_data = get_embedding_out(predict_data)[0]

import matplotlib.pyplot as plt

figure = plt.gcf()
figure.set_size_inches((10, 5))
plt.scatter(result_data[:, 0], result_data[:, 1])

for i, txt in enumerate(predict_words):
    plt.annotate(txt, (result_data[i, 0] + 0.005, result_data[i, 1] + 0.005))
plt.show()

```

Metinsel Uygulamalarda Word Embedding İşlemleri İle Evrişim İşlemlerinin Birlikte Kullanılması

Metinsel uygulamalarda word embedding işlemleri ile evrişim işlemleri birlikte de kullanılabilir. Bu tür modellere girdi katmanı olarak önce bir word embedding katmanı eklenir. Word embedding katmanının çıktısı tek boyutlu evrişim katmanına, evrişim katmanın da çıktısı bir ya da birden fazla dense katmana bağlanmaktadır. Şimdi böyle bir modeli IMDB veri kümesine uygulayalım. Bunun için yine önce Keras içerisinde hazır olarak bulunan IMDB verilerini yükleyelim:

```

from tensorflow.keras.datasets import imdb

VOCAB_SIZE = 30000
TEXT_SIZE = 300

(training_dataset_x, training_dataset_y), (test_dataset_x, test_dataset_y) =
imdb.load_data(num_words=VOCAB_SIZE)

from tensorflow.keras.preprocessing.sequence import pad_sequences

training_dataset_x = pad_sequences(training_dataset_x, maxlen=TEXT_SIZE)
test_dataset_x = pad_sequences(test_dataset_x, maxlen=TEXT_SIZE)

```

Şimdi modelimizi kuralım:

```

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Conv1D, MaxPooling1D, Flatten, Dense

model = Sequential(name='IMDB-Convolution')
model.add(Embedding(MAX_FEATURES, 32, name='Embedding', input_length=MAX_TEXT_LEN))
model.add(Conv1D(32, 3, padding='same', activation='relu', name='Conv1D'))
model.add(MaxPooling1D(name='MaxPooling1D'))
model.add(Flatten(name='Flatten'))
model.add(Dense(128, activation='relu', name='Dense'))
model.add(Dense(1, activation='sigmoid', name='Output'))
model.summary()

```

Modelde önce bir Embedding katmanı kullanarak word embedding işlemini gerçekleştirdik. Daha sonra word embedding işlemi ile elde edilen matrise Conv1D katmanı ile tek boyutlu evrişim işlemi uyguladık. Modelimizdeki Embedding katmanınının çıktısı iki boyutlu olduğu için Conv1D katmanınının ve MaxPooling1D katmanlarının çıktılarında iki boyutlu olduğuna dikkat ediniz. MaxPooling1D katmanınının çıktısını Dense katmana bağlayabilmek için Flatten katmanın kullanımını gerektiğini anımsayınız. Modelimiz için elde edilen summary bilgileri de şöyledir:

Model: "IMDB-Convolution"

Layer (type)	Output Shape	Param #
<hr/>		
Embedding (Embedding)	(None, 300, 32)	960000
Conv1D (Conv1D)	(None, 300, 32)	3104
MaxPooling1D (MaxPooling1D)	(None, 150, 32)	0
Flatten (Flatten)	(None, 4800)	0
Dense (Dense)	(None, 128)	614528
Output (Dense)	(None, 1)	129
<hr/>		
Total params: 1,577,761		
Trainable params: 1,577,761		
Non-trainable params: 0		

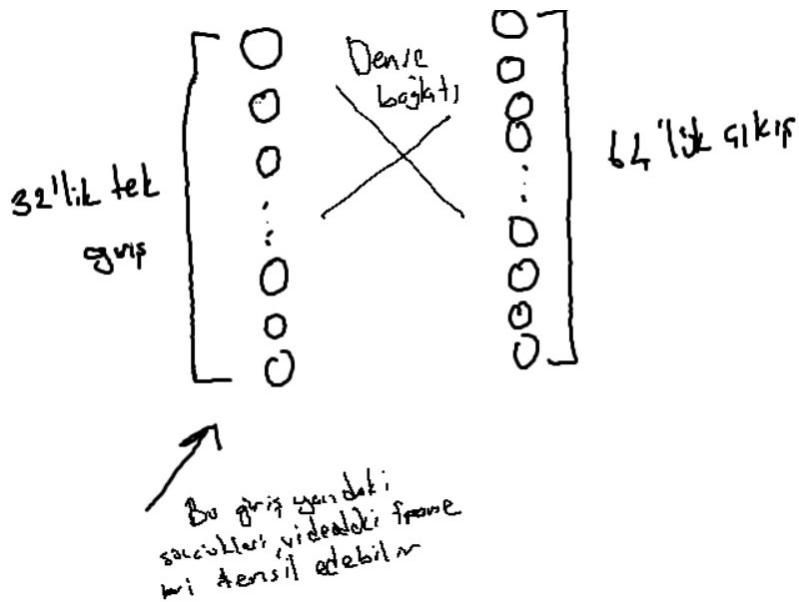
Buradaki değerlerin nasıl elde edildiği üzerinde durmak istiyoruz. Embedding katmanın girişinde 30000'lük bir one hot encoding vektörü vardır. Burada 32'lük çıktı için gereken nöron ağırlıklarının sayısı $30000 * 32 = 960000$ tanedir. Embedding katmanın çıkıştı 300X32'lük bir matris biçimindedir. Ancak buradaki her bir satır için ayrı bir filtre matrisi kullanılmamaktadır. Yani Conv1 katmanına giren bu 300X32'lük verilerin hepsi için aynı filtre kullanılmakadır.

Evrişim için seçilen filtre büyülüğu 3 olduğuna göre bu katmanda aslında matrisin her 32'lük elemanı için 3'lük bir filtreyi oluşturacak ağırlık değerleri tahmin edilmeye çalışılacaktır. Bu durumda Conv1D katmanındaki eğitilebilir parametrelerin sayısı $100 * 32$

Geri Beslemeli Sinir Ağları (Recurrent Neural Networks)

Şimdiye kadar görmüş olduğumuz ağ mimarilerinde bir nöronun çıktısı başka bir nörona girdi yapılıyordu. Bu tür ağ mimarilerine "ileriye doğru beslemeli (feed forward)" ağlar denilmektedir. İleriye doğru beslemeli ağlarda geçmişin kaydı ağıda tutulamamaktadır. Halbuki zamana dayalı uygulamalarda, doğal dil işleme uygulamalarında geçmişin de belli ölçülerde hatırlanması gerekmektedir. Örneğin IMDB veritabanında biz birtakım yorum yazılarından bu yorumların "iyi" mi "kötü" mü olduğu sonucunu çıkartmaya çalışıyorduk. Bu sonucun çıkartılması sözcüklerin önemli olduğu açıklır. Tasarladığımız ileriye doğru beslemeli ağ bu sözcüklerin önemini eğitim yoluyla öğrenmektedir. Ancak o örneğimizde sözcüklerin sırasının bir önemi yoktu. Halbuki yorum yazılarındaki sözcükler bir bağlam içerisinde gerçek anlamını bulabilmektedir. Yani o sözcüklerin kullandığı yerin geçmiş ve geleceği de o sözcüklerin anlamı üzerinde etkili olmaktadır. O halde böyle bir bağlamın sinir ağlarında oluşturulabilmesi için modelde bir biçimde geçmişin de etkili olması gereklidir. Aynı durum video görüntülerinde de söz konusudur. Videodaki görüntüler aslında frame'lerden (hareketli resmin anlık görüntüsü) oluşmaktadır. Fakat bu frame'ler birbirlerinden bağımsız değildir. Bir sonraki frame bir önceki frame'le ilgili olacak biçimde hareketli görüntüyü oluşturmaktadır. Başka bir deyişle görüntüde 10 saniye önce ne olduğu şimdi olanların anlamlanması için önemli olabilmektedir. Yani video frame'lerinden çıkartılacak sonuçlar anlık değil geçmişe de dayalıdır. Benzer biçimde finansal uygulamaların çoğu da belli bir geçmiş veri temelinde anlam kazanmaktadır. İşte "geri beslemeli ağ modeli geçmişin ve bağlamın da kestirimde dikkate alınmasını" sağlayan modeldir.

Şimdi biz yapay sinir ağı mimarisinde geri beslemeli bir ağ katmanın nasıl olması gerektiğini ele alalım. Geri beslemeli ağı yukarıda da açıkladığı gibi peşi sıra (zamansal) girdinin uygulandığı ağ modelidir. Bu birden fazla giriş muhtemelen bir sürecin içerisindeki elemanları oluşturmaktadır. Örneğin bir metindeki sözcükler, bir video görüntüsündeki frame'ler bu sıralı girişleri temsil ediyor olabilir. Örneğin 32'lük bir girdi katmanına (yani girişe) sahip olan çıktısı 64'lük olan bir katman düşünelim. Onun şekilsel gösterimi şöyle olacaktır:

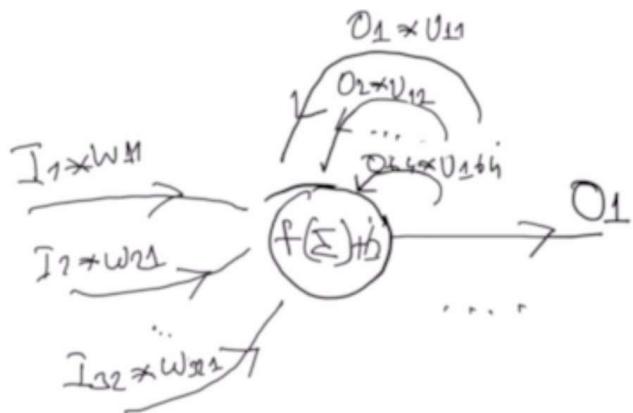


Geri beslemeli ağıda bir öğeye ilişkin girişler bir tane değildir. Bir grup giriş peşi sıra uygulanmaktadır. (Örneğin bizim daha önce yapmış olduğumuz IMDB ve Reuters örneklerinde biz tüm yorum yazısını tek bir girdi olarak ele almıştık. Halbuki aynı örneği geri beslemeli ağı ile yapacak olsak yazındaki her sözcük sıralı girdilerden birini temsil ediyor olacaktır.) Yukarıdaki çizimde girdi 32'lük olduğuna göre bu 32'lük giriş 64'lük bir dense katmana bağlanacağsa burada tahmin edilmesi gereken toplam W değerleri $32 * 64$ tane olacaktır. Çünkü ögenin sıralı girdileri için aynı W değerleri kullanılmaktadır. Geri beslemeli ağlarda bir girdi öğesinin parçalarının tek tek uygulanması meselenin bir yönünü oluşturmaktadır. Meselenin diğer yönü "geri besleme" işlemi ile ilgilidir. Peki bu ağlarda geri besleme nasıl yapılacaktır? Geri besleme çıktı nöron grubundaki her çıktı değerinin yeniden bu katmandaki girdilere geri

sokulmasıyla oluşturulur. Çıktı nöron grubundaki nöronların her bir tanesinin çıktısı başka ağırlık değerleriyle çarpılıp toplanarak (dot product işlemi) bir sonraki girdi ile beraber yine aynı katmana girdi yapılır. Girdinin belli bir öğesinin o andaki parçalı girdisi I_t olsun. Bir önceki parçalı girdiden elde edilen çıktıının da O_{t-1} olduğunu varsayıyalım. Bu durumda nöronun yeni çıkışı şöyle hesaplanır:

$$\text{nöronun yeni çıktı değeri} = \text{activation}(\text{np.dot}(I_t, W) + \text{np.dot}(O_{t-1}, U) + b)$$

Burada nörona uygulanan girdilerin hepsi oradaki ağırlık değerleriyle dot product işlemeye sokulmuş sonra da çıktı değerleri çıktı ağırlıklarıyla dot product işlemeye sokularak toplanmıştır. Bu toplama bias değeri eklenerek elde edilen toplam aktivasyon fonksiyonuna sokulmuştur. Yani başka bir deyişle nöronların çıktı değerleri sanki girdilerin bir parçası olmuş gibi işleme sokulmuştur. 32'lük parçalı girişin 64'lük bir dense katmana bağlanması durumundaki bir dense katman nörounundaki ilk nöronun bağlantıları şöyle olacaktır:



Buradaki bağlantı aşağıdaki gibi bir kod parçasıyla temsilebilir:

```
import numpy as np

TIMESPEC = 100          # ardışıl uygulanacak eleman sayısı
INPUT_FEATURES = 32      # girdi katmanındaki nöron sayısı
OUTPUT_FEATURES = 64      # çıktı katmanındaki nöron sayısı

def activation(x):
    return np.maximum(0, x)

inputs = np.random.random((TIMESPEC, INPUT_FEATURES))
W = np.random.random((OUTPUT_FEATURES, INPUT_FEATURES))
U = np.random.random((OUTPUT_FEATURES, OUTPUT_FEATURES))
b = np.random.random((OUTPUT_FEATURES,))

outputs = []

output_t = np.random.random((OUTPUT_FEATURES,))
for input_t in inputs:
    output_t = activation(np.dot(W, input_t) + np.dot(U, output_t) + b)
    outputs.append(output_t)

total_outputs = np.concatenate(outputs, axis=0)

print(total_outputs)
```

Bu kodda girdi kümesi 32, çıktı kümesi 64 nöronundan oluşmaktadır. Bir öğe toplam 100 ayrı parça biçiminde girişlere uygulanmaktadır. Yani buradaki örneği IMDB ya da Reuters'e benzetirsek yazının 100 sözcükten olduğunu, her sözcüğün toplam 32 sözcükten (vocabulary) bir tanesine ilişkin olduğunu (32'lük one hot encoding) söyleyebiliriz. Başka bir deyişle bu koddaki TIMESPEC peşi sıra oluşan girdilerin sayısını, INPUT_FEATURES ise bu girdilerin her birinin kaç seçenekten birine ilişkin olduğunu temsil etmektedir. Koddaki OUTPUT_FEATURES çıktı nöron grubundaki

nöron sayısını belirtmektedir. Kodumuzda işin başında W , U ve b vektörleri 0 ile 1 arasında rastgele sayılarından oluşanak biçimde alınmıştır. Buradaki W girdi değerlerinin çarpılacağı ağırlıkları, U ise çıktı değerlerinin yeniden girdi olarak işleme sokulurken çarpılacak ağırlıkları belirtmektedir. b ise çıktı nöron grubundaki nöronların bias değerleridir. Aktivasyon fonksiyonu olarak "relu" kullanılmıştır. Kodun döngü kısmına dikkat ediniz:

```
for input_t in inputs:  
    output_t = activation(np.dot(W, input_t) + np.dot(U, output_t) + b)  
    outputs.append(output_t)
```

Burada girdi vektörü W değerleriyle çarpılıp toplanıp bu toplama U değerlerinin çıktı vektörüyle çarpımı da eklenmiştir. Örnek kodumuzda tüm nöron çıktılarının bir liste içerisinde biriktirildiğini de görüyorsunuz. Çünkü bu katmanın çıktısının büyülüğu OUTPUT_FEATURES kadar olmayacağından. OUTPUT_FEATURES * TIMESPEC (yani 64 * 100) kadar olacaktır. Bütün bu çıktı bir sonraki katmana girdi olarak verilecektir. Kodumuzda en sonunda np.concatenate fonksiyonu ile elde edilen liste bir ndarray nesnesine dönüştürülmüştür.

Aslında yukarıdaki kodun yaptığı katman işlemini zaten yapan SimpleRNN isimli hazır bir Keras katman sınıfı vardır. Sınıfın init metodunun parametrik yapısı şöyledir:

```
tf.keras.layers.SimpleRNN(  
    units,  
    activation='tanh',  
    use_bias=True,  
    kernel_initializer='glorot_uniform',  
    recurrent_initializer='orthogonal',  
    bias_initializer='zeros',  
    kernel_regularizer=None,  
    recurrent_regularizer=None,  
    bias_regularizer=None,  
    activity_regularizer=None,  
    kernel_constraint=None,  
    recurrent_constraint=None,  
    bias_constraint=None,  
    dropout=0.0,  
    recurrent_dropout=0.0,  
    return_sequences=False,  
    return_state=False,  
    go_backwards=False,  
    stateful=False,  
    unroll=False,  
    **kwargs  
)
```

Buradaki units değeri katmanın çıkışındaki nöron sayısını belirtmektedir. (Yani örneğimizdeki OUTPUT_FEATURES değerini belirtmektedir.) Tabii Sequential modelde Keras katmanları peşi sıra yiğildiği için bu katmanın girdisi aslında bir önceki katmanın çıktı değeridir. Dolayısıyla SimpleRNN katmanı ağıın ilk katmanı olamaz. (Yani bu katmandan önce bir girdi katmanının olması gereklidir.) Uygulama bağlamında bu katman genellikle "word embedding" içeren Embedding katman olur.

Şimdi bir SimpleRNN katmanını Keras'ta oluşturup modeli analiz edelim:

```
from tensorflow.keras.models import Sequential  
from tensorflow.keras.layers import Embedding, SimpleRNN, Dense  
  
model = Sequential(name='RNN')  
model.add(Embedding(100, 32, name='Embedding'))  
model.add(SimpleRNN(64, activation='relu', name='SimpleRNN'))  
model.add(Dense(1, activation='sigmoid', name='Output'))  
  
model.summary()
```

Burada Embedding katmanının çıktısı tek bir vektör değildir. Her biri 32'lik vektörlerden oluşan bir matristir. Bu vektör aslında SimpleRNN katmanın girdilerini oluşturmaktadır. Başka bir deyişle aslında Embedding katmanın çıktısı olan 32'lik vektörler SimpleRNN katmanını zamansal olarak beslemekte kullanılacak girdileri oluşturmaktadır. Örneğimizdeki SimpleRNN katmanın çıktısı ise 64 nöronundan oluşmaktadır. Modelin summary çıktısı aşağıdaki gibidir:

Model: "RNN"

Layer (type)	Output Shape	Param #
=====		
Embedding (Embedding)	(None, None, 32)	3200
SimpleRNN (SimpleRNN)	(None, 64)	6208
Output (Dense)	(None, 1)	65
=====		
Total params: 9,473		
Trainable params: 9,473		
Non-trainable params: 0		

Modeldeki eğitilebilir parametrelerin sayısının nasıl elde edildiğine bakalım: Embedding katmandaki 32'lik girişlerin 100'lük bir one hot encoding'ten elde edilmesi için toplamda bu katmanda $100 * 32 = 3200$ tane ağırlık değerinin bulunması gereklidir. SimpleRNN katmanın girdileri ise 32'lik vektörlerden oluşmaktadır. Fakat bu girişler aynı W değerleri ile çarpılacağından giriş için gereken W değerleri $32 * 64 = 2048$ tane olur. Öte yandan her çıkış ayrıca dense biçimde yeniden SimpleRNN katmanındaki nöronlara bağlanacağından dolayı buradaki U katsayılarının sayısı da $64 * 64 = 4096$ olacaktır. Tabii bir de bias değerleri vardır. Bunlar da SimpleRNN katmanındaki nöron sayısı kadardır (yani 64). O halde toplam eğitilebilir parametrelerin sayısı $32 * 64 + 64 * 64 + 64 = 6208$ olur. SimpleRNN katmanın çıktısı 64 nöronundan oluşacağına göre ve çıktı katmanında tek bir nöron bulunduğuna göre çıktı katmanında tahmin edilecek parametre sayısı $64 * 1 + 1 = 65$ olacaktır.

Keras'in SimpleRNN katmanın çıktısı en son parçalı girişten elde edilen tek boyutlu bir vektördür. Bu durum SimpleRNN katmanın çıktısına başka bir SimpleRNN katmanı bağlanırken soruna yol açabilmektedir. Örneğin yukarıdaki modele bir tane daha SimpleRNN katmanı ekleyemeye çalışalım:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, SimpleRNN, Dense

model = Sequential(name='RNN')
model.add(Embedding(100, 32, name='Embedding'))
model.add(SimpleRNN(64, activation='relu', name='SimpleRNN-1'))
model.add(SimpleRNN(64, activation='relu', name='SimpleRNN-2'))
model.add(Dense(1, activation='sigmoid', name='Output'))

model.summary()
```

Bu kod çalıştırılmak istendiğinde ikinci SimpleRNN katmanın eklenmesi işleminde exception oluşacaktır. Exception'ın nedeni ikinci SimpleRNN katmanın parçalı (zamansal) beslemede kullanılabilen matrisel bir girdi beklerken tek boyutlu vektörel bir bilgiyle karşılaşıyor olmasıdır. Çünkü SimpleRNN katmanı default durumda çıktı olarak zamansal girişler uygulandığındaki en son çıktının değerini vermektedir. (Halbuki bizim daha önce yazdığımız kod parçası bütün çıktıları biriktirip bir matris olarak veriyordu.) İşte SimpleRNN katmanın çıktısının biriktirilmiş çıktı değerlerinden oluşan bir matris olmasını sağlamak için SimpleRNN fonksiyonunun return_sequences isimli parametresi True geçilmelidir. Bu parametre True geçildiğinde artık ilk SimpleRNN katmanın çıktısı 64'lük vektörlerden oluşan bir matris olur. Böylece ikinci SimpleRNN katmanın girdisi için gereken zamansal matris ihtiyacı karşılanır. O halde yukarıdaki kodun düzeltilmiş biçimi şöyle olmalıdır:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, SimpleRNN, Dense
```

```

model = Sequential(name='RNN')
model.add(Embedding(100, 32, name='Embedding'))
model.add(SimpleRNN(64, name='SimpleRNN-1', return_sequences=True))
model.add(SimpleRNN(64, name='SimpleRNN-2'))
model.add(Dense(1, activation='sigmoid', name='Output'))

model.summary()

```

Buradaki summary çıktısı da şöyledir:

Model: "RNN"

Layer (type)	Output Shape	Param #
<hr/>		
Embedding (Embedding)	(None, None, 32)	3200
SimpleRNN-1 (SimpleRNN)	(None, None, 64)	6208
SimpleRNN-2 (SimpleRNN)	(None, 64)	8256
Output (Dense)	(None, 1)	65
<hr/>		
Total params: 17,729		
Trainable params: 17,729		
Non-trainable params: 0		

Pekiyi burada son SimpleRNN katmanındaki eğitilebilir parametrelerin sayısı nasıl hesaplanmıştır? İkinci SimpleRNN katmanının girdi katmanı aslında 64×64 tane W değeri oluşacaktır. U değerleri de yine 64×64 tane olacaktır. Tabii 64 tane de bias değeri bulunacaktır. O halde toplam eğitilebilir parametrelerin sayısı $64 \times 64 + 64 \times 64 + 64 = 8256$ olmaktadır. Yine ikinci SimpleRNN katmanının çıkışında 64 nöron olacağından modelin çıktı katmanındaki eğitilebilir parametrelerin sayısı $64 \times 1 + 1 = 65$ tane olacaktır.

Geri Beslemeli Ağ Uygulaması Olarak SimpleRNN Katmanlarına Sahip IMDB Örneği

Şimdi yukarıda gördüğümüz konuları IMDB örneğiyle bir araya getirelim. Anımsanacağı gibi IMDB örneğinde kişilerin yorum yazılarından film hakkında olumlu mu olumsuz mu düşündükleri kestirilmeye çalışılıyordu. Daha önce yaptığımız IMDB örneğinde sözcüklerin öncelik sonralık ilişkisine baktık. Şimdi geri beslemeli bir ağ ile sözcükleri bağlam içerisinde değerlendirmeye çalışalım. IMDB verilerini daha önce yaptığımız gibi tensorflow.keras.datasets.imdb modülünü kullanarak okuyabiliriz:

```

from tensorflow.keras.datasets import imdb

VOCAB_SIZE = 10000
TEXT_SIZE = 300

(training_dataset_x, training_dataset_y), (test_dataset_x, test_dataset_y) =
imdb.load_data(num_words=VOCAB_SIZE)

```

Burada zaten okunan training_dataset_x verileri yorumlardaki sözcük indekslerinden oluşan numpy dizisi biçimindedir. Örneğin:

```
In [2]: training_dataset_x
```

```
Out[2]:
```

```
array([list([1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 458, 4468, 66, 3941, 4, 173, 36, 256, 5, 25, 100, 43, 838, 112, 50, 670, 2, 9, 35, 480, 284, 5, 150, 4, 172, 112, 167, 2, 336, 385, 39, 4, 172, 4536, 1111, 17, 546, 38, 13, 447, 4, 192, 50, 16, 6, 147, 2025, 19, 14, 22, 4, 1920, 4613, 469, 4, 22, 71, 87, 12, 16, 43, 530, 38, 76, 15, 13, 1247, 4, 22, 17, 515, 17, 12, 16, 626, 18, 2, 5, 62, 386, 12, 8, 316, 8, 106, 5, 4, 2223, 5244, 16, 480, 66, 3785, 33, 4, 130, 12, 16, 38, 619, 5, 25, 124, 51, 36, 135, 48, 25, 1415, 33, 6, 22, 12, 215, 28, 77, 52, 5, 14, 407, 16, 82, 2, 8, 4, 107, 117, 5952, 15, 256, 4, 2, 7, 3766, 5, 723, 36, 71, 43, 530, 476, 26, 400, 317, 46, 7, 4, 2, 1029, 13, 104, 88, 4, 381, 15, 297, 98, 32, 2071, 56, 26, 141, 6, 194, 7486, 18, 4, 226, 22, 21, 134, 476, 26, 480, 5, 144, 30, 5535, 18, 51, 36, 28, 224, 92, 25, 104, 4, 226, 65, 16, 38, 1334, 88, 12, 16, 283, 5, 16, 4472, 113, 103, 32, 15, 16, 5345, 19, 178, 32]), list([1, 194, 1153, 194, 8255, 78, 228, 5, 6, 1463, 4369, 5012, 134, 26, 4, 715, 8, 118, 1634, 14, 394, 20, 13, 119, 954, 189, 102, 5, 207, 110, 3103, 21, 14, 69, 188, 8, 30, 23, 7, 4, 249, 126, 93, 4, 114, 9, 2300, 1523, 5, 647, 4, 116, 9, 35, 8163, 4, 229, 9, 340, 1322, 4, 118, 9, 4, 130, 4901, 19, 4, 1002, 5, 89, 29, 952, 46, 37, 4, 455, 9, 45, 43, 38, 1543, 1905, 398, 4, 1649, 26, 6853, 5, 163, 11, 3215, 2, 4, 1153, 9, 194, 775, 7, 8255, 2, 349, 2637, 148, 605, 2, 8003, 15, 123, 125, 68, 2, 6853, 15, 349, 165, 4362, 98, 5, 4, 228, 9, 43, 2, 1157, 15, 299, 120, 5, 120, 174, 11, 220, 175, 136, 50, 9, 4373, 228, 8255, 5, 2, 656, 245, 2350, 5, 4, 9837, 131, 152, 491, 18, 2, 32, 7464, 1212, 14, 9, 6, 371, 78, 22, 625, 64, 1382, 9, 8, 168, 145, 23, 4, 1690, 15, 16, 4, 1355, 5, 28, 6, 52, 154, 462, 33, 89, 78, 285, 16, 145, 95]), list([1, 14, 47, 8, 30, 31, 7, 4, 249, 108, 7, 4, 5974, 54, 61, 369, 13, 71, 149, 14, 22, 112, 4, 2401, 311, 12, 16, 3711, 33, 75, 43, 1829, 296, 4, 86, 320, 35, 534, 19, 263, 4821, 1301, 4, 1873, 33, 89, 78, 12, 66, 16, 4, 360, 7, 4, 58, 316, 334, 11, 4, 1716, 43, 645, 662, 8, 257, 85 1200 42 1228 2578 83 68 2912 15 36 165 1529 278 36 69 2 780 8 106 14 6905
```

Göründüğü gibi buradaki numpy dizisi Python listelerinden oluşmaktadır. Python listelerinde de yorumlardaki sözcüklerin indeks numaraları vardır. Şimdi yorumlardaki sözcük sayılarını eşit hale getirelim:

```
from tensorflow.keras.preprocessing.sequence import pad_sequences

training_dataset_x = pad_sequences(training_dataset_x, maxlen=TEXT_SIZE)
test_dataset_x = pad_sequences(test_dataset_x, maxlen=TEXT_SIZE)
```

Şimdi artık tüm yorumlar eşit sayıda sözcüklerden oluşan gibi bir durum elde edilmiştir. Artık modelimizi kurabiliriz:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, SimpleRNN, Dense

model = Sequential(name='IMDB')
model.add(Embedding(VOCAB_SIZE, 64, input_length=TEXT_SIZE, name='Embedding'))
model.add(SimpleRNN(32, activation='relu', return_sequences=True, name='SimpleRNN-1'))
model.add(SimpleRNN(32, activation='relu', name='SimpleRNN-2'))
model.add(Dense(1, activation='sigmoid', name='Output'))

model.summary()
```

Burada Embedding katmanının çıktısı 64'lük vektörlerden oluşan bir matristir. Bu matris SimpleRNN katmanına parçalı girdi yapılmıştır. İlk SimpleRNN katmanın çıktısı bu katmanda return_sequences=True yapıldığı için tek bir vektor değil 32'lük vektörlerden oluşan bir matris olacaktır. İkinci SimpleRNN katmanın çıktısı ise return_sequences=True yapılmadığı için tek bir vektördür (son vektor). Bu katman da çıktı katmanına dense biçimde bağlanmıştır. Bu tür modellerde nöron sayılarını artırırken dikkat etmelisiniz. Eğitim veri kümesi azken nöron sayılarını artırmak performansı yükseltmek yerine düşürebilmektedir. Modeldeki nöron sayılarının fazla tutulmadığına ve ek Dense katmanların bulundurulmadığına dikkat ediniz.

Modelden elde edilen summary bilgileri şöyledir:

Model: "IMDB"

Layer (type)	Output Shape	Param #
<hr/>		
Embedding (Embedding)	(None, None, 64)	1920000

SimpleRNN-1 (SimpleRNN)	(None, None, 32)	3104
SimpleRNN-2 (SimpleRNN)	(None, 32)	2080
Output (Dense)	(None, 1)	33
=====		
Total params: 1,925,217		
Trainable params: 1,925,217		
Non-trainable params: 0		

Şimdi modelimi derleyip eğitelim:

```
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['binary_accuracy'])

from tensorflow.keras.callbacks import EarlyStopping

esc = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True, verbose=1)
hist = model.fit(training_dataset_x, training_dataset_y, epochs=20, batch_size=32,
validation_split=0.2, callbacks=[esc])
```

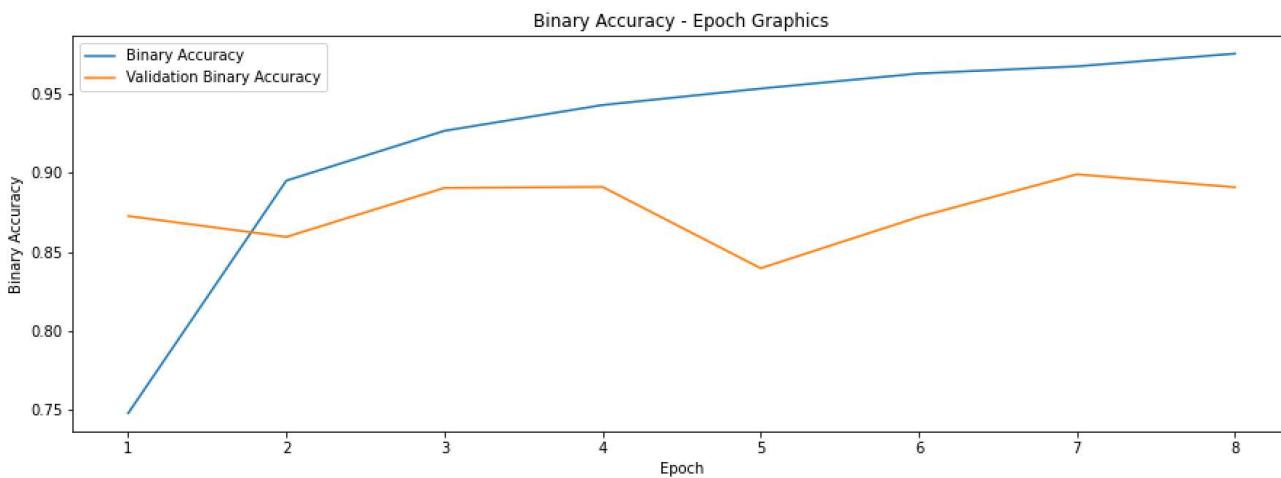
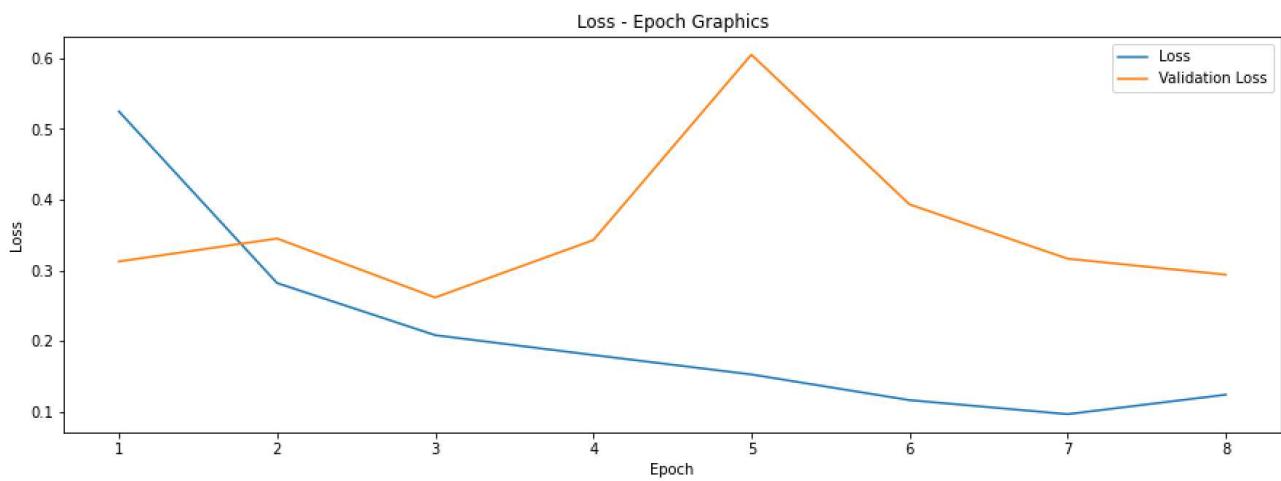
Burada "val_loss" değerinde üst üste 5 kere iyileşme olmazsa model sonlandırılmıştır ve en iyi ağırlık değerleri modele yüklenmiştir. Şimdi elde edilen modelin epoch grafiğini çizelim:

```
import matplotlib.pyplot as plt

figure = plt.gcf()
figure.set_size_inches((15, 5))
plt.title('Loss - Epoch Graphics')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.plot(range(1, len(hist.history['loss']) + 1), hist.history['loss'])
plt.plot(range(1, len(hist.history['val_loss']) + 1), hist.history['val_loss'])
plt.legend(['Loss', 'Validation Loss'])
plt.show()

figure = plt.gcf()
figure.set_size_inches((15, 5))
plt.title('Binary Accuracy - Epoch Graphics')
plt.xlabel('Epoch')
plt.ylabel('Binary Accuracy')
plt.plot(range(1, len(hist.history['binary_accuracy']) + 1), hist.history['binary_accuracy'])
plt.plot(range(1, len(hist.history['val_binary_accuracy']) + 1),
hist.history['val_binary_accuracy'])
plt.legend(['Binary Accuracy', 'Validation Binary Accuracy'])
plt.show()
```

EarlyStopping sonlandırmasına kadar elde edilmiş olan değerlerin grafiği şöyledir:



Şimdi de modelimizi test edelim:

```
eval_result = model.evaluate(test_dataset_x, test_dataset_y)
for i in range(len(eval_result)):
    print(f'{model.metrics_names[i]} --> {eval_result[i]}')
```

Şu sonuçlar elde edilmiştir:

```
loss --> 0.2805195450782776
binary_accuracy --> 0.8838000297546387
```

Yukardaki örneğin tüm kodları bir bütün olarak aşağıda veriyoruz:

```
from tensorflow.keras.datasets import imdb

VOCAB_SIZE = 30000
TEXT_SIZE = 300

(training_dataset_x, training_dataset_y), (test_dataset_x, test_dataset_y) =
imdb.load_data(num_words=VOCAB_SIZE)

from tensorflow.keras.preprocessing.sequence import pad_sequences

training_dataset_x = pad_sequences(training_dataset_x, maxlen=TEXT_SIZE)
test_dataset_x = pad_sequences(test_dataset_x, maxlen=TEXT_SIZE)

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, SimpleRNN, Dense
```

```

model = Sequential(name='IMDB')
model.add(Embedding(VOCAB_SIZE, 64, input_length=TEXT_SIZE, name='Embedding'))
model.add(SimpleRNN(32, activation='relu', return_sequences=True, name='SimpleRNN-1'))
model.add(SimpleRNN(32, activation='relu', name='SimpleRNN-2'))
model.add(Dense(1, activation='sigmoid', name='Output'))
model.summary()

model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['binary_accuracy'])

from tensorflow.keras.callbacks import EarlyStopping

esc = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True, verbose=1)

hist = model.fit(training_dataset_x, training_dataset_y, epochs=20, batch_size=32,
validation_split=0.2, callbacks=[esc])

import matplotlib.pyplot as plt

figure = plt.gcf()
figure.set_size_inches((15, 5))
plt.title('Loss - Epoch Graphics')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.plot(range(1, len(hist.history['loss']) + 1), hist.history['loss'])
plt.plot(range(1, len(hist.history['val_loss']) + 1), hist.history['val_loss'])
plt.legend(['Loss', 'Validation Loss'])
plt.show()

figure = plt.gcf()
figure.set_size_inches((15, 5))
plt.title('Binary Accuracy - Epoch Graphics')
plt.xlabel('Epoch')
plt.ylabel('Binary Accuracy')
plt.plot(range(1, len(hist.history['binary_accuracy']) + 1), hist.history['binary_accuracy'])
plt.plot(range(1, len(hist.history['val_binary_accuracy']) + 1),
hist.history['val_binary_accuracy'])
plt.legend(['Binary Accuracy', 'Validation Binary Accuracy'])
plt.show()

eval_result = model.evaluate(test_dataset_x, test_dataset_y)
for i in range(len(eval_result)):
    print(f'{model.metrics_names[i]} --> {eval_result[i]}')

```

Sinir Ağlarında Düzenleme (Regularization) İşlemleri

Önceki bölümlerde sinir ağlarının eğitiminde "overfitting" ve "underfitting" olgularını ele almıştık. Overfitting modelin istediğimiz şeyi değil başka şeyleri öğrenmesi underfitting ise modelin istediğimiz şeyleri arzu ettiğimiz düzeyde öğrenememesi anlamına geliyordu. Anımsayacağınız gibi modelin eğitimde gösterdiği başarıyı sınama ya da test sırasında gösteremiyor olması overfitting durumunun en önemli göstergesiydı. Yine ilgili bölümde overfitting durumunun başlıca nedenlerinin şunlar olabileceğini ifade etmiştik:

- Modelin karmaşık olması yani çok fazla nöron içermesi
- Eğitim verilerinin az olması
- Eğitim sırasında fazla epoch uygulanması

Makine öğrenmesinde overfitting durumunu engellemek için kullanılan yöntemlere düzenleme (regularization) işlemleri denilmektedir. En çok kullanılan düzenleme işlemlerinden iki tanesi "L1 / L2" düzenlemesi" ve "dropout" düzenlemesidir. Biz burada dropout düzenlemesi üzerinde duracağız.

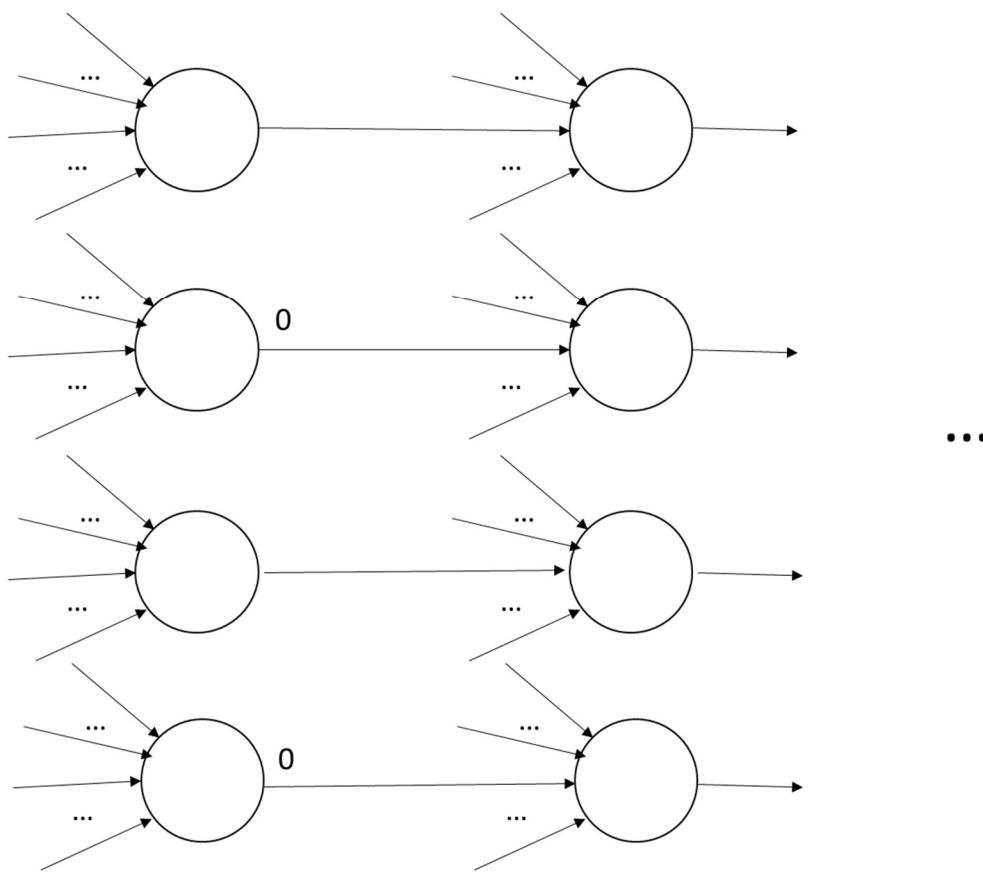
Dropout Düzenlemesi (Dropout Regularization)

Dropout düzenlemesi son zamanlarda en çok kullanılan düzenleme tekniklerinden biridir. Bu teknik ilk kez 2014 yılında Toronto Üniversitesi öğretim elemanları Srivastana, Hinton ve arkadaşları tarafından ortaya atılmıştır. Bu teknique göre sinir ağının saklı katmanlarındaki nöronların belli kısımları eğitim sırasında rastgele bir biçimde ağdan atılırsa bu durum overfitting durumuna direnç oluşturmaktadır. Teknik önce sezgisel olarak düşünülmüş ancak deneyisel olarak faydalı olduğu doğrulanmıştır. Dropout işlemi özellikle ağın saklı katmanları üzerinde uygulanmaktadır.

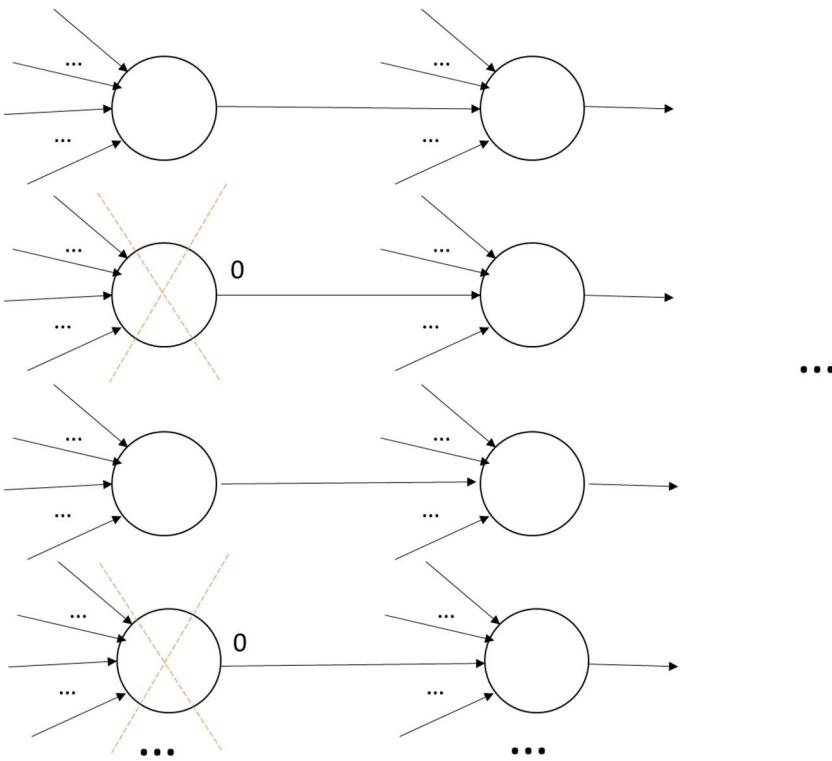
Aslında dropout işlemi sırasında uygulamada atılacak nöronlar gerçekten ağdan atılmamakta yalnızca onların çıktı değerleri bir sonraki katman için 0'a çekilmektedir. (Nöronun ağdan atılmasıyla çıktı değerinin 0'a çekilmesi arasında anlamsal bir farklılık olmadığına dikkat ediniz.) Pekiyyi dropout işlemi sırasında ağdan atılacak nöronlar nasıl belirlenmektedir? İşte bu işlem belli bir olasılık dahilinde rastgele bir biçimde yapılmaktadır. Örneğin nöron atılma olasılığı için 0.5 değerinin kullanıldığını varsayıyalım. Bu durumda katmandaki her nöronun atılma olasılığı 0.5'tir. (Yani örneğin bu durumda 0 ile 1 arasında rastgele bir sayı üretilebilir. Eğer bu sayı 0.5'ten düşükse nöron atılır, değilse nöron atılmaz. Benzer biçimde örneğin nöron atılma olasılığı 0.2 ise katmandaki her nöron için yine 0 ile 1 arasında rastgele bir sayı üretilebilir. Bu sayı 0.2'den küçükse nöron atılır, değilse nöron atılmaz. Uygulamada saklı katmanlar için genellikle 0.2 ile 0.5 arasında değerler girdi katmanı için ise 0.8 civarında değerler kullanılmaktadır.)

Dropout işlemi eğitim sırasında her batch işleminden sonra uygulanmaktadır. Yani her batch işlemi bittiğinde nöron ağırlık değerleri güncellendikten sonra yeniden ilgili katmanda belirlenen oranda nöron ağdan atılmaktadır. Böylece her batch işlemi sırasında ağdan farklı nöron grubu atılmaktadır. Geri beslemeli (recurrent) ağlarda bir nöronun ağdan atılması (yani çıkışının sıfır çekilmesi) yapılan geri beslemenin de iptal edilmesi gibi bir anlama gelmektedir.

Şimdi dropout işleminin nasıl yapıldığını şekilsel olarak göstermeye çalışalım. Aşağıdaki gibi bir ara katman söz konusu olsun:



Soldaki katmana dropout düzenlemesi uygulanacak olsun. Bu durumda her batch işleminde bu katmandaki rastgele nöronların çıkışları 0 yapılarak ağdan atılmış etkisi oluşturulmaktadır:



Dropout işleminden sonra genellikle katmanda sıfırlanmayan nöronlar belli değerlerle çarpılarak büyütülürler. Bu büyütme miktarları da genellikle dropout olasılığı ile orantılı biçimde yapılmaktadır. Örneğin katmandan atılma olasılığının 0.5 olduğunu varsayıyalım. Bu durumda katmandaki nöronların yarısı atılacaktır. İşte atılmayan nöronların çıktı değerleri de 2 ile çarpılarak büyütülebilmektedir.

Dropout işlemi yalnızca eğitim sırasında yapılan bir işlemidir. Ağ eğitildikten sonra test ya da kestirim sırasında dropout işlemi yapılmaz. Yani test ve kestirim işlemi ağdaki tüm nöronlar kullanılarak yapılmaktadır. Bir nöronun ağdan atılmasının aslında gerçek anlamda bir atılma anlamına gelmediğine nöron bilgilerinin korunup yalnızca çıkışının sıfır yapıldığına dikkat ediniz.

Keras'ta Dropout düzenlemesi için tensorflow.keras.layers modülünde Dropout isimli bir sınıf bulundurulmuştur. Dropout sınıfının `__init__` metodunun parametrik yapısı şöyledir:

```
tensorflow.keras.layers.Dropout(
    rate,
    noise_shape=None,
    seed=None,
    **kwargs
)
```

Metodun birinci parametresi katmandan atım için uygulanacak olasılığı belirtmektedir. Dropout katmanı yerleştirildiği yerin öncesindeki katmandaki nöronlar üzerinde etkili olmaktadır. Siz bu katmanı önceki katmanın çıkışlarının bazılarını 0 yapan bir katman olarak düşünebilirsiniz. Keras'in Dropout katmanı atılmayan nöronların çıktı değerlerini $1 / (1 - \text{rate})$ oranında yükseltmektedir. Örneğin:

```
import numpy as np
import tensorflow as tf

layer = tf.keras.layers.Dropout(0.5)
data = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10], dtype=np.float32)
result = layer(data, training=True)
print(data)
print(result)
```

Buradan şöyle bir çıktı elde edilmiştir:

```
[ 1.  2.  3.  4.  5.  6.  7.  8.  9. 10.]  
tf.Tensor([ 2.  4.  0.  0. 10. 12. 14. 16. 18.  0.], shape=(10,), dtype=float32)
```

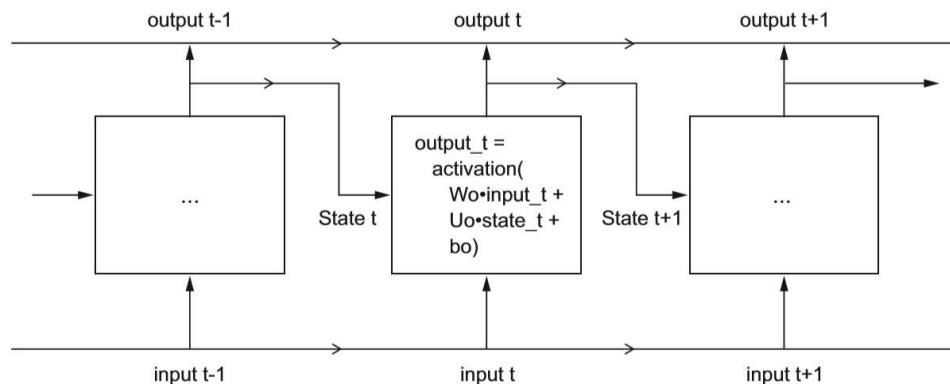
Göründüğü gibi burada her nöron için atılma olasılığı 0.5 olarak belirlenmiştir. Atılmayan nöronların çıktı değerleri de $1 / (1 - 0.5) = 2$ oranında yükseltilmiştir.

Geri Beslemeli Ağlar İçin LSTM Katmanı

Yukarıda kullanmış olduğumuz SimpleRNN katmanı aslında uygulamalarda pek tercih edilmemektedir. Çünkü bu katmanın bazı sorunları vardır. SimpleRNN katmanı her ne kadar geçmişe doğru bir hafıza oluşturuyorsa da bu hafıza "vanishing gradient problem" ismi verilen bir sorun yüzünden yakın geçmişe yönelik olarak kalmaktadır. Yani SimpleRNN katmanı o andaki bağlamın hemen öncesini yine bir biçimde hatırlayabilmekte geçmişi gittikçe daha zor hatırlayabilmektedir. Bu durum bağlamsal etkiyi azaltmaktadır. Bunun çözümü araştırcılar çeşitli yöntemler önermişlerdir. 90'lı yılların sonlarına doğru LSTM ve GRU katmanları bu sorunu çözmek için kullanılmıştır.

LSTM katmanını ele almaya başladan önce SimpleRNN katmanını anımsatmak istiyoruz. Aşağıda SimpleRNN katmanın parçalı (zamansal) verilerle beslenme biçimini görüyorsunuz:

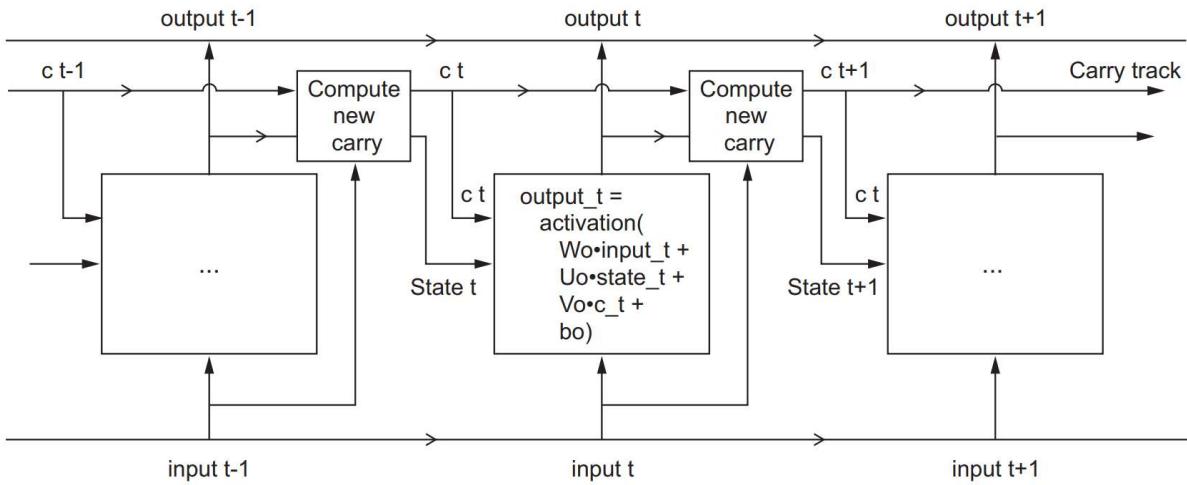
Bu yeni modeli SimpleRNN ile karşılaştırınız:



Alıntı Notu: Görsel "Deep Learning with Python (François Chollet)" isimli kitaptan alınmıştır.

Burada "input t" ile belirtilen veriler parçalı (zamansal) verilerdir. Şekilde her parça ayrı bir kutucukla gösterilmiştir. Göründüğünüz gibi önceki parçanın çıktısı sonraki parçayla birlikte yeniden girdi yapılmaktadır.

LSTM (Long Short Term Memory) katmanı ağa hem kısa süreli hem de uzun süreli bir hafıza oluşturmayı hedeflemektedir. LSTM SimpleRNN'nin mimari olarak biraz daha geliştirilmiş bir biçimidir. LSTM katmanında uzun dönem etkisi için yeni bir giriş daha ağa eklenmiş durumdadır. Mimarisinin temsili görüntüsü şöyledir:



Aıntı Notu: Görsel "Deep Learning with Python (François Chollet)" isimli kitaptan alınmıştır.

Bu mimaride SimpleRNN'den farklı olarak bir de c girişi uygulanmıştır. Bu durumda geri beslemeli ağın girdileri şunlar olmaktadır: Zamansal gerçek girdiler (yani metinsel uygulamalardaki sözcükler, şekildeki "input"lar), önceki çıktılar (şekildeki "state"ler) ve uzun dönem hafıza etkisi sağlayan "c" girişleri (şekildeki "c"ler). Pekiyi burada "c" vektörleri nasıl hesaplanmaktadır? Şekilden c vektörlerinin nöronun çıktısıyla, bir önceki c değeriyle ve girişle kombinere edilerek oluşturulduğunu görüyorsunuz. Bu oluşturulma işleminin matematiksel biçimini söyleyelim:

- Öncelikle buradaki parçalı girdinin çıktısı şöyle hesaplanmaktadır:

```
output_t = activation(np.dot(input_t, W) + np.dot(state_t, U) + np.dot(c_t, V) + b)
```

Buradaki V matrisi aslında 9 matrisin birleşimini temsil etmektedir. Eşitlikteki c_t girişleri şöyle hesaplanmaktadır:

```
c_t+1 = i_t * k_t + c_t * f_t
```

Yukarıda belirtilen c_t girişlerinin hesaplanmasıında kullanılan i_t , k_t ve f_t vektörleri ise şöyle hesaplanmaktadır:

```
i_t = activation(np.dot(state_t, U_i) + np.dot(input_t, W_i) + b_i)
f_t = activation(np.dot(state_t, U_f) + np.dot(input_t, W_f) + b_f)
k_t = activation(np.dot(state_t, U_k) + np.dot(input_t, W_k) + b_k)
```

Burada işin içine başka ağırlık değerlerinin de karıştığını görüyorsunuz: U_i , U_f , U_k , W_i , W_f , W_k , b_i , b_f ve b_k .

Şimdi LSTM katmanındaki eğitilebilir parametrelerin sayısını hesaplayalım. Bunun için önce iki büyülüye isim vereceğiz:

n : LSTM katmanına giren parçalı nöron sayısı olsun

m : LSTM katmanındaki nöron sayısı olsun

Bu durumda W matrisinin eleman sayısı $n * m$, U matrislerisinin eleman sayısı $m * m$ 'dir. U_i , U_f ve U_k matrislerinin eleman sayıları ise $m * m$, W_i , W_f ve W_k matrislerinin eleman sayıları ise $n * m$ tanedir. Bu katmandaki toplam bias değerlerinin sayısının da $b + b_i + b_f + b_k$ olduğuna dikkat ediniz. Bunların hepsi m tanedir. O halde LSTM katmanındaki eğitilebilir parametrelerin sayısı da $4 * (n * m + m * m + m)$ olacaktır.

Keras'ta LSTM katmanı doğrudan LSTM isimli sınıfla temsil edilmiştir. Bu katmanın kullanılması SimpleRNN katmanına benzemektedir.

```
tensorflow.keras.layers.LSTM(
    units,
    activation='tanh',
    recurrent_activation='sigmoid',
```

```

        use_bias=True,
        kernel_initializer='glorot_uniform',
        recurrent_initializer='orthogonal',
        bias_initializer='zeros',
        unit_forget_bias=True,
        kernel_regularizer=None,
        recurrent_regularizer=None,
        bias_regularizer=None,
        activity_regularizer=None,
        kernel_constraint=None,
        recurrent_constraint=None,
        bias_constraint=None,
        dropout=0.0,
        recurrent_dropout=0.0,
        implementation=2,
        return_sequences=False,
        return_state=False,
        go_backwards=False,
        stateful=False,
        unroll=False
)

```

Burada görüldüğü gibi tek zorunlu parametre birinci units parametresidir. Bu parametre çıktıının nöron sayısını belirtir. Yani bu anlamda LSTM katmanının basit kullanımı tamamen SimpleRNN katmanı gibidir. Burada önemli olan bir nokta LSTM katmanında aktivasyon fonksiyonunun "tanh" gibi bir şeyle sahip olmasıdır. LSTM katmanlarında relu aktivasyon fonksiyonunu kullanmayınız.

Pekiyi LSTM katmanı her zaman SimpleRNN'ye göre tercih edilmeli midir? Aslında LSTM uzun dönem hafıza etkisini sağladığına göre bunun bazı modellerde dezavantajı da olabilmektedir. Fakat genellikle bu uzun dönem etki istenir. Bu nedenle LSTM katmanı çoğu kez SimpleRNN'ye tercih edilmektedir. Şimdi LSTM katmanını IMDB veri kümesine uygulayalım. Önce yine Keras içerisinde hazır halde bulunan IMDB verilerini yükleyelim ve parçalı girdiler için düzenlemeyi yapalım:

```

VOCAB_SIZE = 30000
TEXT_SIZE = 300

from tensorflow.keras.datasets import imdb

(training_dataset_x, training_dataset_y), (test_dataset_x, test_dataset_y) =
imdb.load_data(num_words=VOCAB_SIZE)

word_dict = imdb.get_word_index()

from tensorflow.keras.preprocessing.sequence import pad_sequences

training_dataset_x = pad_sequences(training_dataset_x, TEXT_SIZE, padding='post')
test_dataset_x = pad_sequences(test_dataset_x, TEXT_SIZE, padding='post')

```

Şimdi modelimizi oluşturalım. Aslında modelimiz SimpleRNN örneğindekine benzer olacaktır. Ancak tabii geri besleme katmanı olarak SimpleRNN yerine LSTM kullanacağız:

```

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dropout, Dense

model = Sequential(name='IMDB-LSTM')
model.add(Embedding(VOCAB_SIZE, 64, input_length=TEXT_SIZE, name='Embedding'))
model.add(LSTM(64, activation='tanh', name='LSTM'))
model.add(Dropout(0.5, name='Dropout-1'))
model.add(Dense(128, activation='relu', name='Dense'))
model.add(Dropout(0.5, name='Dropout-2'))
model.add(Dense(1, activation='sigmoid', name='Output'))

```

```
model.summary()
```

Burada elde edilen modelin özet bilgileri şöyledir:

Model: "IMDB-LSTM"

Layer (type)	Output Shape	Param #
=====		
Embedding (Embedding)	(None, 300, 64)	1920000
LSTM (LSTM)	(None, 64)	33024
Dropout-1 (Dropout)	(None, 64)	0
Dense (Dense)	(None, 128)	8320
Dropout-2 (Dropout)	(None, 128)	0
Output (Dense)	(None, 1)	129
=====		
Total params:	1,961,473	
Trainable params:	1,961,473	
Non-trainable params:	0	

Embedding katmanınınındaki eğitilebilir parametrelerin sayısının $30000 * 64 = 1920000$ olduğunu görmüştük. Yukarıda LSTM katmanındaki eğitilebilir parametrelerin sayısının $4 * (n * m + m * m + m) = 33024$ olduğunu gördük. Değerleri yerine koyarsak $4 * (64 * 64 + 64 * 64 + 64) = 33024$ olur. Şimdi modeli derleyip eğitelim. Modelimizi yine loss değeri üst üste 5 kez iyileştirilmeden sonlandıracagız:

```
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['binary_accuracy'])

from tensorflow.keras.callbacks import EarlyStopping

esc = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True, verbose=1)
hist = model.fit(training_dataset_x, training_dataset_y, batch_size=32, epochs=20,
validation_split=0.2, callbacks=[esc])
```

Şimdi epoch grafiğini çizdirelim:

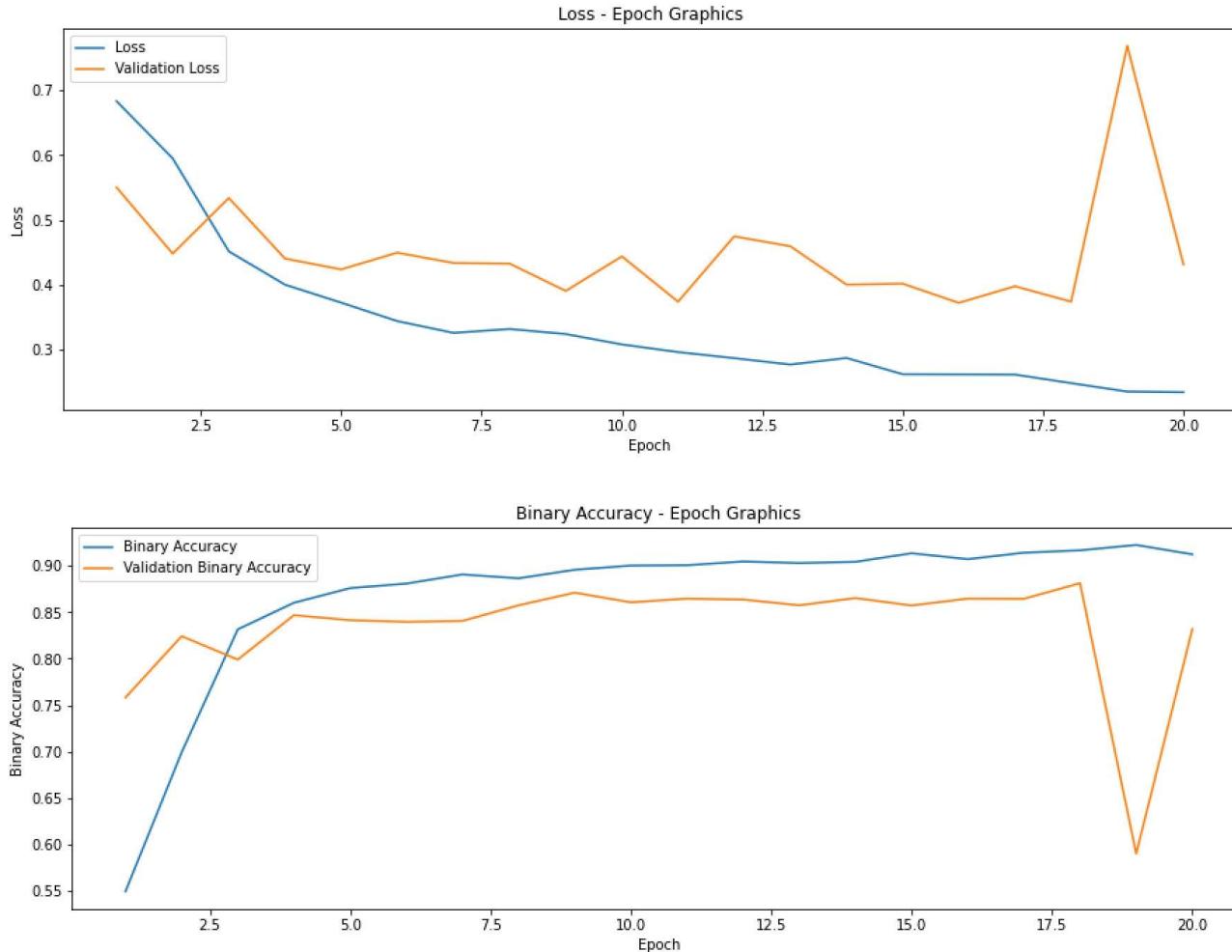
```
import matplotlib.pyplot as plt

figure = plt.gcf()
figure.set_size_inches((15, 5))
plt.title('Loss - Epoch Graphics')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.plot(range(1, len(hist.history['loss']) + 1), hist.history['loss'])
plt.plot(range(1, len(hist.history['val_loss']) + 1), hist.history['val_loss'])
plt.legend(['Loss', 'Validation Loss'])
plt.show()

figure = plt.gcf()
figure.set_size_inches((15, 5))
plt.title('Binary Accuracy - Epoch Graphics')
plt.xlabel('Epoch')
plt.ylabel('Binary Accuracy')
plt.plot(range(1, len(hist.history['binary_accuracy']) + 1), hist.history['binary_accuracy'])
plt.plot(range(1, len(hist.history['val_binary_accuracy']) + 1),
hist.history['val_binary_accuracy'])
plt.legend(['Binary Accuracy', 'Validation Binary Accuracy'])
plt.show()
```

```
eval_result = model.evaluate(test_dataset_x, test_dataset_y)
```

Buradan şu grafikler elde edilmiştir:



Şimdi test işlemimizi yapalım:

```
for i in range(len(eval_result)):  
    print(f'{model.metrics_names[i]} --> {eval_result[i]}')
```

Test işleminden şu değerler elde edilmiştir:

```
loss --> 0.4831244647502899  
binary_accuracy --> 0.809440016746521
```

Yukarıdaki kodu bir bütün olarak yeniden vermek itiyoruz:

```
VOCAB_SIZE = 30000  
TEXT_SIZE = 300  
  
from tensorflow.keras.datasets import imdb  
  
(training_dataset_x, training_dataset_y), (test_dataset_x, test_dataset_y) =  
imdb.load_data(num_words=VOCAB_SIZE)  
  
word_dict = imdb.get_word_index()  
  
from tensorflow.keras.preprocessing.sequence import pad_sequences  
  
training_dataset_x = pad_sequences(training_dataset_x, TEXT_SIZE, padding='post')
```

```

test_dataset_x = pad_sequences(test_dataset_x, TEXT_SIZE, padding='post')

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dropout, Dense

model = Sequential(name='IMDB-LSTM')
model.add(Embedding(VOCAB_SIZE, 64, input_length=TEXT_SIZE, name='Embedding'))
model.add(LSTM(64, activation='tanh', name='LSTM'))
model.add(Dropout(0.5, name='Dropout-1'))
model.add(Dense(128, activation='relu', name='Dense'))
model.add(Dropout(0.5, name='Dropout-2'))
model.add(Dense(1, activation='sigmoid', name='Output'))
model.summary()

model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['binary_accuracy'])

from tensorflow.keras.callbacks import EarlyStopping

esc = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True, verbose=1)
hist = model.fit(training_dataset_x, training_dataset_y, batch_size=32, epochs=20,
validation_split=0.2, callbacks=[esc])

import matplotlib.pyplot as plt

figure = plt.gcf()
figure.set_size_inches((15, 5))
plt.title('Loss - Epoch Graphics')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.plot(range(1, len(hist.history['loss']) + 1), hist.history['loss'])
plt.plot(range(1, len(hist.history['val_loss']) + 1), hist.history['val_loss'])
plt.legend(['Loss', 'Validation Loss'])
plt.show()

figure = plt.gcf()
figure.set_size_inches((15, 5))
plt.title('Binary Accuracy - Epoch Graphics')
plt.xlabel('Epoch')
plt.ylabel('Binary Accuracy')
plt.plot(range(1, len(hist.history['binary_accuracy']) + 1), hist.history['binary_accuracy'])
plt.plot(range(1, len(hist.history['val_binary_accuracy']) + 1),
hist.history['val_binary_accuracy'])
plt.legend(['Binary Accuracy', 'Validation Binary Accuracy'])
plt.show()

eval_result = model.evaluate(test_dataset_x, test_dataset_y)
for i in range(len(eval_result)):
    print(f'{model.metrics_names[i]} --> {eval_result[i]}')

```

Çift Yönlü LSTM Katmanı

LSTM katmanı biraz daha iyileştirilebilir mi? Normal olarak LSTM katmanı uzun dönem geçmişi de hatırlama konusunda katkı sağlamaktadır. Pekiyi bağlamsal olarak gelecek geçmiş ile ilişkili olabilir mi? Yani bizim gelecek verilerden elde ettiğimiz bilgiler geçmiş'i yorumlamada kullanılabılır mı? Bunun yanıtı bazı uygulamalar için "evet" olacaktır. Ancak deneyimler bunun her türlü geri beslemeli ağda mutlak anlamda bir fayda sağlamadığını da göstermiştir. Gelecek bilginin geçmişteki verilerin anlaşılması kolaylaştırması metinsel uygulamalarda kullanılabilir bir durumdur. Örneğin kişi yorumunda önce birisinin bir yere gittiğini söylüyor olabilir. Daha sonra bu yerin eczane olduğu anlaşılıyor olabilir. Bu durumda geçmişte o yerin eczane olduğu bilinse belki de metin daha iyi anlaşılacaktır. Bazı dillerin gramatik yapısı bu durumu daha belirgin hale getirmektedir.

Keras'ta çift yönlü LSTM katmanını oluşturabilmek için LSTM nesnesi Biderictional isimli sınıfı verilir. Yani katmanın tipik oluşturulma biçimini söylemek:

```
model.add(Bidirectional(LSTM(64, name='LSTM'), name='Bidirectional'))
```

Şimdi aynı modeli Bidirectional LSTM ile kurmaya çalışalım. Aslında kodda tek değiştireceğimiz yer LSTM katmanının çıktısını Bidirectional katmanına bağlamak olacaktır.

```
VOCAB_SIZE = 30000
TEXT_SIZE = 300

from tensorflow.keras.datasets import imdb

(training_dataset_x, training_dataset_y), (test_dataset_x, test_dataset_y) =
imdb.load_data(num_words=VOCAB_SIZE)

word_dict = imdb.get_word_index()

from tensorflow.keras.preprocessing.sequence import pad_sequences

training_dataset_x = pad_sequences(training_dataset_x, TEXT_SIZE, padding='post')
test_dataset_x = pad_sequences(test_dataset_x, TEXT_SIZE, padding='post')

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Bidirectional, Dropout, Dense

model = Sequential(name='IMDB-LSTM')
model.add(Embedding(VOCAB_SIZE, 64, input_length=TEXT_SIZE, name='Embedding'))
model.add(Bidirectional(LSTM(64, activation='tanh', name='LSTM'), name='Bidirectional'))
model.add(Dropout(0.5, name='Dropout-1'))
model.add(Dense(128, activation='relu', name='Dense'))
model.add(Dropout(0.5, name='Dropout-2'))
model.add(Dense(1, activation='sigmoid', name='Output'))
model.summary()

model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['binary_accuracy'])

from tensorflow.keras.callbacks import EarlyStopping

esc = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True, verbose=1)
hist = model.fit(training_dataset_x, training_dataset_y, batch_size=32, epochs=20,
validation_split=0.2, callbacks=[esc])

import matplotlib.pyplot as plt

figure = plt.gcf()
figure.set_size_inches((15, 5))
plt.title('Loss - Epoch Graphics')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.plot(range(1, len(hist.history['loss']) + 1), hist.history['loss'])
plt.plot(range(1, len(hist.history['val_loss']) + 1), hist.history['val_loss'])
plt.legend(['Loss', 'Validation Loss'])
plt.show()

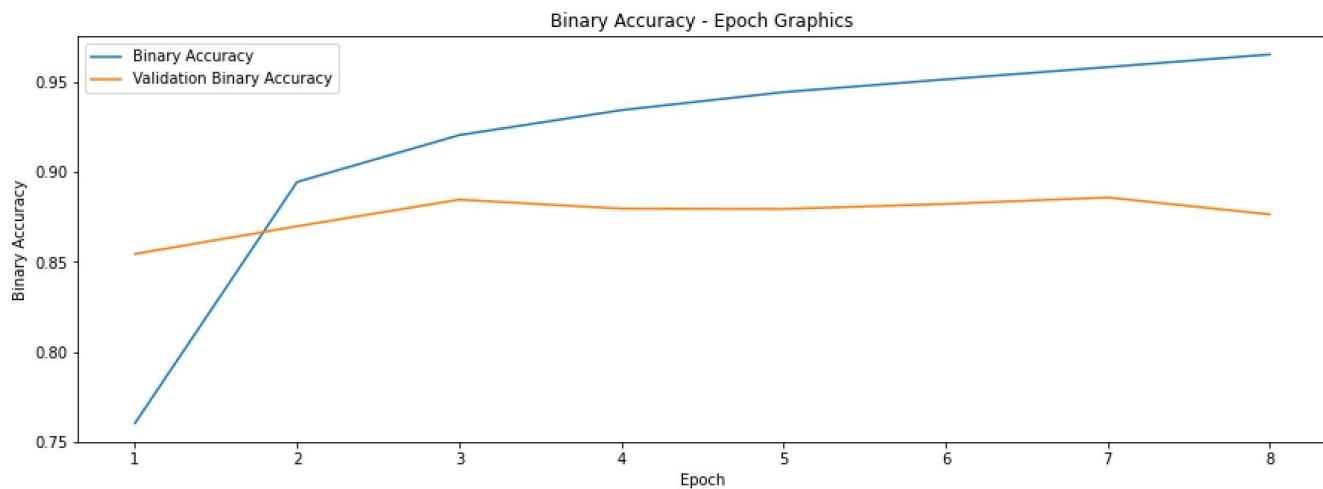
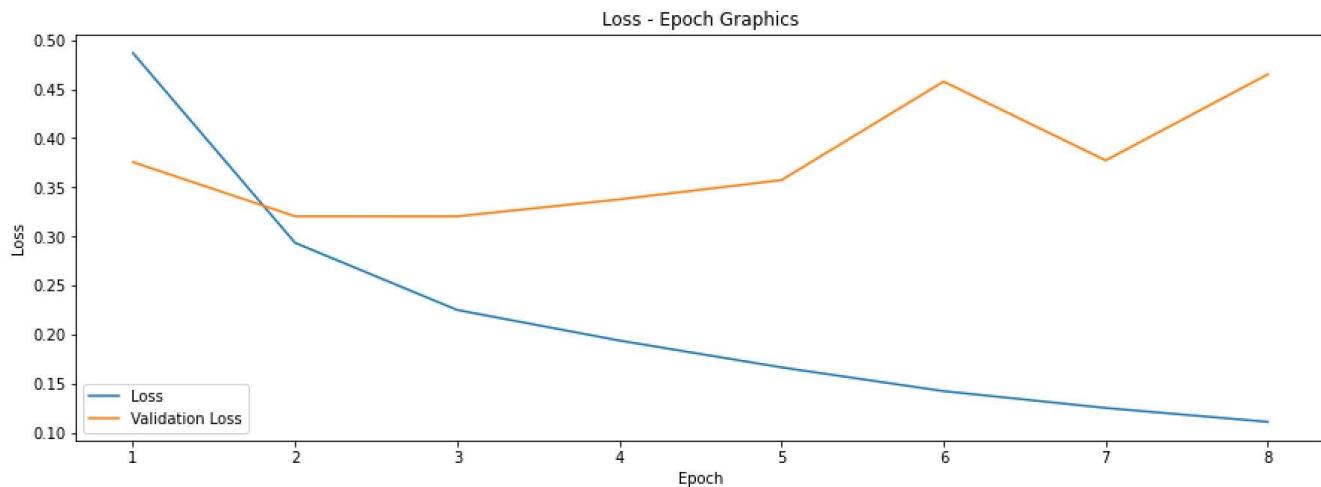
figure = plt.gcf()
figure.set_size_inches((15, 5))
plt.title('Binary Accuracy - Epoch Graphics')
plt.xlabel('Epoch')
plt.ylabel('Binary Accuracy')
plt.plot(range(1, len(hist.history['binary_accuracy']) + 1), hist.history['binary_accuracy'])
plt.plot(range(1, len(hist.history['val_binary_accuracy']) + 1),
hist.history['val_binary_accuracy'])
plt.legend(['Binary Accuracy', 'Validation Binary Accuracy'])
plt.show()
```

```

eval_result = model.evaluate(test_dataset_x, test_dataset_y)
for i in range(len(eval_result)):
    print(f'{model.metrics_names[i]} -> {eval_result[i]}')

```

Buradan elde edilen epoch grafikleri şöyledir:



Modelin testinden de şu sonuçlar elde edilmiştir:

```

loss --> 0.36292675137519836
binary_accuracy --> 0.8650000095367432

```

Göründüğü gibi burada değerler iyileşmiştir. Yani IMDBörneğinde çift yönlü geri beslemeli mimarının tek yönlü geri besleme mimarisine göre daha iyi sonuç verdiği görülmektedir.

Geri Beslemeli Ağlar İçin GRU Katmanı

GRU (Gated Recurrent Unit) yöntemi LSTM'ye bir alternatif oluşturmaktadır. GRU yöntemi de tipki LSTM yönteminde olduğu gibi ağa uzun dönem hafıza kazandırmaya çalışmaktadır. GRU yöntemi LSTM'ye göre daha az karmaşık olduğu için toplamdan LSTM'ye göre daha az eğitilebilir parametreye sahip olmaktadır. Dolayısıyla GRU yönteminde LSTM'ye göre hem eğitim süresi daha kısalıdır hem de bu yöntemde daha az bellek kullanılmaktadır. Ancak genel olarak (fakat her zaman değil) GRU katmanı LSTM'ye göre daha düşük performans göstermektedir. Bu nedenle pek çok uygulamada ilk tercih edilecek yöntem LSTM'dir. Eğer donanımsal bir kısıt söz konusuya (örneğin düşük güçlü CPU ve az miktarda bellek) LSTM yerine GRU katmanı tercih edilebilir. Katmanın genel kullanımı LSTM'de olduğu gibidir. Tabii GRU katmanı için genel performans LSTM'den kötü olabiliyorsa da SimpleRNN'den oldukça iyidir. Şimdi yukarıda LSTM katmanı ile yaptığımız IMDBörneğini bu kez GRU katmanı ile yapalım:

```

VOCAB_SIZE = 30000
TEXT_SIZE = 300

from tensorflow.keras.datasets import imdb

(training_dataset_x, training_dataset_y), (test_dataset_x, test_dataset_y) =
imdb.load_data(num_words=VOCAB_SIZE)

word_dict = imdb.get_word_index()

from tensorflow.keras.preprocessing.sequence import pad_sequences

training_dataset_x = pad_sequences(training_dataset_x, TEXT_SIZE, padding='post')
test_dataset_x = pad_sequences(test_dataset_x, TEXT_SIZE, padding='post')

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, GRU, Dropout, Dense

model = Sequential(name='IMDB-LSTM')
model.add(Embedding(VOCAB_SIZE, 64, input_length=TEXT_SIZE, name='Embedding'))
model.add(GRU(64, activation='tanh', name='GRU'))
model.add(Dropout(0.5, name='Dropout-1'))
model.add(Dense(64, activation='relu', name='Dense'))
model.add(Dropout(0.5, name='Dropout-2'))
model.add(Dense(1, activation='sigmoid', name='Output'))

model.summary()

model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['binary_accuracy'])

from tensorflow.keras.callbacks import EarlyStopping

esc = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True, verbose=1)
hist = model.fit(training_dataset_x, training_dataset_y, batch_size=32, epochs=20,
validation_split=0.2, callbacks=[esc])

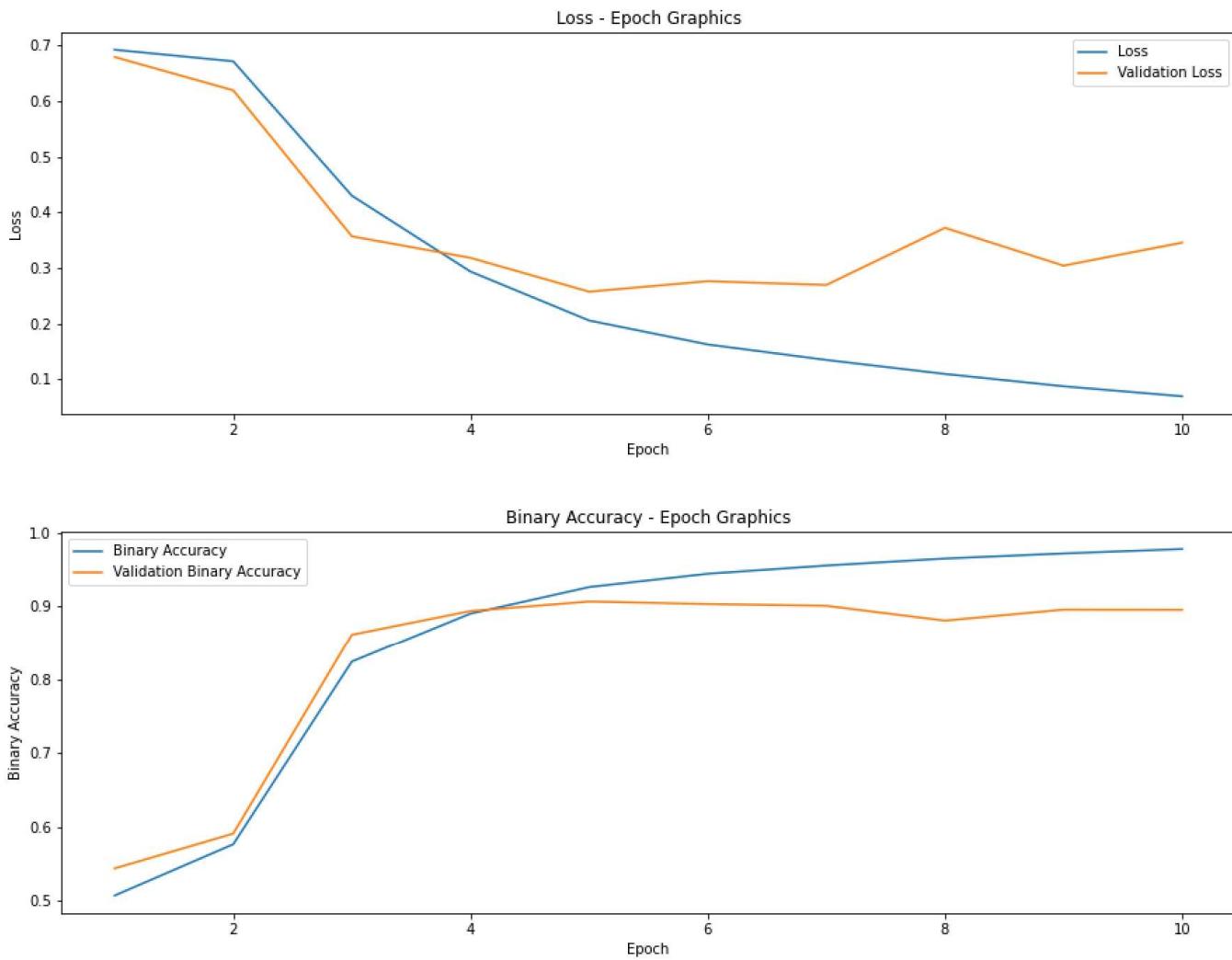
import matplotlib.pyplot as plt

figure = plt.gcf()
figure.set_size_inches((15, 5))
plt.title('Loss - Epoch Graphics')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.plot(range(1, len(hist.history['loss']) + 1), hist.history['loss'])
plt.plot(range(1, len(hist.history['val_loss']) + 1), hist.history['val_loss'])
plt.legend(['Loss', 'Validation Loss'])
plt.show()

figure = plt.gcf()
figure.set_size_inches((15, 5))
plt.title('Binary Accuracy - Epoch Graphics')
plt.xlabel('Epoch')
plt.ylabel('Binary Accuracy')
plt.plot(range(1, len(hist.history['binary_accuracy']) + 1), hist.history['binary_accuracy'])
plt.plot(range(1, len(hist.history['val_binary_accuracy']) + 1),
hist.history['val_binary_accuracy'])
plt.legend(['Binary Accuracy', 'Validation Binary Accuracy'])
plt.show()

```

Epoch grafikleri şöyledir:



Şimdi modeli test edelim:

```
eval_result = model.evaluate(test_dataset_x, test_dataset_y)
for i in range(len(eval_result)):
    print(f'{model.metrics_names[i]} --> {eval_result[i]}')
```

Şu sonuçlar elde edilmiştir:

```
loss --> 0.275068998336792
binary_accuracy --> 0.8964400291442871
```

Göründüğü gibi IMDB modeli için GRU katmanı LSTM ve İki yönlü LSTM'ye göre daha iyi bir performans göstermiştir. Ancak buradaki durumu genellemek yanlıştır.

Keras'taki Bidirectional katmanının bir dekoratör biçiminde oluşturulduğuna dikkat ediniz. Böylece biz herhangi bir geri beslemeli katmanı çift yönlü hale getirebiliriz. Örneğin GRU katmanı da aşağıdaki gibi çift yönlü hale getirilebilir:

```
training_dataset_x = pad_sequences(training_dataset_x, TEXT_SIZE, padding='post')
test_dataset_x = pad_sequences(test_dataset_x, TEXT_SIZE, padding='post')

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Bidirectional, GRU, Dropout, Dense

model = Sequential(name='IMDB-Bidirectional-GRU')
model.add(Embedding(VOCAB_SIZE, 64, input_length=TEXT_SIZE, name='Embedding'))
model.add(Bidirectional(GRU(64, activation='tanh', name='GRU'), name='Bidirectional'))
model.add(Dropout(0.5, name='Dropout-1'))
```

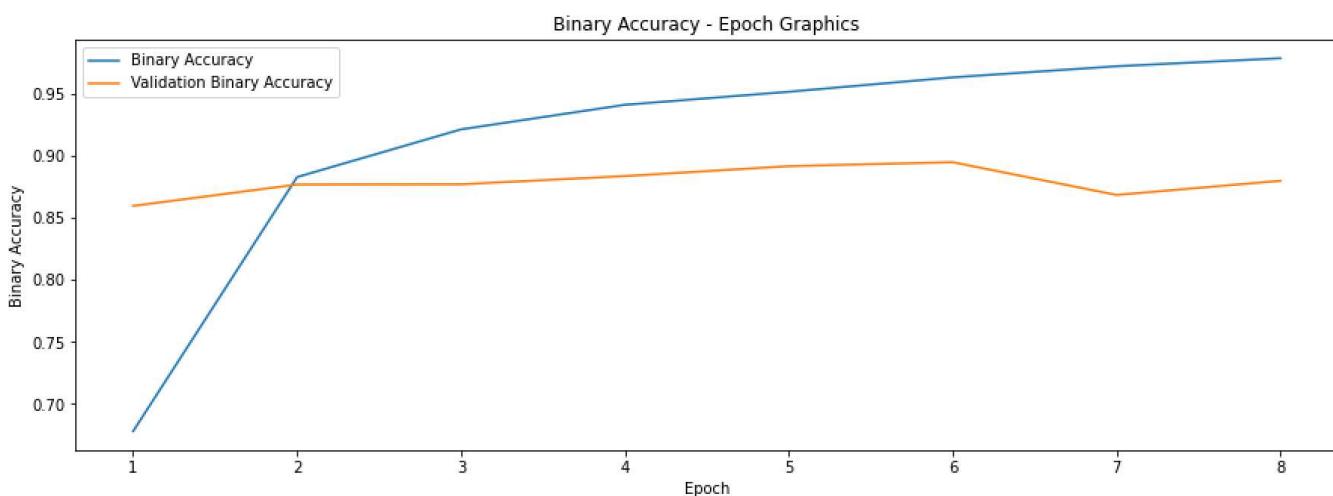
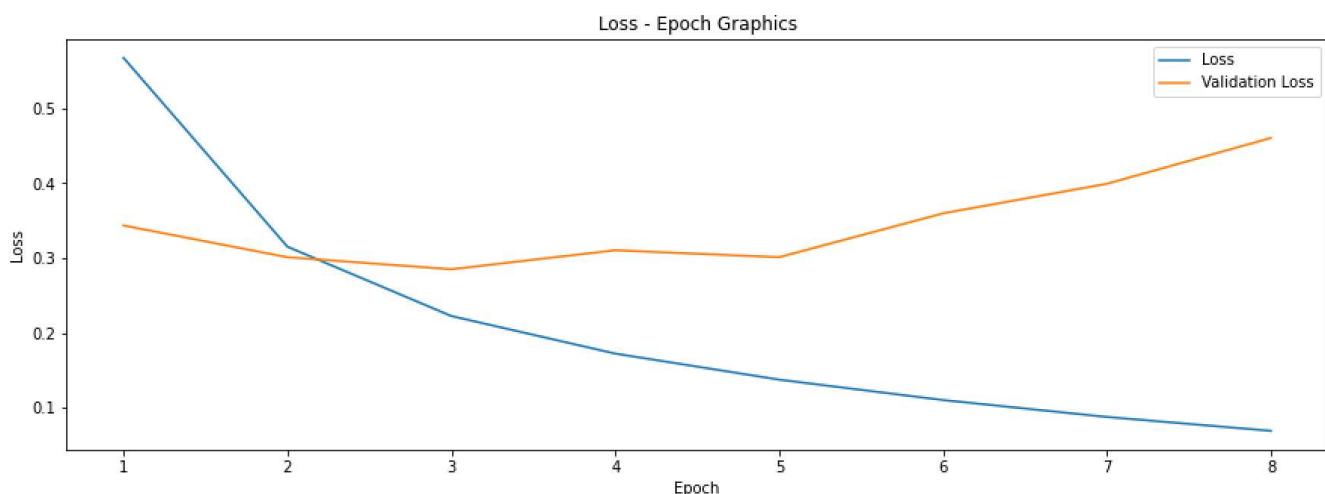
```

model.add(Dense(64, activation='relu', name='Dense'))
model.add(Dropout(0.5, name='Dropout-2'))
model.add(Dense(1, activation='sigmoid', name='Output'))

model.summary()

```

Modeli bu biçimde oluşturduğumuzda elde ettiğimiz epoch grafiği şöyledir:



Test sonuçları da şöyle elde edilmiştir:

```

loss --> 0.31287142634391785
binary_accuracy --> 0.8642799854278564

```

Geri Beslemeli Ağların Gelecek Kestiriminde Kullanılması ve Jena Climate Örneği

Biz yukarıdaki IMDB örneğinde geri beslemeli ağları yazışal metinlerden anlam çıkarmayı hedefleyen IMDB verilerinde kullandık. Halbuki bu ağlar yazışal verilerin dışında tamamen sayısal verilerin söz konusu olduğu kestirimlerde de kullanılabilir. Çünkü pek çok sayısal veri tipki yazısı oluşturan sözcükler gibi bir bağlama sahip olabilmektedir. Örneğin hava tahmini, borsada belli bir kağıdın gelecekteki değerinin tahmini geçmiş verilere bakılarak geri beslemeli bir ağ ile yapılmak istenebilir. Bir hava tahmini uygulamasında bir sonraki günde hava önceki günde havalara göre yani bir bağlam içerisinde oluşmaktadır. Benzer biçimde borsada bir kağıdın ya da kripto paranın fiyatı bir bağlam içerisinde değişmektedir.

Biz burada hava tahmini üzerinde örnek bir uygulama yapacağız. Geçmiş hava durumu raporlarına bakarak bir gün sonraki hava sıcaklığını tahmin edeceğiz. Tabii şüphesiz bu tahmin daha önce yaptığımız gibi geri beslemeli olmayan ağlarla (feed forward networks) da yapılabilir. Ancak biz burada bu kestirimini LSTM katmanlı bir ağ yapacağız.

Hava durumuna ilişkin "Jena Climate" örnek verileri aşağıdaki adresten indirilebilir:

<https://www.kaggle.com/stytch16/jena-climate-2009-2016>

Buradan indirilen "jena_climate_2009_2016.csv" dosyasının görünümü şöyledir:

```
Date Time","p (mbar)","T (degC)","Tpot (K)","Tdew (degC)","rh (%)","VPmax (mbar)","VPact (mbar)","VPdef (mbar)","sh (g/kg)","H2OC (mmol/mol)","rho (g/m**3)","wv (m/s)","max. wv (m/s)","wd (deg)"  
01.01.2009 00:10:00,996.52,-8.02,265.40,-8.90,93.30,3.33,3.11,0.22,1.94,3.12,1307.75,1.03,1.75,152.30  
01.01.2009 00:20:00,996.57,-8.41,265.01,-9.28,93.40,3.23,3.02,0.21,1.89,3.03,1309.80,0.72,1.50,136.10  
01.01.2009 00:30:00,996.53,-8.51,264.91,-9.31,93.90,3.21,3.01,0.20,1.88,3.02,1310.24,0.19,0.63,171.60  
01.01.2009 00:40:00,996.51,-8.31,265.12,-9.07,94.20,3.26,3.07,0.19,1.92,3.08,1309.19,0.34,0.50,198.00  
01.01.2009 00:50:00,996.51,-8.27,265.15,-9.04,94.10,3.27,3.08,0.19,1.92,3.09,1309.00,0.32,0.63,214.30  
01.01.2009 01:00:00,996.50,-8.05,265.38,-8.78,94.40,3.33,3.14,0.19,1.96,3.15,1307.86,0.21,0.63,192.70  
01.01.2009 01:10:00,996.50,-7.62,265.81,-8.30,94.80,3.44,3.26,0.18,2.04,3.27,1305.68,0.18,0.63,166.50  
01.01.2009 01:20:00,996.50,-7.62,265.81,-8.36,94.40,3.44,3.25,0.19,2.03,3.26,1305.69,0.19,0.50,118.60  
01.01.2009 01:30:00,996.50,-7.91,265.52,-8.73,93.80,3.36,3.15,0.21,1.97,3.16,1307.17,0.28,0.75,188.50  
...
```

Bu CSV dosyasının başında bir başlık kısmı vardır. Satırlar virgülerle ayrılmış değerler içermektedir. Hava durumuna ilişkin değerler 2009'dan 2016'ya kadar 10 dakika aralıklarla elde edilmiştir. İlk sütunda elde edilen bilginin tarih ve zaman bilgisi, üçüncü sütunda (2'inci indeksli sütunda) ise havanın derece cinsinden sıcaklığı bulunmaktadır.

İlk aşamada bizim bu CSV dosyasını alarak bir ndarray nesnesine dönüştürmemiz gereklidir. Tabii bu dönüştürme sırasında ilk sütunun da atılması uygun olur. Çünkü eğitimde bu sütunu kullanmayacağız. Önce CSV dosyasını okuyalım:

```
import numpy as np  
  
PREDICT_INTERVAL = 144  
  
dataset_x = np.loadtxt('jena_climate_2009_2016.csv', delimiter=',', skiprows=1,  
usecols=range(1, 15), dtype=np.float32)
```

Burada biz ilk sütunu atarak diğer tüm sütunları elde ettik. Bunlar bizim x verilerimizi oluşturacaktır. Şimdi dataset_y verilerini oluşturalım:

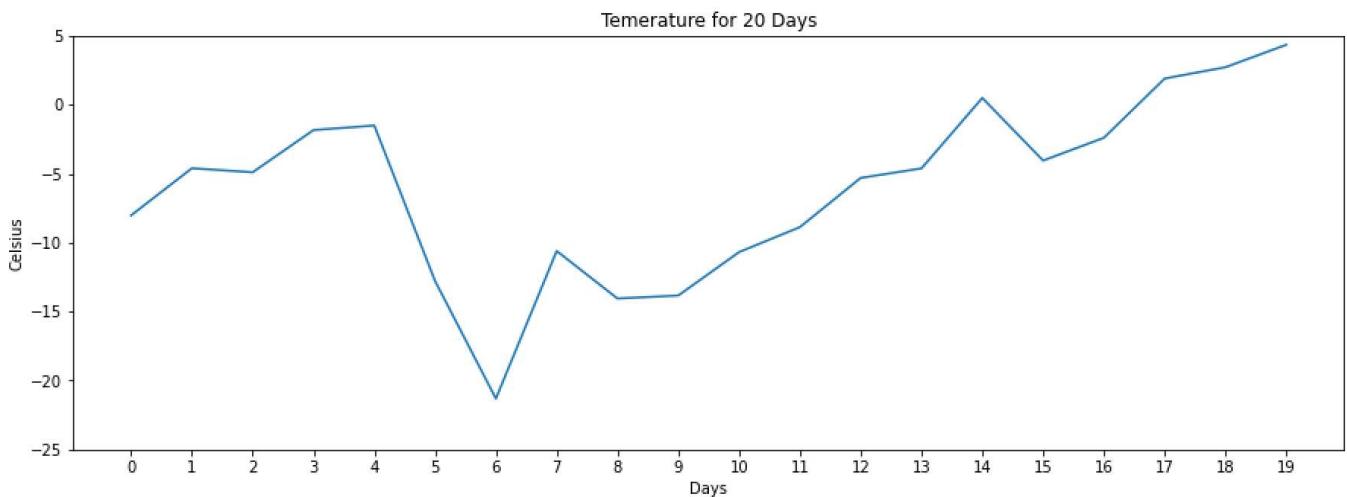
```
dataset_y = dataset_x[:, 1]
```

dataset_x matrisindeki her bir satırın öncekinden 10 dakika sonraki hava durumu bilgisini içerdigini söylemiştim. O halde örneğin index değeri dataset_x matrisindeki bir satırın sıra numarasını göstermek üzere, bu satırın belirttiği zamandan 1 gün sonraki hava sıcaklığı dataset_y[index + 144] ifadesiyle elde edilecektir. $24 * 6 // 10 = 144$ olduğuna dikkat ediniz.

Şimdi 20 günlük hava sıcaklığının grafiğini şöyle çizebiliriz:

```
import matplotlib.pyplot as plt  
  
figure = plt.gcf()  
figure.set_size_inches((15, 5))  
plt.title('Temerature for 20 Days')  
plt.xlabel('Days')  
plt.ylabel('Celsius')  
plt.ylim((-25, 5))  
plt.xticks(range(20))  
  
plt.plot(range(20), dataset_y[0:PREDICT_INTERVAL * 20:PREDICT_INTERVAL])  
plt.show()
```

Şöyledir bir grafik elde edilmişdir:



Şimdi de elimizdeki veri kümesini eğitim ve test olmak üzere iki kısma ayıralım:

```
from sklearn.model_selection import train_test_split

training_dataset_x, test_dataset_x, training_dataset_y, test_dataset_y =
train_test_split(dataset_x[:-PREDICT_INTERVAL], dataset_y[PREDICT_INTERVAL:], test_size=0.2)
```

Şimdi verileri ölçeklendirelim. Doğa olayları genel olarak normal dağılıma uydugu için Min-Max ölçeklendirmesi yerine Standart ölçeklendirme uygulayacağız:

```
from sklearn.preprocessing import StandardScaler

ss = StandardScaler()
ss.fit(training_dataset_x)
training_dataset_x = ss.transform(training_dataset_x)
test_dataset_x = ss.transform(test_dataset_x)
```

Önce modelimizi ileri beslemeli (feed forward) ağ ile deneyelim.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dropout, Dense

model = Sequential(name='Jane-Climate')
model.add(Dense(128, input_dim=training_dataset_x.shape[1], activation='relu', name='Dense-1'))
model.add(Dropout(0.2, name='Dropout-1'))
model.add(Dense(128, activation='relu', name='Dense-2'))
model.add(Dropout(0.2, name='Dropout-2'))
model.add(Dense(1, activation='linear', name='Output'))

model.summary()

model.compile(optimizer='rmsprop', loss='mse', metrics=['mae'])

from tensorflow.keras.callbacks import EarlyStopping

esc = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True, verbose=1)
hist = model.fit(training_dataset_x, training_dataset_y, batch_size=32, epochs=20,
validation_split=0.2, callbacks=[esc])

import matplotlib.pyplot as plt

figure = plt.gcf()
figure.set_size_inches((15, 5))
```

```

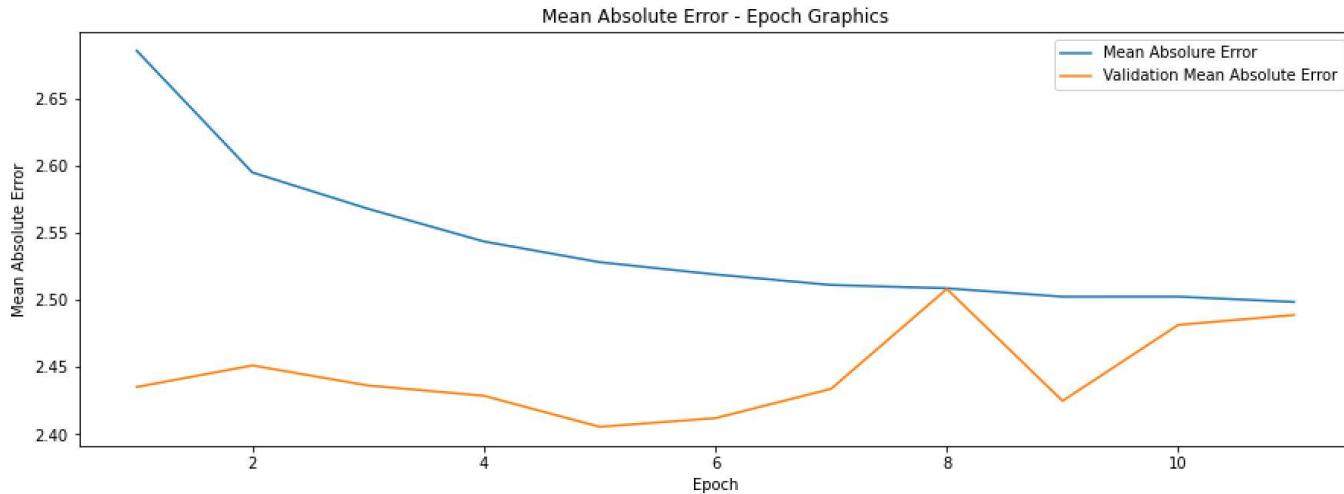
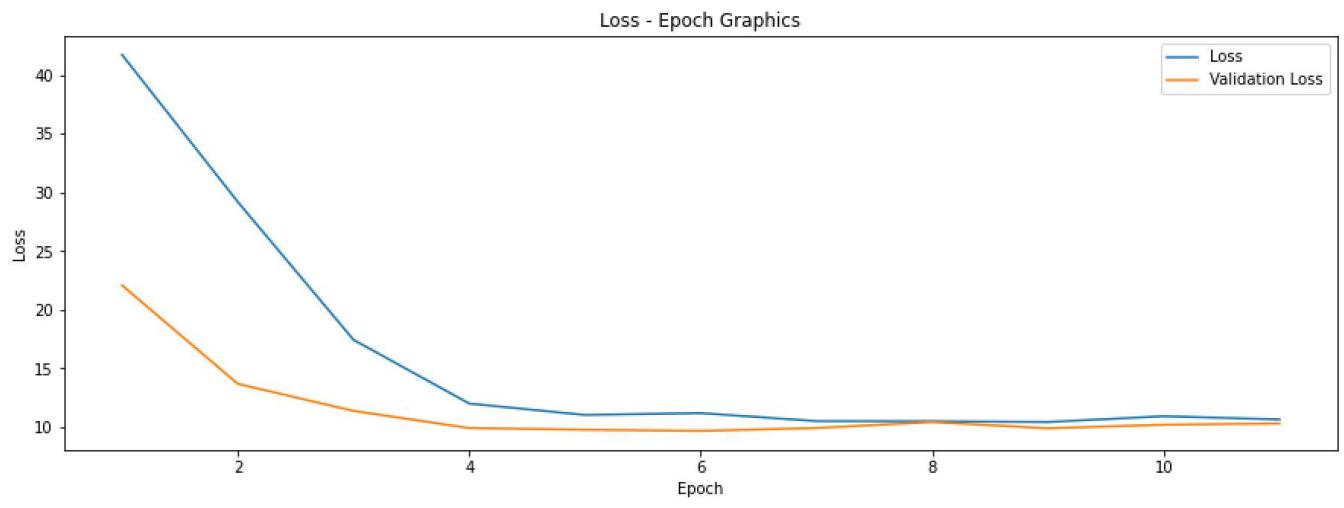
plt.title('Loss - Epoch Graphics')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.plot(range(1, len(hist.history['loss']) + 1), hist.history['loss'])
plt.plot(range(1, len(hist.history['val_loss']) + 1), hist.history['val_loss'])
plt.legend(['Loss', 'Validation Loss'])
plt.show()

figure = plt.gcf()
figure.set_size_inches((15, 5))
plt.title('Mean Absolute Error - Epoch Graphics')
plt.xlabel('Epoch')
plt.ylabel('Mean Absolute Error')
plt.plot(range(1, len(hist.history['mae']) + 1), hist.history['mae'])
plt.plot(range(1, len(hist.history['val_mae']) + 1), hist.history['val_mae'])
plt.legend(['Mean Absolute Error', 'Validation Mean Absolute Error'])
plt.show()

eval_result = model.evaluate(test_dataset_x, test_dataset_y)
for i in range(len(eval_result)):
    print(f'{model.metrics_names[i]} --> {eval_result[i]}')

```

Elde edilen epoch grafiği şöyledir:



Test soncunda elde edilen değerler de şöyledir:

```

loss --> 9.610762596130371
mae --> 2.414771556854248

```

Yukarıdaki kodu bir bütün olarak aşağıda veriyoruz:

```
import numpy as np

PREDICT_INTERVAL = 144

dataset_x = np.loadtxt('jena_climate_2009_2016.csv', delimiter=',', skiprows=1,
usecols=range(1, 15), dtype=np.float32)

dataset_y = dataset_x[:, 1]

import matplotlib.pyplot as plt

figure = plt.gcf()
figure.set_size_inches((15, 5))
plt.title('Temerature for 20 Days')
plt.xlabel('Days')
plt.ylabel('Celsius')
plt.ylim((-25, 5))
plt.xticks(range(20))
plt.plot(range(20), dataset_y[0:PREDICT_INTERVAL * 20:PREDICT_INTERVAL])
plt.show()

from sklearn.model_selection import train_test_split

training_dataset_x, test_dataset_x, training_dataset_y, test_dataset_y =
train_test_split(dataset_x[:-PREDICT_INTERVAL],

dataset_y[PREDICT_INTERVAL:],

test_size=0.2)

from sklearn.preprocessing import StandardScaler

ss = StandardScaler()
ss.fit(training_dataset_x)
training_dataset_x = ss.transform(training_dataset_x)
test_dataset_x = ss.transform(test_dataset_x)

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dropout, Dense

model = Sequential(name='Jane-Climate')
model.add(Dense(128, input_dim=training_dataset_x.shape[1], activation='relu', name='Dense-1'))
model.add(Dropout(0.2, name='Dropout-1'))
model.add(Dense(128, activation='relu', name='Dense-2'))
model.add(Dropout(0.2, name='Dropout-2'))
model.add(Dense(1, activation='linear', name='Output'))

model.summary()

model.compile(optimizer='rmsprop', loss='mse', metrics=['mae'])

from tensorflow.keras.callbacks import EarlyStopping

esc = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True, verbose=1)
hist = model.fit(training_dataset_x, training_dataset_y, batch_size=32, epochs=20,
validation_split=0.2,
callbacks=[esc])

import matplotlib.pyplot as plt

figure = plt.gcf()
```

```

figure.set_size_inches((15, 5))
plt.title('Loss - Epoch Graphics')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.plot(range(1, len(hist.history['loss']) + 1), hist.history['loss'])
plt.plot(range(1, len(hist.history['val_loss']) + 1), hist.history['val_loss'])
plt.legend(['Loss', 'Validation Loss'])
plt.show()

figure = plt.gcf()
figure.set_size_inches((15, 5))
plt.title('Mean Absolute Error - Epoch Graphics')
plt.xlabel('Epoch')
plt.ylabel('Mean Absolute Error')
plt.plot(range(1, len(hist.history['mae']) + 1), hist.history['mae'])
plt.plot(range(1, len(hist.history['val_mae']) + 1), hist.history['val_mae'])
plt.legend(['Mean Absolute Error', 'Validation Mean Absolute Error'])
plt.show()

eval_result = model.evaluate(test_dataset_x, test_dataset_y)
for i in range(len(eval_result)):
    print(f'{model.metrics_names[i]} -> {eval_result[i]}')

```

Şimdi bu kestirim için LSTM katmanına sahip geri beslemeli bir model kullanalım. Burada ağımızın belli sayıda 10'ar dakikalık bilgileri vererek belli bir zaman sonraki hava sıcaklığını tahmin etmeye çalışacağız. Önce bazı parametrik değerleri belirleyip CSV dosyasını okuyalım:

```

import numpy as np

PREDICT_INTERVAL = 144
LOOKBACK_INTERVAL = 144
BATCH_SIZE = 32

dataset_x = np.loadtxt('jena_climate_2009_2016.csv', delimiter=',', skiprows=1,
usecols=range(1, 15), dtype=np.float32)

dataset_y = dataset_x[:, 1]

```

Burada PREDICT_INTERVAL bizim kaç 10 dakika sonraki hava sıcaklığını tahmin etmek istediğimizi belirtiyor. Bu örnekte yine biz bir gün sonraki hava sıcaklığını tahmin edeceğiz. Biz burada zamansal olarak ağımızın bir bellek kazandırmak istiyoruz. LOOKBACK_INTERVAL ağımızın ne kadar geçmiş anımsayacağıdır. Yani biz burada belirtilen değer kadar 10 dakikalık verileri zamansal olarak ağımızın vereceğiz.

Bu uygulamada dikkat edilmesi gereken nokta bizim belli sayıda 10 dakikalık verileri ağa verip belli bir zaman sonraki değeri tahmin etmeye çalıştığımızdır. Bunun için training_dase_x ve training_dataset_y dizilerini oluşturmak çok zor ve verimsiz bir yöntemdir. Bu nedenle biz burada parçalı verilerle eğitim işlemi uygulayacağz. Yani her batch işleminde bizim metodumuz çağrılmak ve biz de bir batch uzunluğu kadar bilgiyi oluşturup vereceğiz. Böylelikle büyük bir bellek kullanmak zorunda kalmayacağz. Şimdi veri kümemizi önce eğitim ve test biçiminde iki parçaya ayıralım. Sonra da eğitim veri kümelerinden sınıma veri kümelerini oluşturalım:

```

from sklearn.model_selection import train_test_split

training_dataset_x, test_dataset_x, training_dataset_y, test_dataset_y =
train_test_split(dataset_x, dataset_y,
shuffle=False, test_size=0.1)

from sklearn.preprocessing import StandardScaler

ss = StandardScaler()
ss.fit(training_dataset_x)

```

```

training_dataset_x = ss.transform(training_dataset_x)
test_dataset_x = ss.transform(test_dataset_x)

training_dataset_x, validation_dataset_x, training_dataset_y, validation_dataset_y =
train_test_split(training_dataset_x, training_dataset_y, shuffle=False, test_size=0.2)

```

Şimdi parçalı verilerle eğitim için generator sınıfımızı yazalım:

```

from tensorflow.keras.utils import Sequence

class DataGenerator(Sequence):
    def __init__(self, dataset_x, dataset_y, batch_size):
        self.dataset_x = dataset_x
        self.dataset_y = dataset_y
        self.batch_size = batch_size
        self.indices = np.arange((len(self.dataset_x) - PREDICT_INTERVAL) // self.batch_size,
                               dtype=np.int32)

    def __len__(self):
        return (len(self.dataset_x) - PREDICT_INTERVAL) // self.batch_size

    def __getitem__(self, index):
        result_x = np.zeros((BATCH_SIZE, LOOKBACK_INTERVAL, training_dataset_x.shape[1]))
        result_y = np.zeros(BATCH_SIZE)

        for i in range(self.batch_size):
            start = self.indices[index] * self.batch_size + i
            result_x[i] = self.dataset_x[start:start + LOOKBACK_INTERVAL]
            result_y[i] = self.dataset_y[start + PREDICT_INTERVAL]

        return result_x, result_y

    def on_epoch_end(self):
        np.random.shuffle(self.indices)

```

Burada DataGenerator sınıfının `__init__` metodu bizden sırasıyla `dataset_x`, `dataset_y` ve `batch_size` değerlerini alıp bunları nesnenin örnek özniteliklerinde saklamaktadır. Anımsanacağı gibi sınıfın `__len__` metodu bir epoch işleminde kaç adet batch kullanacağını belirtmektedir. Yine her batch için batch numarasının `__getitem__` metoduna geçirildiğini anımsayınız. Biz de bu `__getitem__` metodu içerisinde bir batch uzunluğu kadar satır ve sütundan oluşan matris yaratıp bu matrisin içini doldurduk. Her bir batch elemanına karşı gelen değerin `PREDICT_INTERVAL` kadar sonraki değer olduğuna dikkat ediniz. Yine her epoch sonucunda eğitim veri kümesi karıştırılmaktadır.

Artık sıra modelimizi kurmaya gelmiştir. Modelin girdi katmanını LSTM olarak alacağız. LSTM katmanın girdisi `LOOKBACK_INTERVAL` kadar 10'ar dakikalık hava durumu verilerinden oluşacaktır ve bu 10'ar dakikalık veriler modele zamansal (temporal) olarak uygulanacaktır:

```

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dropout, LSTM, Dense

model = Sequential(name='Jane-Climate-LSTM')
model.add(LSTM(64, activation='tanh', name='LSTM', input_shape=(None,
                                                               training_dataset_x.shape[1])))
model.add(Dense(64, activation='relu', name='Dense-1'))
model.add(Dropout(0.2, name='Dropout-1'))
model.add(Dense(64, activation='relu', name='Dense-2'))
model.add(Dropout(0.2, name='Dropout-2'))
model.add(Dense(1, activation='linear', name='Output'))

model.summary()

```

Artık modeli derleyip eğitebiliriz:

```

model.compile(optimizer='rmsprop', loss='mse', metrics=['mae'])

from tensorflow.keras.callbacks import EarlyStopping

esc = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True, verbose=1)
hist = model.fit(DataGenerator(training_dataset_x, training_dataset_y, BATCH_SIZE),
validation_data=DataGenerator(validation_dataset_x, validation_dataset_y, BATCH_SIZE),
epochs=10, callbacks=[esc])

```

Epoch grafiklerini çizdirelim:

```

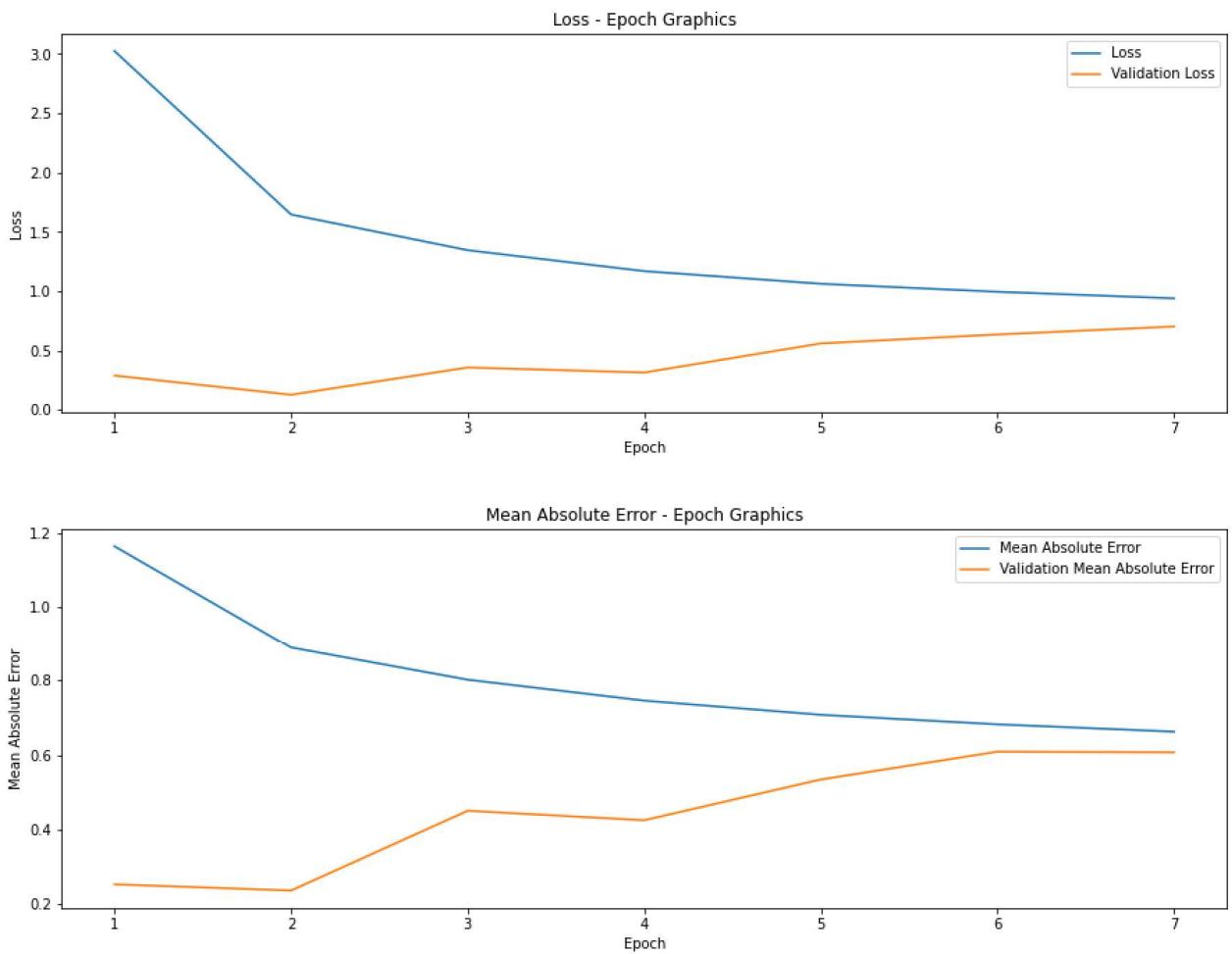
import matplotlib.pyplot as plt

figure = plt.gcf()
figure.set_size_inches((15, 5))
plt.title('Loss - Epoch Graphics')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.plot(range(1, len(hist.history['loss']) + 1), hist.history['loss'])
plt.plot(range(1, len(hist.history['val_loss']) + 1), hist.history['val_loss'])
plt.legend(['Loss', 'Validation Loss'])
plt.show()

figure = plt.gcf()
figure.set_size_inches((15, 5))
plt.title('Mean Absolute Error - Epoch Graphics')
plt.xlabel('Epoch')
plt.ylabel('Mean Absolute Error')
plt.plot(range(1, len(hist.history['mae']) + 1), hist.history['mae'])
plt.plot(range(1, len(hist.history['val_mae']) + 1), hist.history['val_mae'])
plt.legend(['Mean Absolute Error', 'Validation Mean Absolute Error'])
plt.show()

```

Şu grafikler elde edilmiştir:



Şimdi de modelimizi test edelim:

```
eval_result = model.evaluate(DataGenerator(test_dataset_x, test_dataset_y, BATCH_SIZE))
for i in range(len(eval_result)):
    print(f'{model.metrics_names[i]} --> {eval_result[i]}')
```

Şu değerler elde edilmiştir:

```
loss --> 0.1031017005443573
mae --> 0.22029602527618408
```

Bu değerlerin ileri beslemeli ağdan elde ettiğimiz değerlerden oldukça iyi olduğuna dikkat ediniz.

Pekiyi burada oluşturduğumuz modelde kestirim işlemini nasıl yapacağız? Daha önceki çeşitli defalar belirttiğimiz gibi kestirim işlemi eğitim nasıl yapılmışsa o koşullarda yapılmalıdır. Biz burada eğitimde 144 tane 10'ar dakikalık verileri standart ölçeklendirmeye sokarak kullandık. O halde kesitimi de tamamen aynı biçimde yapmamız gerekmektedir. Yani bu örneğimizde kestirim yapabilmemiz için elimizde 144 tane 10 dakikalık verilerin olması gerekmektedir.

Geri Beslemeli Ağlarla Çıktı Üretimi

Biz şimdide kadarki örneklerimizde gelecek kestirimini yapmaya çalıştık. Ancak bazen bu gelecek kestirimini çıktı üretimleri için de kullanılabilmektedir. Özellikle metinlerin üretiminde 2015 yılından itibaren LSTM tabanlı geri beslemeli ağlar kullanılmaya başlanmıştır. Bu konuda önemli başarılar elde edilmiştir.

Geri beslemeli ağlarla metin oluşturma işlemi tipik olarak şöyle yapılmaktadır:

- 1) Üretilen metni belli bir kişinin tarzına uydurmak için o kişinin yazıları elde edilir.

2) Sonra bir geri beslemeli sinir ağı oluşturularak sinir ağı bu yazılarla eğitilir. Burada karakter temelli bir çıktı üretiminin söz konusu olduğunu düşünelim. Bunun için seçilen kişiye ilişkin yazıların belli miktardaki karakterden oluşan kısımları - örneğin 32 karakter olduğunu varsayıyalım- ve bu kısımdan sonra gelen karakterler eğitim amacıyla toplanır. Böylece bu işlemden 32 karakterden oluşan bir yazı parçaları ile (dataset_x), 1 karakterden oluşan sonuç değerleri (dataset_y) elde edilecektir. İ

3) Geri beslemeli bir yapay sinir ağı modeli kurularak ağ oluşturulan veri kümesiyle eğitilir. Geri besleme için genellikle LSTM modeli tercih edilmektedir.

4) Artık elimizde 32 karakterini girdi olarak verdigimiz bir metnin 33'üncü karakterini tahmin edebilen bir geri beslemeli yapay sinir ağı modeli vardır. Bu durumda biz 32 karakterlik bir yazıyı başlangıç yazısı olarak verirsek ağdan bunun 33'üncü karakterini elde edebiliriz. Bundan sonra yine bu 33 karakterli yazının son 32 karakterini alarak yeni bir karakter elde ederiz. Bu işlemi böyle sürdürürsek istediğimiz uzunlukta bir yazı elde etmiş oluruz.

5) Modelin bu biçimde oluşturulması yazının tek düzeye ve yaratıcılık içermeyen bir yapıda olmasına yol açabilmektedir. Bu nedenle her defasında ağdan elde edilen karakteri doğrudan kullanmak yerine işin içine bir miktar rassallık da katılarak belli bir olasılık dağılımı çerçevesinde yeni karakterin elde edilmesi daha yaratıcı bir metnin oluşmasına yol açmaktadır.

Tabii çıktı üretimi aslında yalnızca metinsel girdiler üzerinde değil yaratıcılık etkinliği olarak değerlendirilen diğer girdiler üzerinde de uygulanabilmektedir. Örneğin belli bir bestecinin tarzına göre beste yapan bir sinir ağı modeli oluşturulabilir.

Şimdi Nietzsche'nin metinlerinden oluşan bir yazıyı kullanarak Nietzsche tarzında metin üreten bir uygulama üzerinde duralım.

Akıntı Notu: Bu uygulama "Deep Learning With Python (François Chollet)" isimli kitabın 274. Sayfasında da bulunmaktadır.

Uygulamaya konu olan Nietzsche metinleri aşağıdaki adresden indirilebilir:

<https://s3.amazonaws.com/text-datasets/nietzsche.txt>

Aslında indirme işlemini tensorflow.keras.utils modülündeki get_file fonksiyonuyla da yapabiliriz:

```
import tensorflow.keras

path = tensorflow.keras.utils.get_file('nietzsche.txt', origin='https://s3.amazonaws.com/text-datasets/nietzsche.txt')

with open(path) as f:
    text = f.read().lower()

print(text)
```

Burada artık tüm metin bir string olarak text değişkeninde bulunmaktadır.

Şimdi metin okunduktan sonra yazıda 3'er atlamlı olarak 60'ar karakterden oluşan parçaları sentences isimli bir listeye yerlestirelim. Yapay sinir ağı 60 karakterden sonraki 61'inci karakteri tahmin edeceğini dolayısıyla bu 60 karakterden sonraki karakteri de next_char isimli bir listede toplayacağız:

```
TEXT_LENGTH = 60
STEP = 3

sentences = []
next_chars = []

for i in range(0, len(text) - TEXT_LENGTH, STEP):
    sentences.append(text[i:i + TEXT_LENGTH])
    next_chars.append(text[i + TEXT_LENGTH])
```

```

next_chars.append(text[i + TEXT_LENGTH])

print(f'Number of sequences: {len(sentences)}')

```

Böylece biz eğitimde kullanacağımız 60'ar karakterden oluşan bir liste (training_dataset_x) ile bunlardan sonra gelen karakterlerden oluşan diğer bir liste (training_dataset_y) elde etmiş olduk. Bundan sonra yazı içerisinde farklı kaç tane karakter olduğunu tespit edip bu karakterleri de bir listeye yerleştirerek koda devam edelim:

```

chars = sorted(list(set(text)))
print('Number of unique chars: {}'.format(len(chars)))

```

Tabii bizim yazı parçalarındaki karakterlerin her birini "one hot encoding" biçiminde oluşturmamız gereklidir. Zira eğer biz karakter kodlarını doğrudan kullanırsak model sanki sıralı (ordinal) bir ölçek söz konusu olmuş gibi davranış olacaktır. One hot encoding işlemini kolaylaştırmak için bir sözlük oluşturabiliriz. Bu sözlük sayesinde yazı içerisindeki bir karakter verildiğinde onun indeksini hızlı bir biçimde elde edebileceğiz:

```
char_dict = {char: index for index, char in enumerate(chars)}
```

Bu özlük sayesinde biz karakteri verdiğimizde ona karşı gelen sayısal değeri alabileceğiz. Artık sıra training_dataset_x ve training_dataset_y nesnelerinin ndarray olarak oluşturulmasına geldi. Bunun için önce bu diziler içinde sıfır olacak biçimde aşağıdaki yaratalım:

```

import numpy as np

training_dataset_x = np.zeros((len(sentences), TEXT_LENGTH, len(chars)), dtype=np.uint8)
training_dataset_y = np.zeros((len(sentences), len(chars)), dtype=np.uint)

```

Burada training_dataset_x üç boyutlu bir numpy dizisidir. Dizinin TEXT_LENGTH (60) uzunluktaki her yazının her karakterinin index numaralarını one hot encoding biçiminde tuttuğuna dikkat ediniz. training_dataset_y ise yazındaki TEXT_LENGTH + 1'inci karakterin one hot encoding karşılığını tutmaktadır. Mademki bu iki dizinin tüm elemanları np.zeros fonksiyonuyla sıfır yapılmıştır. O halde biz yalnızca uygun elemanı 1 yaparak one hot encoding vektörlerini kolay bir biçimde oluşturabiliriz:

```

for i, sentence in enumerate(sentences):
    for k, char in enumerate(sentence):
        training_dataset_x[i, k, char_dict[char]] = 1
        training_dataset_y[i, char_dict[next_chars[i]]] = 1

```

Artık trainig_dataset_x ve training_dataset_y one hot encoding biçiminde oluşturmuş durumdadır. Şimdi artık LSTM katmanı içeren geri beslemeli ağ modelimizi oluşturabiliriz:

```

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dropout, Dense

model = Sequential(name='TextProduction')
model.add(LSTM(128, activation='tanh', input_shape=(TEXT_LENGTH, len(chars)), name='LSTM'))
model.add(Dropout(0.3, name='Dropout'))
model.add(Dense(len(chars), activation='softmax', name='Output'))
model.summary()

```

Cıktı katmanının toplamda len(chars) uzunlukta nöronlardan oluştuğuna dikkat ediniz. Cıktı katmanının aktivasyon fonksiyonu "softmax" olduğuna göre tüm çıktı nöronlarının toplam değeri 1 olacaktır. Tabii biz bunlar arasından en yüksek değeri tahmin olarak alacağız. Ağımızın optimizasyon algoritması "rmsprop" olarak loss fonksiyonu da daha önceki benzer sınıflandırma örneklerinde yapmış olduğumuz gibi "categorical_crossentropy" olarak alınmıştır. Şimdi artık ağımızı derleyip eğitebiliriz:

```

optimizer = RMSprop(lr=0.01)
model.compile(optimizer, loss='categorical_crossentropy', metrics=['categorical_accuracy'])

```

```

from tensorflow.keras.callbacks import EarlyStopping

esc = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True, verbose=1)
hist = model.fit(training_dataset_x, training_dataset_y, batch_size=32, epochs=20,
validation_split=0.2, callbacks=[esc])

```

Burada "rmsprop" algoritması için learning_rate değerini biraz yükselttik. learning_rate parametresinin yükseltilmesi hedefe daha hızlı yakınsama sağlamaktadır. Bu konu ileride ele alınacaktır. Şimdi de epoch grafiklerini çizelim:

```

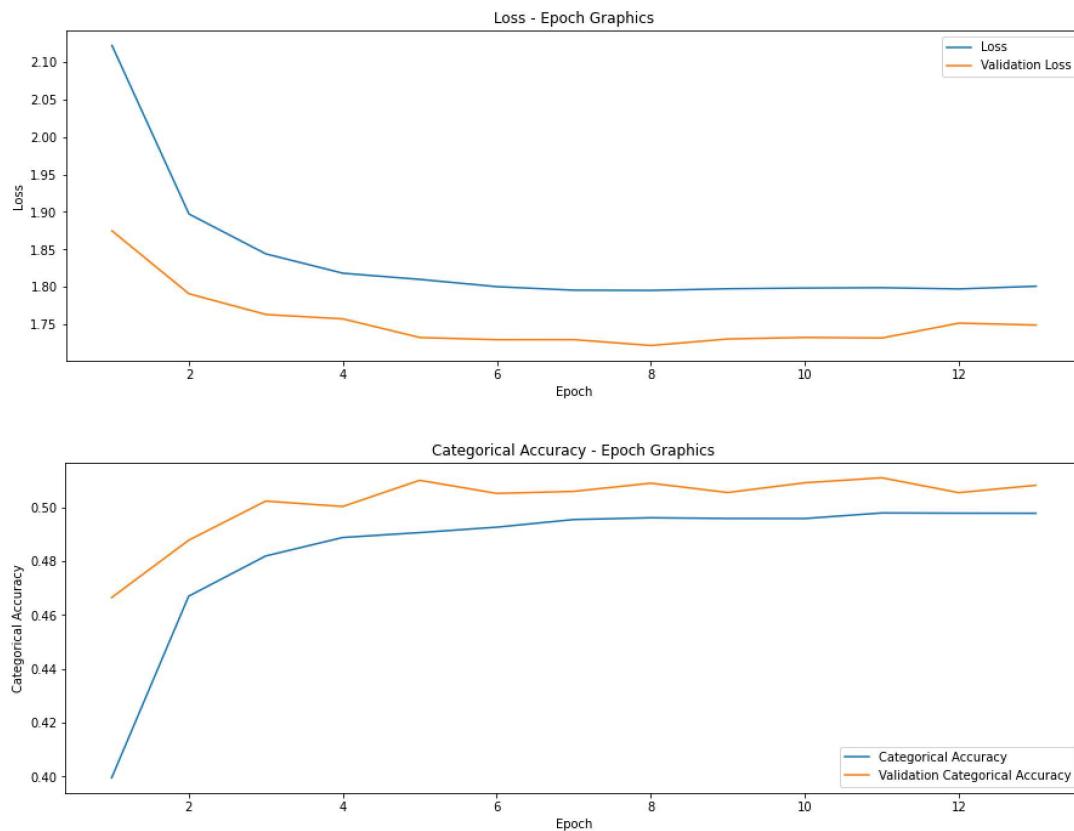
import matplotlib.pyplot as plt

figure = plt.gcf()
figure.set_size_inches((15, 5))
plt.title('Loss - Epoch Graphics')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.plot(range(1, len(hist.history['loss']) + 1), hist.history['loss'])
plt.plot(range(1, len(hist.history['val_loss']) + 1), hist.history['val_loss'])
plt.legend(['Loss', 'Validation Loss'])
plt.show()

figure = plt.gcf()
figure.set_size_inches((15, 5))
plt.title('Categorical Accuracy - Epoch Graphics')
plt.xlabel('Epoch')
plt.ylabel('Categorical Accuracy')
plt.plot(range(1, len(hist.history['categorical_accuracy']) + 1),
hist.history['categorical_accuracy'])
plt.plot(range(1, len(hist.history['val_categorical_accuracy']) + 1),
hist.history['val_categorical_accuracy'])
plt.legend(['Categorical Accuracy', 'Validation Categorical Accuracy'])
plt.show()

```

Şu grafikler elde edilmiştir:



Şimdi de belli bir TEXT_LENGTH (60) uzunluğunda yazı alıp bir döngü içerisinde ona 400 karakter ekleyerek yeni bir yazı üretelim. Bu işlemi şöyle yapabiliriz: TEXT_LENGTH uzunluğundaki yazıyı verip bundan edilecek TEXT_LENGTH + 1'inci karakteri tahmin ederiz. Sonra son TEXT_LENGTH uzunlukta yazıyı verip sonraki karakteri tahmin ederiz. Böylece 400 karakteri bir döngü içerisinde bu biçimde elde ederiz. Örneğimizde ağ tarafından elde edilen karakterlerin olasılık dağılımına göre biraz değiştirilmesi gerekebilmiştir. Yani üretilen karakterin değil de ona yakın olan bir karakterin seçilmesi tekdüzeliği engellemek bakımından uygun olur. Bunun için aşağıdaki gibi bir fonksiyondan faydalana bilir:

```
def sample(preds, temperature=1.0):
    preds = np.asarray(preds).astype('float64')
    preds = np.log(preds) / temperature
    exp_preds = np.exp(preds)
    preds = exp_preds / np.sum(exp_preds)
    probas = np.random.multinomial(1, preds, 1)

    return np.argmax(probas)
```

Programın yazı üretim kısmı için TEXT_LENGTH uzunluktaki yazıları one hot encoding biçimine dönüştürmemiz gereklidir. Bunun için aşağıdaki gibi bir fonksiyon yazabilirmiz:

```
def str_to_ohe(s):
    ohe = np.zeros(shape=(1, TEXT_LENGTH, len(chars)))
    for index, char in enumerate(s):
        ohe[0, index, char_dict[char]] = 1.0

    return ohe
```

Şimdi rastgele bir TEXT_LENGTH (60) uzunluğunda bir yazı oluşturalım:

```
r = np.random.randint(len(sentences) - maxlen - 1)
random_initial_sentence = text[r:r + 60]
```

Şimdi de 400 karakterlik bir yazı üretelim:

```
generated_text = initial_text
for i in range(400):
    ohe = str_to_ohe(generated_text[i: TEXT_LENGTH + i])
    result = model.predict(ohe)[0]
    char = chars[sample(result, 0.2)]
    generated_text += char

print(f'{generated_text[0:TEXT_LENGTH]}\n{generated_text[TEXT_LENGTH:]}')
```

Tüm kodları aşağıda bir bütün olarak veriyoruz:

```
from tensorflow.keras.utils import get_file

path = get_file('nietzsche.txt', origin='https://s3.amazonaws.com/text-datasets/nietzsche.txt')

with open(path) as f:
    text = f.read().lower()

print(text)

TEXT_LENGTH = 60
STEP = 3

sentences = []
next_chars = []

for i in range(0, len(text) - TEXT_LENGTH, STEP):
```

```

sentences.append(text[i:i + TEXT_LENGTH])
next_chars.append(text[i + TEXT_LENGTH])

print(f'Number of sequences: {len(sentences)}')

chars = sorted(list(set(text)))
print(f'Number of unique chars: {len(chars)}')

char_dict = {char: index for index, char in enumerate(chars)}

import numpy as np

training_dataset_x = np.zeros((len(sentences), TEXT_LENGTH, len(chars)), dtype=np.uint8)
training_dataset_y = np.zeros((len(sentences), len(chars)), dtype=np.uint8)

for i, sentence in enumerate(sentences):
    for k, char in enumerate(sentence):
        training_dataset_x[i, k, char_dict[char]] = 1
    training_dataset_y[i, char_dict[next_chars[i]]] = 1

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dropout, Dense

model = Sequential(name='TextProduction')
model.add(LSTM(128, activation='tanh', input_shape=(TEXT_LENGTH, len(chars)), name='LSTM'))
model.add(Dropout(0.3, name='Dropout'))
model.add(Dense(len(chars), activation='softmax', name='Output'))
model.summary()

from tensorflow.keras.optimizers import RMSprop

optimizer = RMSprop(learning_rate=0.01)
model.compile(optimizer, loss='categorical_crossentropy', metrics=['categorical_accuracy'])

from tensorflow.keras.callbacks import EarlyStopping

esc = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True, verbose=1)
hist = model.fit(training_dataset_x, training_dataset_y, batch_size=32, epochs=20,
validation_split=0.2,
callbacks=[esc])

import matplotlib.pyplot as plt

figure = plt.gcf()
figure.set_size_inches((15, 5))
plt.title('Loss - Epoch Graphics')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.plot(range(1, len(hist.history['loss']) + 1), hist.history['loss'])
plt.plot(range(1, len(hist.history['val_loss']) + 1), hist.history['val_loss'])
plt.legend(['Loss', 'Validation Loss'])
plt.show()

figure = plt.gcf()
figure.set_size_inches((15, 5))
plt.title('Categorical Accuracy - Epoch Graphics')
plt.xlabel('Epoch')
plt.ylabel('Categorical Accuracy')
plt.plot(range(1, len(hist.history['categorical_accuracy']) + 1),
hist.history['categorical_accuracy'])
plt.plot(range(1, len(hist.history['val_categorical_accuracy']) + 1),
hist.history['val_categorical_accuracy'])
plt.legend(['Categorical Accuracy', 'Validation Categorical Accuracy'])

```

```

plt.show()

def sample(preds, temperature=1.0):
    preds = np.asarray(preds).astype('float64')
    preds = np.log(preds) / temperature
    exp_preds = np.exp(preds)
    preds = exp_preds / np.sum(exp_preds)
    probas = np.random.multinomial(1, preds, 1)

    return np.argmax(probas)

def str_to_ohe(s):
    ohe = np.zeros(shape=(1, TEXT_LENGTH, len(chars)))
    for index, char in enumerate(s):
        ohe[0, index, char_dict[char]] = 1.0

    return ohe

r = np.random.randint(len(text) - TEXT_LENGTH + 1)
initial_text = text[r: r + TEXT_LENGTH]

generated_text = initial_text
for i in range(400):
    ohe = str_to_ohe(generated_text[i: TEXT_LENGTH + i])
    result = model.predict(ohe)[0]
    char = chars[sample(result, 0.2)]
    generated_text += char

print(f'{generated_text[0:TEXT_LENGTH]}{generated_text[TEXT_LENGTH:]}')

```

Yukarıdaki örnekte word embedding uygulanmamıştır. Bu örneğin word embedding uygulanmış halini de aşağıda veriyoruz. Anımsanacağı gibi Keras'taki Embedding katmanı girdi olarak 3 boyutlu değil 2 boyutlu indekslerden oluşan bir matris istemektedir. Çıktı olarak one hot encoding kullanılmaktadır.

```

import tensorflow.keras

path = tensorflow.keras.utils.get_file('nietzsche.txt', 'https://s3.amazonaws.com/text-dataset')
text = open(path).read().lower()

sentences = []
next_chars = []
maxlen = 60
step = 3

for i in range(0, len(text) - maxlen, step):
    sentences.append(text[i:i + maxlen])
    next_chars.append(text[i + maxlen])

chars = sorted(list(set(text)))
char_dict = dict((char, index) for index, char in enumerate(chars))

import numpy as np

training_dataset_x = np.zeros((len(sentences), maxlen), dtype='int8')
training_dataset_y = np.zeros((len(sentences), len(chars)), dtype='int8')

for i, sentence in enumerate(sentences):
    for k, char in enumerate(sentence):
        training_dataset_x[i, k] = char_dict[char]
    training_dataset_y[i, char_dict[next_chars[i]]] = 1

from tensorflow.keras.models import Sequential

```

```

from tensorflow.keras.layers import Embedding, Dropout, LSTM, Dense

model = Sequential()
model.add(Embedding(maxlen, 64))
model.add(Dropout(0.2))
model.add(LSTM(128))
model.add(Dropout(0.2))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(len(chars), activation='softmax'))

epochs = 1

model.compile(optimizer='RMSprop', loss='categorical_crossentropy')
hist = model.fit(training_dataset_x, training_dataset_y, batch_size=128, epochs=epochs)

def sample(preds, temperature=1.0):
    preds = np.asarray(preds).astype('float64')
    preds = np.log(preds) / temperature
    exp_preds = np.exp(preds)
    preds = exp_preds / np.sum(exp_preds)
    probas = np.random.multinomial(1, preds, 1)

    return np.argmax(probas)

def str_to_indexes(s):
    a = np.zeros((1, len(s)), dtype='int8')
    for index, char in enumerate(s):
        a[0, index] = char_dict[char]

    return a

import matplotlib.pyplot as plt

plt.title('Epoch - Loss Graph')
plt.xlabel = 'Epoch'
plt.ylabel = 'Loss'
plt.plot(range(1, len(hist.epoch) + 1), hist.history['loss'])
plt.legend(['loss', 'val_loss'])

r = np.random.randint(len(text) - maxlen - 1)
random_initial_sentence = text[r:r + maxlen]

generated_text = random_initial_sentence
for i in range(400):
    a = str_to_indexes(generated_text[i:i + maxlen])
    result = model.predict(a)[0]
    generated_text += chars[sample(result, 0.2)]

print('[{}][{}].format(generated_text[0:maxlen], generated_text[maxlen:]))
```

Keras'ta Modellerin Fonksiyonel Olarak Oluşturulması

Şimdiye kadar biz Keras'ta Sequential bir model nesnesi yarattık ve bu model nesnesine katmanları add metoduyla ekledik. Model sınıfının add metodu bir önceki katmanın çıktısını bir sonraki katmana girdi yapmaktadır. Zaten buradaki Sequential ismi de bunu anlatmaktadır. Bu biçimde model oluşturmak pek çok problem için yeterliyse de bazı tarz problemlerde yetersiz kalabilmektedir. Örneğin bazı modellerin girdileri birden fazla olabilmektedir. Benzer biçimde bazı modellerin çıktıları da yine birden fazla olabilmektedir. Halbuki Sequential modelde tek girdi ve tek çıktı vardır. Örneğin biz yazıyı hem sınıflandıracak olalım hem de yazının eleştiri derecesini tespit edecek olalım. Buradaki çıktı bir tane değildir, iki tanedir. Tabii ilk akla gelen şey modelleri ayrı ayrı kurmak ve iki ayrı çıktıyı ayrı ayrı elde etmektir. Fakat böylesi bir çaba fazladan programlama yükü oluşturmaktadır. Çünkü bu çıktılar için eğitimin yeniden

yapılması gereklidir. Ayrıca oluşan ağırlık değerlerinin yeniden saklanması da gerekmektedir. Oysa tek bir modelle iki ya da daha fazla çıktı elde edilebilir. Benzer biçimde örneğin biz bir yazıyı hem sınıflandırmak istediğimizi hem de yazının diline bakılarak onun hangi tarihte yazıldığını belirlemek istediğimizi düşünelim. Bu örnekte de iki çıktı fakat tek bir girdi vardır. Bu iki çıktı için iki ayrı model oluşturulabilir fakat tek bir modelle de bu işlem yapılabilir. Tabii girdilerin de birden fazla olması durumuyla karşılaşılabilirmektedir. Örneğin birinci girdi olarak yazının kendisini, ikinci girdi olarak da yazının yazarına ilişkin bilgileri kullanabiliyoruz. Bu iki girdi kümelerinden hareketle de yazıyı sınıflandırmak isteyebiliriz. Ya da örneğin bir evin fiyatını belirleyebilmek için evin büyülüğü, merkeze uzaklıği gibi birtakım bilgilerin yanı sıra evin fotoğraflarından da faydalanan makine öğrenmesi isteyebiliriz. Bu iki tür girdi birbirlerinden farklıdır ve bu iki tür bilgi üzerinde uygulanacak işlemler de birbirinden farklı olacaktır. Bu tür çok girdili modelleri tek girdili hale getirmek için bazen verilerin birleştirilmesi yoluna gidilebilmektedir. Ancak son örnekte olduğu gibi pek çok modelde farklı veriler girdi katmanının farklılığından dolayı tek bir girdi olarak birleştirilememektedir. İşte Keras'ın Sequential modeli böyle çoklu girdi ve çoklu çıktı üzerinde işlem yapamamaktadır. Bu işlemlerin yapılabilmesi için Sequential model yerine modelin fonksiyonel biçimde oluşturulması gerekmektedir.

Keras'ta aslında Dense gibi LSTM gibi katman sınıflarının fonksiyon çağrıma operatör metotları (`__call__` metotları) yazılmıştır. Bu metot çağrıma metotları katmanın girdisini oluşturmak ve katmanları birbirlerine bağlamak için kullanılmaktadır. Örneğin aşağıdaki gibi bir kod parçası olsun:

```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense

inp = Input(shape=(784, ), name='Input')
a = Dense(128, activation='relu', name='Dense-1')(inp)
b = Dense(128, activation='relu', name='Dense-2')(a)
output = Dense(10, activation='softmax', name='Output')(b)
model = Model(inputs=inp, outputs=output, name='Model')

model.summary()
```

Buradan elde edilen özet bilgiler şöyledir:

Model: "Model"

Layer (type)	Output Shape	Param #
Input (InputLayer)	[(None, 784)]	0
Dense-1 (Dense)	(None, 128)	100480
Dense-2 (Dense)	(None, 128)	16512
Output (Dense)	(None, 10)	1290

Total params: 118,282
 Trainable params: 118,282
 Non-trainable params: 0

Burada Dense fonksiyonu Dense türünden bir sınıf nesnesi yaratmaktadır. Bu sınıf nesnesi ile fonksiyon çağrıma operatörü kullanıldığından aslında bir önceki katmanın girdileri bu katmana bağlanmış olur. Dolayısıyla yukarıdaki kod parçasında Input katmanı aslında birinci Dense katmanına girdi yapılmıştır. Diğer Dense katmanları da diğerlerine girdi yapılmıştır. Toplamda bu kod parçası Sequential sınıfının yaptığı şeyi yapmaktadır. Başka bir deyişle bu işlemin Sequential eşdeğeri şöyledir:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential(name='Sequential')
model.add(Dense(128, input_dim=784, activation='relu', name='Dense-1'))
```

```
model.add(Dense(128, activation='relu', name='Dense-2'))
model.add(Dense(10, activation='softmax', name='Output'))
model.summary()
```

Buradan elde edilen özet bilgiler de şöyledir:

Model: "Sequential"

Layer (type)	Output Shape	Param #
=====		
Dense-1 (Dense)	(None, 128)	100480
Dense-2 (Dense)	(None, 128)	16512
Output (Dense)	(None, 10)	1290
=====		
Total params: 118,282		
Trainable params: 118,282		
Non-trainable params: 0		

Her ne kadar yukarıdaki iki kod parçası işlevsel bakımdan eşdeğerse de biz birinci kod parçasında artık Sequential sınıfından kurtulmuş olduk. Böylece bu yöntemle Sequential sınıfı ile yapamadığımız bazı şeyleri de yapar hale gelebileceğiz.

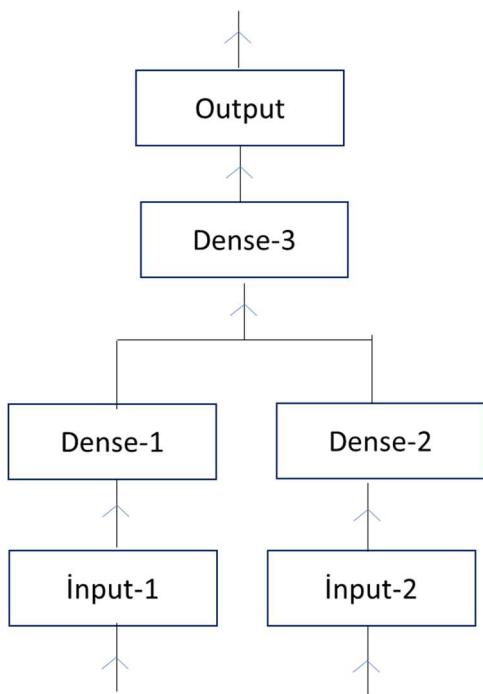
Sinir ağı modelini fonksiyonel olarak oluştururken girdileri oluşturmak için Input isimli bir katmanın kullanıldığına dikkat ediniz. Bu katmanın shape isimli parametresi girdinin boyutlarını belirtmektedir. Bu parametre bir demet biçiminde girilmelidir.

Fonksiyonel modelin kendisi Sequential sınıfı ile değişim Model isimli sınıf ile oluşturulmaktadır. Model sınıfının inputs parametresi modelin girdilerini outputs parametresi ise modelin çıktılarını belirtir. Biz yukarıdaki örnekte Model nesnesini şöyle yarattık:

```
model = Model(inputs=inp, outputs=output, name='Model')
```

Burada girdi ve çıktı katmanları bir tanedir. Eğer girdi ve çıktı katmanları birden fazla ise inputs ve outputs parametreleri demet ya da liste olarak girilir.

Fonksiyonel model sayesinde biz modeli graf biçiminde oluşturabiliriz. Örneğin:



Buradaki modelde Input-1 ve Input-2 isimli iki girdi katmanı bulunmaktadır. Bu katmanlardan ayrı ayrı Dense-1 ve Dense-2 katmanlarına bağlanmış, daha sonra da Dense-1 ve Dense-2 katmanlarının çıktıları birleştirilerek Dense-3 katmanına, Dense-3 katmanı da Output katmanına bağlanmıştır. Burada Dense-1 ve Dense-2 katmanlarının birleştirildiğini görüyorsunuz. Bu birleştirme işlemi tensorflow.keras.layers modülündeki concatenate fonksiyonuyla yapılmaktadır. concatenate fonksiyonu bizden birleştirilecek katmanların çıktılarını (ya da numpy dizilerini) bir demet ya da liste biçiminde parametre olarak alır ve bunları birleştirerek birleştirilmiş bir çıktı haline getirir. Fonksiyonun axis parametresi hangi eksene göre birleştirme yapılacağını belirtmektedir. Birleştirme işleminde birleştirilen eksenin dışındaki tüm eksenlerin uzunlıklarının aynı olması gereklidir. Örneğin:

```

from tensorflow.keras.layers import concatenate
import numpy as np

x = np.arange(20).reshape(4, 5)
y = np.arange(20).reshape(4, 5)
concatenated = concatenate([x, y], axis=0)

print(concatenated)

```

Burada birleştirme ilk eksene göre yapılacaktır. Şu sonuç elde edilmiştir:

```

tf.Tensor(
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]
 [ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]], shape=(8, 5), dtype=int32)

```

Örneğin:

```
concatenated = concatenate([x, y], axis=1)
```

Burada birleştirme 1 numaralı eksen üzerinde yapılmıştır. Yani sütunlar birleştirilmektedir. Şu sonuç elde edilmiştir:

```

tf.Tensor(
[[ 0  1  2  3  4  0  1  2  3  4]

```

```
[ 5  6  7  8  9  5  6  7  8  9]
[10 11 12 13 14 10 11 12 13 14]
[15 16 17 18 19 15 16 17 18 19]], shape=(4, 10), dtype=int32)
```

axis parametresinin default değeri 1'dir.

Şimdi bir metin ve o metne dayalı bir soruyu içeren ve sorunun Doğru ya da Yanlış biçiminde yanıtlandığı temsili çok girişli bir örnek verelim. Modelimizi fonksiyonel olarak şöyle kurabiliyoruz:

```
VOCAB_SIZE = 30000
TEXT_SIZE = 300
QUESTION_SIZE = 100

from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Embedding, LSTM, concatenate, Dense, Dropout

inp1 = Input(shape=(TEXT_SIZE,), name='Input-Text')
inp2 = Input(shape=(QUESTION_SIZE,), name='Input-Question')

x = Embedding(VOCAB_SIZE, 64, name='Embedding-Text')(inp1)
x = LSTM(64, activation='tanh', name='LSTM-Text')(x)
x = Dropout(0.2, name='Dropout-1')(x)

y = Embedding(VOCAB_SIZE, 64, name='Embedding-Question')(inp2)
y = LSTM(32, activation='tanh', name='LSTM-Question')(y)
y = Dropout(0.2, name='Dropout-2')(y)

z = concatenate([x, y])
z = Dense(128, activation='relu', name='Dense-1')(z)
z = Dropout(0.2, name='Dropout-3')(z)
z = Dense(64, activation='relu', name='Dense-2')(z)
z = Dropout(0.2, name='Dropout-4')(z)
z = Dense(1, activation='sigmoid', name='Output')(z)

model = Model(inputs=[inp1, inp2], outputs=z)
model.summary()
```

Burada metinler 300 karakterden sorular da 100 karakterden oluşmaktadır. Modelin iki ayrı girdisi vardır ve her girdiye word embedding uygulanmıştır. Sonra bu girdiler concatenate fonksiyonuyla birleştirilmiş ve Dense katmanlarla devam edilmiştir. Buradan elde edilen özet model bilgileri şöyledir:

Model: "model"

Layer (type)	Output Shape	Param #	Connected to
Input-Text (InputLayer)	[(None, 300)]	0	
Input-Question (InputLayer)	[(None, 100)]	0	
Embedding-Text (Embedding)	(None, 300, 64)	1920000	Input-Text[0][0]
Embedding-Question (Embedding)	(None, 100, 64)	1920000	Input-Question[0][0]
LSTM-Text (LSTM)	(None, 64)	33024	Embedding-Text[0][0]
LSTM-Question (LSTM)	(None, 32)	12416	Embedding-Question[0][0]
Dropout-1 (Dropout)	(None, 64)	0	LSTM-Text[0][0]
Dropout-2 (Dropout)	(None, 32)	0	LSTM-Question[0][0]
concatenate (Concatenate)	(None, 96)	0	Dropout-1[0][0] Dropout-2[0][0]

Dense-1 (Dense)	(None, 128)	12416	concatenate[0][0]
Dropout-3 (Dropout)	(None, 128)	0	Dense-1[0][0]
Dense-2 (Dense)	(None, 64)	8256	Dropout-3[0][0]
Dropout-4 (Dropout)	(None, 64)	0	Dense-2[0][0]
Output (Dense)	(None, 1)	65	Dropout-4[0][0]
=====			
Total params:	3,906,177		
Trainable params:	3,906,177		
Non-trainable params:	0		

Çok girişli modelleri eğitirken x verilerinin de bir demet ya da liste biçiminde verilmesi gerekmektedir. Biz burada girdileri rastgele değerlerle üreteceğiz. Çünkü amacımız belli bir problemi çözmekten ziyade çok girişli bir modelin yapısına ilişkin açıklamalar yapmak olacak:

```
import numpy as np

def generate_random_inputs(n):
    x1 = np.random.randint(0, VOCAB_SIZE, (n, TEXT_SIZE))
    x2 = np.random.randint(0, VOCAB_SIZE, (n, QUESTION_SIZE))
    y = np.random.randint(0, 2, n)

    return x1, x2, y

model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['binary_accuracy'])

training_dataset_text_x, training_dataset_question_x, training_dataset_y
=generate_random_inputs(1000)

hist = model.fit([training_dataset_text_x, training_dataset_question_x], training_dataset_y,
validation_split=0.2, epochs=5)
```

Burada fit metodunun x girdilerinin iki tane olduğuna dikkat ediniz. Bu girdiler bir demet ya da liste biçiminde girilebilmektedir. Tabii test işlemi de yine iki girdi kullanılarak yapılacaktır:

```
test_dataset_text_x, test_dataset_question_x, test_dataset_y = generate_random_inputs(100)

eval_result = model.evaluate([test_dataset_text_x, test_dataset_question_x], test_dataset_y)
for i in range(len(eval_result)):
    print(f'{model.metrics_names[i]} --> {eval_result[i]}')
```

Kestirim işlemini de tabii metin ve soru olmak üzere girdi ile yapmalıyız:

```
predict_text = np.random.randint(0, VOCAB_SIZE, TEXT_SIZE)
predict_question = np.random.randint(0, VOCAB_SIZE, QUESTION_SIZE)

predict_result = model.predict([predict_text.reshape(1, -1), predict_question.reshape(1, -1)])
if predict_result[0, 0] < 0.5:
    print('YANLIŞ')
else:
    print('DOĞRU')
```

Bir modelin birden fazla çıktısı da olabilir. Bu durumda çıktılar için ayrı Dense katmanları oluşturulur. Sonra bu katmanlar fit işleminde ayrı ayrı belirtilir. Buradaki önemli bir nokta üç farklı çıkışın skalalarının farklı olabilmesidir. Elde edilen modelin ağırlıklarını güncellenirken kullanılan optimizasyon algoritmaları tek bir loss değerine bakmaktadır. O halde üretilecek çıktıların tek bir loss değeri elde edilmek üzere birleştirilmesi gerekmektedir. Keras bu işlemi de arka planda otomatik olarak yapar.

TENSORFLOW KÜTÜPHANESİNİN KULLANIMI

TensorFlow Google tarafından 2015 yılında yapay zeka ve makine öğrenmesi konuları için tasarlanmış bir kütüphanedir. Daha önce de belirttiğimiz gibi Keras backend olarak Tensorflow ve diğer kütüphaneleri de kullanabilen yüksek seviyeli bir kütüphane durumundayken Tensorflow 2 ile birlikte artık Tensorflow kütüphanesinin bir parçası haline getirilmiştir. Kütüphaneye ismini veren Tensor matematik ve makine öğrenmesinde matrisel bir kavramı temsil etmektedir. İsimdeki Flow ise sözcüğü işlemlerin akışsal bir biçimde yürütüldüğünü anlatmaktadır. TensorFlow yalnızca Python'dan değil pek çok programlama dilinden de kullanılabilir. Zaten kütüphanenin önemli bir bölümü C++ Programlama Dilinde yazılmıştır.

TensorFlow kütüphanesinin Python için ana API dokümantasyonu aşağıdaki adreste bulunmaktadır:

https://www.tensorflow.org/api_docs/python

TensorFlow kütüphanesini Python'da kullanırken pek çokları gibi biz de import işlemini aşağıdaki gibi yapacağız:

```
import tensorflow as tf
```

Bir süre önceye kadar kursumuzda Tensorflow kütüphanesinin 1'li versiyonlarını anlatıyorduk. Sonra kütüphanenin 2'li versiyonları çıktıığında her ikisini de anlatmaya başladık. Ancak bir süredir artık kütüphanenin yalnızca 2'li versiyonlarını anlatıyoruz.

TensorFlow kütüphanesi 2'li versiyonlarla birlikte "eager execution" denilen pratik bir özelliğe sahip olmuştur. Bu özellik nedeniyle artık Tensorflow 1 için geçerli olan bazı özellikler Tensorflow 2 için geçerli olmamaktadır. Yani Tensorflow 2 ile birlikte kütüphanenin geriye doğru uyumluluğu (backward compatibility) bozulmuş durumdadır. Ancak TensorFlow ekibi bu uyumsuzluğu pratik bir biçimde gidermek için ayrı bir paket de düzenlemiştir. Eğer Tensorflow 1 ile çalışmak istiyorsanız fakat makinenizde TensorFlow'un 2'li versiyonları yüklü ise geriye doğru uyumu koruyabilmek için aşağıdaki bildirimleri yapmalısınız:

```
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
```

Aşağıdaki dokümanda TensorFlow'un 1'li versiyonları ile 2'li versiyonları arasındaki farklılıklar resmi olarak dokümante edilmiştir:

<https://www.tensorflow.org/guide/migrate>

Yukarıda da belirttiğimiz gibi biz artık kursumuzda yalnızca Tensorflow'un 2'li versiyonlarını ele alacağımız.

TensorFlow ile Numpy Kütüphaneleri Arasındaki Benzerlikler ve Farklılıklar

Tensorflow kütüphanesinin 1'li versiyonları metaprogramlama modeline dayandırılmıştı. Bu versiyonlarda önce kod akışını bir graf biçiminde oluşturup sonra onu çalıştırıyordu. Dolayısıyla Tensorflow'un 1'li versiyonları programlama modeli Numpy kütüphanesindeki modelden oldukça farklıydı. Ancak TensorFlow'da 2'li versiyonlarla birlikte bu metaprogramlama modeli karmaşık olduğu gereçesiyle kaldırılmıştır. Bu nedenle TensorFlow 2'deki model klasik numpy modeline daha çok benzer hale gelmiştir.

Hem Numpy hem de TensorFlow vektörel işlemler yapabilen kütüphanelerdir. Yani her iki kütüphanede de biz dizileri tek hamlede hiç döngü kullanmadan işlemlere sokabilmekteyiz. Bu bağlamda TensorFlow'daki Tensor nesnelerini Numpy'daki ndarray nesnelerine benzetilebiliriz. Her iki kütüphane de işlevsel bakımından birbirlerine benzese de aralarında belirgin farklılıklar davardır. Bu farklılıkları tek tek ele almak istiyoruz:

- TensorFlow makine öğrenmesi için kullanılan bir kütüphanedir. Halbuki Numpy nümerik işlemler için kullanılan genel amaçlı bir kütüphanedir. Bu nedenle Tensorflow yapay sinir ağları, görüntü işleme gibi alanlarda hazır modüllere de sahiptir.
- Numpy Python için yazılmış bir kütüphanedir. Oysa TensorFlow Python dışında başka programlama dillerinden de kullanılabilmektedir.
- TensorFlow gradient tabanlı optimizasyon işlemlerini otomatik olarak bünyesinde bulundurmaktadır. Gradient tabanlı optimizasyonun makine öğrenmesinde önemli bir yeri vardır.
- TensorFlow paralel programlama yeteneğine sahiptir. Paralel programlama uygularken CPU çekirdeklerini ve GPU'yu da kullanabilmektedir. Oysa Numpy paralel programlama yeteneğine sahip değildir.

Tensör Nesnelerinin Yaratılması

TensorFlow kütüphanesindeki en önemli kavram "tensor" kavramıdır. Bir tensör matrisel bir bilgidir. Tensorflow için tensörler Numpy'daki ndarray nesneleri ile benzer anlamdadır. TensorFlow'da da tıpkı Numpy'da olduğu gibi ensörlerin birer dtype türleri ve shape özellikleri vardır. Tensörler read only ya da read/write olabilmektedir. TensorFlow 1'de tensörlere isimler ilişirilmektedir. Graf oluştururken bu isimlerden faydalanaabiliyordu. Tensorflow 2'de tensörlerin yine ism olsa da "eager execution" özelliği nedeniyle artık isimlerin bir önemi kalmamıştır.

Bir tensor yaratmanın en çok kullanılan yöntemlerinden biri tf.constant fonksiyonunu kullanmaktadır. Fonksiyonun parametrik yapısı şöyledir:

```
tf.constant(
    value,
    dtype=None,
    shape=None,
    name='Const'
)
```

Fonksiyonun birinci parametresi yaratılacak tensörün bilgilerini, ikinci parametresi de dtype türünü belirtmektedir. dtype türü belirtilmezse tam sayı değerler için dtype türü tf.int32, noktalı sayılar için float32'dir (Numpy'da noktalı sayılar için default dtype türünün np.float64 olduğunu anımsayınız). dtype türü yine Numpy'da olduğu gibi string biçiminde de belirtilebilmektedir. shape parametresi de tensörün boyut bilgisini belirtir. name parametrelerinin Tensorflow 2 için önemi yoktur. Örneğin:

```
t = tf.constant([1, 2, 3, 4, 5, 6], dtype=tf.float32, shape=(2, 3))
print(t)
```

Şu çıktı elde edilmiştir:

```
tf.Tensor(
[[1. 2. 3.]
 [4. 5. 6.]], shape=(2, 3), dtype=float32)
```

Burada value parametresi herhangi bir dolaşılabilir nesne olarak girilebilmektedir. shape parametresi girilmezse value parametresindeki şekil tensörün boyutu olarak kabul edilmektedir. constant fonksiyonu ile yaratılan tensörün elemanları değiştirilemez. constant fonksiyonuyla yaratılan tensör read-only durumdadır. Biz onun herhangi bir elemanını değiştirememiz: Örneğin:

```
In [16]: t[2] = 10
Traceback (most recent call last):

  File "C:\Users\aslan\AppData\Local\Temp\ipykernel_12508/2011961731.py",
line 1, in <module>
    t[2] = 10

TypeError: 'tensorflow.python.framework.ops.EagerTensor' object does not
support item assignment
```

Fonksiyonun value parametresi tek bir değer olarak da girilebilir. Bu durumda shape ile belirtilen boyuttaki elemanların tamamı o değerle doldurulur. Örneğin:

```
In [15]: tf.constant(1, shape=(3, 3))
Out[15]:
<tf.Tensor: shape=(3, 3), dtype=int32, numpy=
array([[1, 1, 1],
       [1, 1, 1],
       [1, 1, 1]])>
```

Tensör yaratmak için diğer bir seçenek de tf.convert_to_tensor fonksiyonunu kullanmaktadır. Fonksiyonun parametrik yapısı şöyledir:

```
tf.convert_to_tensor(
    value,
    dtype=None,
    dtype_hint=None,
    name=None
)
```

Bu fonksiyon da read-only bir tensör yaratmaktadır. Örneğin:

```
t = tf.convert_to_tensor([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
```

Fonksiyonun bir shape parametresinin olmadığına dikkat ediniz.

TensorFlow'da da tipki Numpy'da olduğu gibi zeros, ones ve fill fonksiyonları vardır:

```
tf.zeros(
    shape,
    dtype=tf.dtypes.float32,
    name=None
)

tf.ones(
    shape,
    dtype=tf.dtypes.float32,
    name=None
)

tf.fill(
    dims,
    value,
    name=None
)
```

Örneğin:

```
In [21]: tf.zeros((3, 2))
Out[21]:
<tf.Tensor: shape=(3, 2), dtype=float32, numpy=
array([[0., 0.],
       [0., 0.],
       [0., 0.]], dtype=float32)>
```

Yine TensorFlow'da Numpy'da olduğu gibi zeros_like ve ones_like fonksiyonları da bulunmaktadır.

```
In [22]: tf.ones((3, 2))
Out[22]:
<tf.Tensor: shape=(3, 2), dtype=float32, numpy=
array([[1., 1.],
       [1., 1.],
       [1., 1.]], dtype=float32)>
```

```
In [23]: tf.fill((3, 2), 10)
Out[23]:
<tf.Tensor: shape=(3, 2), dtype=int32, numpy=
array([[10, 10],
       [10, 10],
       [10, 10]])>
```

tf.range fonksiyonu Python'un range fonksiyonu gibi kullanılmaktadır. Bize dizilimdeki değerleri içeren bir tensör nesnesi verir. Fonksiyonun parametrik yapısı şöyledir:

```
tf.range(
    start,
    limit,
    delta=1,
    dtype=None,
    name='range'
)
```

Örneğin:

```
In [33]: tf.range(10)
Out[33]: <tf.Tensor: shape=(10,), dtype=int32, numpy=array([0, 1, 2, 3,
4, 5, 6, 7, 8, 9])>
```

Örneğin:

```
In [34]: tf.range(10.5, 20.5, 0.5)
Out[34]:
<tf.Tensor: shape=(20,), dtype=float32, numpy=
array([10.5, 11. , 11.5, 12. , 12.5, 13. , 13.5, 14. , 14.5, 15. ,
       16. , 16.5, 17. , 17.5, 18. , 18.5, 19. , 19.5, 20. ],
       dtype=float32)>
```

Fonksiyon parametrelerinin int değil float türden olabildiğine dikkat ediniz. Yine TensorFlow'da da Numpy'da olduğu gibi bir linspace fonksiyonu vardır:

```
tf.linspace(
    start,
    stop,
    num,
    name=None,
    axis=0
)
```

Örneğin:

```
In [6]: tf.linspace(5, 10, 10)
Out[6]:
<tf.Tensor: shape=(10,), dtype=float64, numpy=
array([ 5.          ,  5.55555556,  6.11111111,  6.66666667,  7.22222222,
       7.77777778,  8.33333333,  8.88888889,  9.44444444, 10.        ])>
```

Bir tensör nesnesinin dtype örnek özniteliği nesnenin dtype türünü elde etmek için, shape örnek özniteliği ise nesnesinin boyut bilgisini elde etmek için kullanılmaktadır. Örneğin

```
In [24]: t = tf.constant([1, 2, 3, 4, 5, 6], shape=(2, 3),
dtype=tf.float32)
```

```
In [25]: t.dtype
Out[25]: tf.float32
```

```
In [26]: t.shape
Out[26]: TensorShape([2, 3])
```

Tensör nesnesinin boyutsal özellikleri shape örnek özniteliğinin yanı sıra eşdeğer olarak get_shape metoduyla da elde edilebilmektedir. Aslında tensör nesnelerinin boyutsal özellikleri tf.shape fonksiyonuyla da elde edilebilmektedir. Örneğin:

```
In [18]: tf.shape(t)
Out[18]: <tf.Tensor: shape=(1,), dtype=int32, numpy=array([10])>
```

Tensor sınıfının shape örnek özniteliğinin boyutsal özellikleri TensorShape nesnesi biçiminde, ancak tf.shape fonksiyonunun bir Tensor nesnesi biçiminde verdiğine dikkat ediniz. TensorShape nesnesi bir Tensor nesnesi değildir. Tensor nesnelerinin boyutsal özellikleri de tf.rank fonksiyonuyla elde edilebilmektedir. Bu fonksiyonun bir örnek özniteliği ya da metot karşılığı yoktur. Örneğin:

```
In [24]: t = tf.constant([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
```

```
In [25]: tf.rank(t)
Out[25]: <tf.Tensor: shape=(), dtype=int32, numpy=3>
```

Skaler tensör nesnelerinin boyutsal özelliklerinin boş olduğuna dikkat ediniz. Örneğin:

```
In [46]: a = tf.constant(10)
```

```
In [47]: b = tf.constant([10, 20, 30])
```

```
In [48]: a.shape
Out[48]: TensorShape([])
```

```
In [49]: b.shape
Out[49]: TensorShape([3])
```

Bir tensör nesnesinin boyutsal özelliklerini değiştirebilmek için tf.reshape fonksiyonu kullanılmaktadır. Bu fonksiyonun bir metot karşılığı yoktur. Örneğin:

```
In [106]: t = tf.constant([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
```

```
In [107]: k = tf.reshape(t, (9, ))
```

```
In [108]: k
```

```
Out[108]: <tf.Tensor: shape=(9,), dtype=int32, numpy=array([1, 2, 3, 4, 5, 6, 7, 8, 9])>
```

Tıpkı Numpy kütüphanesinde olduğu gibi belli bir eksen değeri -1 girilirse bu durum "diğer eksenlerin durumuna göre bu eksenin değerini belirle" anlamına gelmektedir.

Bir tensör nesnesinin toplam eleman sayısı tf.size fonksiyonuyla elde edilebilir. Bu fonksiyonun bir örnek özniteliği ya da metot karşılığı yoktur. Örneğin:

```
In [78]: t = tf.Variable([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
```

```
In [79]: tf.size(t)
```

```
Out[79]: <tf.Tensor: shape=(), dtype=int32, numpy=9>
```

Tensör nesnelerinin elemanlarına köşeli parantez operatörleriyle erişilebilir. Dilimleme işlemi yine Numpy dizilerinde olduğu gibidir. Örneğin:

```
In [28]: t = tf.constant([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
```

```
In [29]: t[2]
```

```
Out[29]: <tf.Tensor: shape=(4,), dtype=int32, numpy=array([ 9, 10, 11, 12])>
```

```
In [30]: t[0:3, 2:4]
```

```
Out[30]:
```

```
<tf.Tensor: shape=(3, 2), dtype=int32, numpy=
array([[ 3,  4],
       [ 7,  8],
       [11, 12]])>
```

```
In [31]: t[:-1, :-2]
```

```
Out[31]:
```

```
<tf.Tensor: shape=(2, 2), dtype=int32, numpy=
array([[1, 2],
       [5, 6]])>
```

Bir tensör nesnesi içerisindeki değer numpy dizisi olarak numpy isimli metot ile elde edilebilir. Örneğin:

```
In [92]: t = tf.constant([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]])
```

```
In [93]: t.numpy()
```

```
Out[93]:
```

```
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [10, 11, 12]])
```

Bir tensör nesnesi tf.Variable sınıfıyla da oluşturulabilir. tf.Variable sınıfıyla oluşturulan tensörlerin tuttuğu değerler değiştirilebilmektedir. tf.Variable sınıfının `__init__` metodunun parametrik yapısı şöyledir:

```
tf.Variable(
    initial_value=None,
    trainable=None,
    validate_shape=True,
```

```

caching_device=None,
name=None,
variable_def=None,
dtype=None,
import_scope=None,
constraint=None,
synchronization=tf.VariableSynchronization.AUTO,
aggregation=tf.compat.v1.VariableAggregation.NONE,
shape=None
)

```

Variable nesnesi yaratılırken yine ilkdeğer verilmek zorundadır. Örneğin:

```
In [35]: t = tf.Variable([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
```

```
In [36]: t
Out[36]:
<tf.Variable 'Variable:0' shape=(3, 3) dtype=int32, numpy=
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])>
```

Variable nesneleri de birer tensör olduğu için tensörler üzerindeki işlemler bunlar üzerinde de uygulanabilmektedir. Bir Variable nesnesinin içerisindeki değer assign metodu ile değiştirilebilir Örneğin:

```
In [65]: t = tf.Variable([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
```

```
In [66]: t
Out[66]:
<tf.Variable 'Variable:0' shape=(3, 3) dtype=int32, numpy=
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])>
```

```
In [67]: t.assign([[10, 20, 30], [40, 50, 60], [70, 80, 90]])
Out[67]:
<tf.Variable 'UnreadVariable' shape=(3, 3) dtype=int32, numpy=
array([[10, 20, 30],
       [40, 50, 60],
       [70, 80, 90]])>
```

```
In [68]: t
Out[68]:
<tf.Variable 'Variable:0' shape=(3, 3) dtype=int32, numpy=
array([[10, 20, 30],
       [40, 50, 60],
       [70, 80, 90]])>
```

Bir Variable nesnesinin belli bir elemanına köşeli parantez operatörüyle değer atayamayız. Örneğin:

```
In [73]: t[1, 2] = 10
Traceback (most recent call last):
  File "C:\Users\aslan\AppData\Local\Temp\ipykernel_19348/3177698712.py",
line 1, in <module>
    t[1, 2] = 10
```

TypeError: 'ResourceVariable' object does not support item assignment

Rastgele değerlerden tensör nesneleri oluşturmak için tf.random modülündeki fonksiyonlar kullanılmaktadır. tf.random.uniform fonksiyonu düzgün dağılmış rastgele sayılarından oluşan tensör nesneleri oluşturmaktadır. Fonksiyonun parametrik yapısı şöyledir:

```
tf.random.uniform(
    shape,
    minval=0,
    maxval=None,
    dtype=tf.dtypes.float32,
    seed=None,
    name=None
)
```

Burada minval değeri aralığa dahil, maxval değeri aralığa dahil değildir. Örneğin:

```
In [81]: tf.random.uniform((3, 3), 0, 10, dtype=tf.int32)
Out[81]:
<tf.Tensor: shape=(3, 3), dtype=int32, numpy=
array([[6, 9, 3],
       [9, 2, 3],
       [0, 8, 4]])>
```

tf.random.normal fonksiyonu ise normal dağılıma uygun rastgele sayılar üretmektedir. Fonksiyonun parametrik yapısı şöyledir:

```
tf.random.normal(
    shape,
    mean=0.0,
    stddev=1.0,
    dtype=tf.dtypes.float32,
    seed=None,
    name=None
)
```

Örneğin:

```
In [84]: tf.random.normal((5, 5))
Out[84]:
<tf.Tensor: shape=(5, 5), dtype=float32, numpy=
array([[ 0.6709525 , -0.5455337 , -0.09969767, -0.5458125 , -0.72455513],
       [ 0.7689879 , -0.23801427,  0.46185434,  0.88133913, -0.39394924],
       [ 0.49497288,  0.39744157,  0.8921115 , -0.57044065, -0.90798306],
       [ 0.0737021 ,  0.59513694, -1.9811838 ,  0.3674018 , -1.4103378 ],
       [-1.830832 , -0.0887295 , -1.4519913 ,  0.62525374,  0.48967224]],
```

tf.random.shuffle fonksiyonu in-place karıştırma işlemi yapmaktadır. Fonksiyonun parametrik yapısı şöyledir:

```
tf.random.shuffle(
    value,
```

```
    seed=None,  
    name=None  
)
```

Karıştırma her zaman ilk eksene göre yapılmaktadır. Örneğin:

Örneğin:

```
In [89]: t = tf.constant([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]])
```

```
In [90]: t
```

```
Out[90]:  
<tf.Tensor: shape=(4, 3), dtype=int32, numpy=  
array([[ 1,  2,  3],  
       [ 4,  5,  6],  
       [ 7,  8,  9],  
       [10, 11, 12]])>
```

```
In [91]: tf.random.shuffle(t)
```

```
Out[91]:  
<tf.Tensor: shape=(4, 3), dtype=int32, numpy=  
array([[ 1,  2,  3],  
       [ 7,  8,  9],  
       [10, 11, 12],  
       [ 4,  5,  6]])>
```

tf.random modülünde daha pek çok fonksiyon vardır. Bunlar hakkında bilgileri Tensorflow dokümanlarından elde edebilirsiniz.

Tensör Nesneleri Üzerinde İşlemler

Tıpkı Numpy kütüphanesinde olduğu gibi TensorFlow kütüphanesinde de tensör nesneleri üzerinde vektörel işlemler yapılabilmektedir. Tensor sınıfının operatör metodları iki tensor nesnesi üzerinde işlemler yapılmasına olanak sağlamaktadır. Örneğin:

```
import tensorflow as tf  
  
a = tf.constant([1, 2, 3], dtype=tf.float32)  
b = tf.constant([1, 2, 3], dtype=tf.float32)  
  
c = a + b  
print(c)
```

Şöyledir bir çıktı elde edilmiştir:

```
tf.Tensor([2. 4. 6.], shape=(3,), dtype=float32)
```

Vektörel işlemler Numpy kütüphanesinde olduğu gibidir. * operatörü karşılıklı elemanları çarpmaktadır. Matrisel çarpım için tf.matmul fonksiyonu kullanılır. Örneğin:

```
import tensorflow as tf  
  
a = tf.constant([[1, 2, 3], [4, 5, 6], [7, 8, 9]], dtype=tf.float32)  
b = tf.constant([[1, 2, 3], [4, 5, 6], [7, 8, 9]], dtype=tf.float32)  
  
c = tf.matmul(a, b)  
print(c)
```

Şu çıktı elde edilmiştir:

```
tf.Tensor(  
[[ 30.  36.  42.]  
 [ 66.  81.  96.]  
[102. 126. 150.]], shape=(3, 3), dtype=float32)
```

Örneğin:

Numpy kütüphanesinde olduğu gibi Tensorflow kütüphanesinde de "broadcasting" özelliği vardır. Örneğin:

```
import tensorflow as tf  
  
a = tf.constant([[1, 2, 3]], dtype=tf.float32)  
  
b = tf.constant([[1, 2, 3], [4, 5, 6], [7, 8, 9]], dtype=tf.float32)  
  
c = a + b  
print(c)
```

Şu çıktı elde edilmiştir:

```
tf.Tensor(  
[[ 2.  4.  6.]  
 [ 5.  7.  9.]  
[ 8. 10. 12.]], shape=(3, 3), dtype=float32)
```

tf.math modülü içerisinde axis parametresi alarak matematiksel işlem yapan pek çok hazır fonksiyon bulunmaktadır. Bu fonksiyonlardaki axis parametresi Numpy fonksiyonlarındaki axis parametresi ile aynı anlamdadır. axis parametresi değişimin hangi eksen üzerinde yapılacağı (işlem sırasında hangi eksen değerler değiştirilerek işlemin yapılacağı) anlamına gelir. Tensörün satırları ya da sütunları üzerinde temel işlemler tf.math.reduce_sum, tf.math.reduce_mean, tf.math.reduce_max, tf.math.reduce_min gibi tf.math.reduce_xxx biçiminde isimlendirilmiş fonksiyonlarla yapılmaktadır. Örneğin:

```
In [128]: t = tf.constant([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11,  
12]])  
  
In [129]: tf.math.reduce_sum(t)  
Out[129]: <tf.Tensor: shape=(), dtype=int32, numpy=78>  
  
In [130]: tf.math.reduce_sum(t, axis=0)  
Out[130]: <tf.Tensor: shape=(3,), dtype=int32, numpy=array([22, 26,  
30])>  
  
In [131]: tf.math.reduce_sum(t, axis=1)  
Out[131]: <tf.Tensor: shape=(4,), dtype=int32, numpy=array([ 6, 15,  
24, 33])>
```

tf.math modülündeki reduce_xxx biçiminde isimlendirilmemiş fonksiyonlar axis parametresi almamaktadır. Örneğin:

```
In [142]: t = tf.constant([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]], dtype=tf.float32)
```

```
In [143]: tf.math.sqrt(t)
```

```
Out[143]:
```

```
<tf.Tensor: shape=(4, 3), dtype=float32, numpy=
array([[0.99999994, 1.4142134 , 1.7320508 ],
       [1.9999999 , 2.236068 , 2.4494896 ],
       [2.6457512 , 2.8284268 , 2.9999998 ],
       [3.1622777 , 3.3166249 , 3.4641016 ]], dtype=float32)>
```

tf.math modülündeki segment_xxx biçiminde isimlendirilmiş olan fonksiyonlar matristeki değerleri bölmelere (segment'lere) ayırarak o bölmeler üzerinde işlemler yaparlar. Bölmeler 0'dan başlayarak artan sırada tamsayı değerlerle ifade edilmektedir ve bölüm belirten bu değerlerin sıralı olması gerekmektedir. Örneğin:

```
In [147]: a = tf.constant([3, 5, 7, 9, 4, 2, 1, 6])
```

```
In [148]: tf.math.segment_sum(a, [0, 0, 1, 1, 1, 2, 2, 3])
```

```
Out[148]: <tf.Tensor: shape=(4,), dtype=int32, numpy=array([ 8, 20, 3, 6])>
```

Burada segment değerlerinin tensör'ün eleman sayısı ile aynı uzunlukta olduğuna dikkat ediniz. 0, 1, 2, 3 değerlerine karşı gelen elemanlar kendi aralarında toplama işlemine sokulmuştur. Bölümleme her zaman 0'dan itibaren yapılmaktadır. Örneğin:

```
In [152]: a = tf.constant([3, 5, 7, 9, 4, 2, 1, 6])
```

```
In [153]: tf.math.segment_sum(a, [1, 1, 1, 2, 2, 3, 4, 4])
```

```
Out[153]: <tf.Tensor: shape=(5,), dtype=int32, numpy=array([ 0, 15, 13, 2, 7])>
```

Çok boyutlu tenseörlerdeki segment numaraları her zaman son eksene ilişkindir. Örneğin:

```
In [177]: a = tf.constant([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]])
```

```
In [178]: tf.math.segment_sum(a, [0, 0, 1, 1])
```

```
Out[178]:
```

```
<tf.Tensor: shape=(2, 3), dtype=int32, numpy=
array([[ 5,  7,  9],
       [17, 19, 21]])>
```

tf.math modülündeki unsorted_segment_xxx isimli fonksiyonlarda bölüm numaraları sıralı olmak zorunda değildir. Bu fonksiyonlarda ayrıca bölüm sayısının da belirtilmesi gerekmektedir. Örneğin:

```
In [166]: a = tf.constant([3, 5, 7, 9, 4, 2, 1, 6])
```

```
In [167]: tf.math.unsorted_segment_sum(a, [0, 1, 0, 1, 2, 0, 1, 2], num_segments=3)
```

```
Out[167]: <tf.Tensor: shape=(3,), dtype=int32, numpy=array([12, 15, 10])>
```

Buradaki segment_num bölüm sayılarından fazla girilirse geri kalan elemanlar 0 olarak doldurulmaktadır. Örneğin:

```
In [174]: a = tf.constant([3, 5, 7, 9, 4, 2, 1, 6])
In [175]: tf.math.unsorted_segment_sum(a, [0, 1, 0, 1, 2, 0, 1, 2],
num_segments=5)
Out[175]: <tf.Tensor: shape=(5,), dtype=int32, numpy=array([12, 15,
10, 0, 0])>
```

tf.math.argmax ve tf.math.argmin fonksiyonları Numpy'da olduğu gibi en büyük ve en küçük elemanların kendilerini değil indekslerini bulmaktadır.

Tensorflow Veri Kümelerinin Kullanımı

Nasıl Keras içerisinde hazır çeşitli veri kümeleri varsa Tensorflow içerisinde de hazır bazı veri kümeleri bulunmaktadır. Bu veri kümeleri ayrı bir paket olarak oluşturulmuştur. Tensorflow kurulumu sırasında bu paket kurulmamaktadır. Tensorflow veri kümelerini yüklemesini komut satırında pip kurulum programıyla şöyle indirip kurabilirsiniz:

```
pip install tensorflow-datasets
```

Paketi aşağıdaki gibi import ederek kullanabiliriz:

```
import tensorflow_datasets as tfds
```

Tüm veri kümeleri tensorflow_datasets.core.DatasetBuilder sınıfından türetilmiş sınıflar biçiminde oluşturulmuştur. Tüm veri kümelerinin isimleri liste biçiminde şöyle elde edilebilir:

```
all_datasets = tfds.list_builders()
print(all_datasets)
```

Veri kümeleri tensorflow_datasets.load fonksiyonuyla yüklenebilir. Örneğin:

```
datasets = tfds.load( name='horses_or_humans' , split=['train', 'test'])
```

split parametresi veri kümelerini 'train', 'test' ya da ['train', 'test'] biçiminde ayırmak için kullanılmaktadır. Default ayrılm %80-%20 biçimindedir. Ancak oran da verilebilmektedir. load fonksiyonundan elde edilen veri kümelerinin elemanlarına köşeli parantez operatörüyle erişilebilir.

with_info parametresi True geçilirse load fonksiyonunun geri dönüş değeri bir demet olmaktadır:

```
datasets, info = tfds.load( name='horses_or_humans' , split=['train', 'test'], with_info=True)
```

Bu biçimde elde edilen datasets nesnesi bir listedir. Listenin ilk elemanı "train" verilerini ikinci elemanı ise "test" verilerini tutmaktadır.

DENETİMSİZ ÖĞRENME (UNPERVISED LEARNING)

Kursumuzun giriş kısımlarında da belirttiğimiz gibi "makine öğrenmesi (machine learning)" temelde üç grup öğrenme yöntemlerinden oluşmaktadır:

- 1) Denetimli Öğrenme (supervised learning)
- 2) Denetimsiz Öğrenme (unsupervised learning)
- 3) Pekiştirmeli Öğrenme (Reinforcement learning)

Biz şimdide yapay sinir ağlarını kullanarak denetimli öğrenme (supervised learning) süreçlerini ele aldık. Kursumuzun bu bölümünde ise denetimsiz (unsupervised) öğrenme konuları üzerinde duracağız.

Denetimsiz öğrenmede bir eğitim süreci yoktur. Eğitim sürecinin olmaması bir eğitim veri kümесinin de olmayacağı anlamına gelir. Yani örneğin biz denetimli öğrenmede ağı eğitmek için `training_dataset_x` ve `training_dataset_y` veri kümelerini kullandık. Halbuki denetimsiz öğrenmede elimizde yalnızca `dataset_x` kümesi bulunmaktadır.

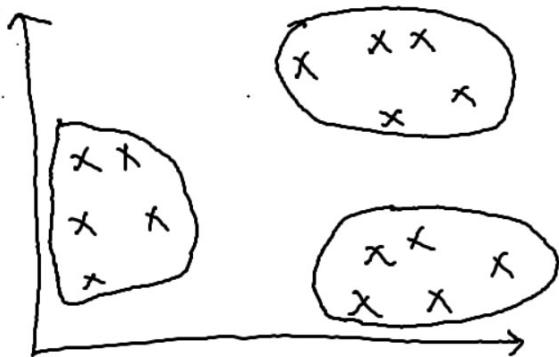
Eğitimli öğrenme ile eğitimsiz öğrenme arasındaki farkı MNIST örneği ile daha iyi açıklayabiliriz. Biz MNIST örneğinde 28x28 tane pixel'den oluşan resimlerin hangi rakamlara ilişkin olduğunu ağa girdi yaparak modeli eğitiyorduk. Yani biz denetimli öğrenmede aslında "bak bu 28x28'lik pixel'ler bu rakamı belirtiyor, bu 28x28'lik pixel'ler de bu rakamı belirtiyor" diyerek sistemin hangi 28x28'lik pixel'lerin hangi rakamı belirttiğini anlamasını sağlıyorduk. Oysa denetimsiz öğrenmede biz modele "bu 28x28'lik pixel'ler bazı bakımlardan birbirlerine benziyor. Birbirlerine benzeyen 28x28'lik pixel'leri grupta" diyoruz. Böylece MNIST için kullandığımız denetimsiz modeller rakamları bilmiyor ancak o rakamların birbirine benzettiğini fark ederek onları birbirlerinden ayırabiliyor.

Kümeleme (Clustering) İşlemleri

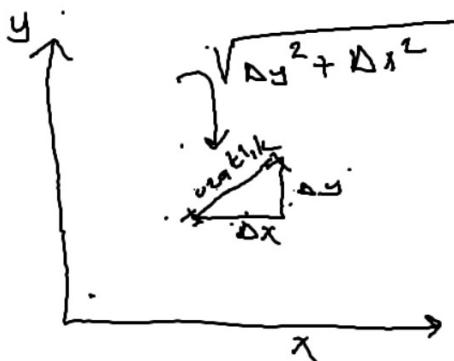
Denetimsiz öğrenme yaklaşımının en önemlilerinden biri kümeleme (clustering) işlemleridir. Nesnelerin ya da olguların özelliklerine bakılarak birbirlerine benzeyenlerin bir araya getirilmesi sürecine kümeleme denilmektedir. Makine öğrenmesinde sınıflandırma ile kümeleme süreçleri birbirine karıştırılabilir. Sınıflandırma bir olgunun daha önce belirlenmiş sınıflardan birine atanması süreci ile ilgilidir. Halbuki kümeleme olguların benzerliklerine göre bir araya getirilmesi süreci ile ilgilidir. Örneğin elimizde üç farklı meyveye ilişkin resimler bulunuyor olsun. Eğer biz bir resmin hangi meyveye ilişkin olduğunu anlamaya çalışıyorsak bu bir sınıflandırma işlemidir. Ancak biz bu resimleri benzerliklerine göre üç gruba ayırmaya çalışıyorsak bu bir kümeleme işlemidir. Yani sınıflandırmada sınıflar işin başında zaten bilinmektedir. Oysa kümelemede sınıflar kendisi oluşturulmaya çalışılmaktadır. Makine öğrenmesinde sınıflandırma işlemlerinin "eğitimli öğrenme (supervised learning)" yöntemlerine kümeleme işlemlerinin ise "eğitsiz öğrenme (unsupervised learning)" yöntemlerine ilişkin olduğuna dikkat ediniz.

Mademki kümeleme işlemleri birbirine benzeyenlerin bir araya getirilme sürecidir. O halde kümelemenin yapılabilmesi için olguların birbirine benzerliğinin belirlenmesi gereklidir. Benzemek fiili insan sezgisi ile ilgilidir. Makinelerin böyle sezgileri olmadığına göre makine öğrenmesinde benzerliklerin somut niceliklerle ifade edilmesi gereklidir. Makine öğrenmesinde benzerlikler için çeşitli ölçütler kullanılabilmektedir. Örneğin benzerlikler için en çok kullanılan ölçütlerden biri uzaklık ölçütüdür. Uzaklık ölçütüne göre biz birbirlerine yakın olan öğelerin birbirlerine benzediğini, birbirlerine uzak olan öğelerin ise birbirlerine benzemediğini iddia ederiz.

İki ögenin uzaklığını ölçmek için değişik yöntemler kullanılabilmektedir. En çok kullanılan uzaklık ölçme yöntemlerinden biri "Öklit Uzaklığı (Euclidean Distance)" denilen yöntemdir. Öklit uzaklığı n boyutlu uzayda iki nokta arasındaki uzaklığı belirtmektedir. Örneğin iki özelliği olan olguları Öklit uzaklığına göre görsel olarak kümelemeye çalışalım. Söz konusu bu iki özellik x ve y olsun. Bu durumda bu olgular kartezyen koordinat sisteminde birer nokta belirtecektir. Aşağıdaki şekele göz gezdiriniz:



Burada gözle baktığımızda noktaların Öklit uzaklıklarına göre kendi aralarında üç grup oluşturduğunu görüyoruz. Birbirlerine yakın olan noktalar aynı grup içerisinde bulunmaktadır. Bildiğiniz gibi kartezyen koordinat sisteminde Öklit uzaklığı şöyle hesaplanmaktadır:



Lineer cebirde aslında iki boyutlu düzlem ile n boyutlu uzay arasında işlemlerin uygulanma biçimini bakımından bir farklılık yoktur. Yani iki boyutlu düzlem için söz konusu olan her şey üç boyutlu, dört boyutlu ve genel olarak n boyutlu uzay için de geçerlidir. Örneğin üç boyutlu uzayda iki nokta arasındaki Öklit uzaklığı benzer biçimde aşağıdaki gibi hesaplanabilir:

$$\sqrt{\Delta x^2 + \Delta y^2 + \Delta z^2}$$

n boyutlu uzay için ise iki nokta arasındaki Öklit uzaklığı boyutlar X_i ile gösterilmek üzere hesaplanacaktır:

$$\sqrt{\sum_{i=1}^n \Delta x_i^2}$$

iki boyutlu düzlem ile n boyutlu uzay arasında genel işlemler bakımından bir farklılık olmadığı için konu anlatımları insan algısına yakınlık nedeniyle genellikle iki boyutlu düzlem üzerinde yapılmaktadır. Makine öğrenmesinde n boyutlu uzayı n tane sütuna sahip olan bir veri tablosu anlamına geldiğine dikkat ediniz.

Şimdi n boyutlu uzay için (yani n tane sütunu olan bir veri tablosu için) iki nokta arasındaki Öklit uzaklığını hesaplayan bir Python fonksiyonu yazalım:

```
import numpy as np

def euclidean_distance(a, b, axis = 0):
    return np.sqrt(np.sum((a - b) ** 2, axis=axis))
```

Örneğin 5 boyutlu uzayda iki nokta arasındaki uzaklığı bu fonksiyonla şöyle hesaplayabiliriz:

```
a = np.array([3, 4, 2, 6, 7])
b = np.array([1, 5, 9, 2, 4])

distance = euclidean_distance(a, b)
print(distance)
```

Buradan şöyle bir çıktı elde edilecektir:

8.888194417315589

Tabii fonksiyonun parametresi olan a ve b birer matris olarak da girilebilir. Bu durumda axis=1 parametresi ile fonksiyon karşılıklı satırları nokta olarak ele alacaktır:

```
a = np.array([[3, 4, 2, 6, 7], [3, 4, 6, 7, 3], [2, 6, 1, 5, 6]], dtype=np.float32)
b = np.array([[1, 5, 9, 2, 4], [3, 5, 7, 1, 2], [4, 5, 2, 1, 7]], dtype=np.float32)

distance = euclidean_distance(a, b, axis=1)
print(distance)
```

Buradan şöyle bir çıktı elde edilecektir:

[8.888194 6.244998 4.7958317]

Öklit uzaklığı için `scipy.spatial.distance` modülündeki `euclidean` fonksiyonu da kullanılabilir. Örneğin:

```
from scipy.spatial.distance import euclidean

a = np.array([3, 4, 2, 6, 7])
b = np.array([1, 5, 9, 2, 4])
```

```
distance = euclidean(a, b)
print(distance)
```

Bu fonksiyon iki matrisin karşılıklı satırları üzerinde işlem yapamamaktadır. Fonksiyonun parametreleri olan noktalar tek boyutlu dizi olarak girilmek zorundadır.

Tensorflow kütüphanesinde Öklit uzaklığı için hazır bir fonksiyon bulunmamaktadır. Ancak böyle bir fonksiyonu aşağıdaki gibi yazabiliriz:

```
import tensorflow as tf

def tf_euclidean_distance(a, b, axis=0):
    return tf.sqrt(tf.math.reduce_sum((a - b) ** 2, axis=axis))

a = tf.Variable([[3, 4, 2, 6, 7], [3, 4, 6, 7, 3], [2, 6, 1, 5, 6]], dtype=tf.float32)
b = tf.Variable([[1, 5, 9, 2, 4], [3, 5, 7, 1, 2], [4, 5, 2, 1, 7]], dtype=tf.float32)

distance = euclidean_distance(a, b, axis=1)
print(distance)
```

Kümeleme Yöntemlerinin Sınıflandırılması

Kümeleme için yüzün üzerinde algoritma önerilmiştir. Bu algoritmaların bazıları birbirinden çok farklı olmasına karşılık bazıları birbirlerine benzemektedir. Kümeleme algoritmalarını aşağıdaki gibi sınıflara ayıralım:

- 1) Ağırlık Merkezi Temelli (Centroid Based) Kümeleme Algoritmaları
- 2) Bağlantı Temelli (Connectivity Based) Kümeleme Algoritmaları (Hiperarşik Kümeleme Algoritmaları)
- 3) Yoğunluk Temelli (Density Based) Kümeleme Algoritmaları
- 4) Dağılım Temelli (Distribution Based) Kümeleme Algoritmaları
- 5) Bulanık Temelli (Fuzzy Based) Kümeleme Algoritmaları

Ağırlık merkezi temelli algoritmalarla bir kğmelerin ağırlık merkezleri temel alınarak elemanlar kümelere ayrılmaktadır. En popüler kümeleme algoritmalarından biri olan K-Means bu gruba örnek olarak verilebilir. Ağırlık merkezi temelli kümeleme algoritmaları verimli (efficient) olmalarına karşın başlangıç koşullarına ve uç değerlere çok duyarlıdır. Bağlantı temelli algoritmalarla hiperarşik kümeleme algoritmaları da denilmektedir. Bunlar "aşağıdan yukarı (bottom up)" ya da "yukarıdan aşağıya (top-down)" kümelerin birleştirilmesi esasına dayanırmaktadır. Aşağıdan yukarıya kümeleme algoritmalarına "agglomerative" algoritmalar, yukarıdan aşağıya kümeleme algoritmalarına ise "divisive" algoritmalar da denilmektedir. Yoğunluk temelli algoritmalarla belli bir bölgedeki yoğunluğa bakılmaktadır. DBSCAN ve OPTICS bu grup algoritmaların en yaygın kullanılanlardan biridir. Dağılım temelli algoritmalar benzer dağılıma sahip olan eleman gruplarını kümelemeye çalışan algoritmalarıdır. Dağılım temelli algoritmalar daha esnek olmasına karşın daha çok ayarlamaya gereksinim duymaktadır. Bulanık temelli algoritmalarla öğeler birden fazla kümeye dahil edilebilmektedir. Bunlara ilişkin maliyet fonksiyonları oluşturulmaktadır.

K-Means Kümeleme Yöntemi

En yaygın kullanılan ve en yalın kümeleme yöntemlerinden biri K-Means denilen yöntemdir. K-Means yönteminde işin başında verilerin kaç kümeye ayrılacağı belirlenmiş olması gereklidir. Bu değerin k olduğunu varsayıyalım. Algoritmada kümelenenecek elemanları nokta olarak ifade edeceğiz. Kümelenenecek noktaların sayısının da n tane olduğunu varsayıyalım. Algoritmanın işleyişi şöyledir:

- 1) Önce k tane kümeye (yani her kümeye) rastgele k tane ağırlık merkezi (centroid) oluşturulur.
- 2) Tüm noktaların bu ağırlık merkezlerine olan uzaklıklar hesaplanır. Bir nokta hangi ağırlık merkezine yakınsa o kümeye dahil edilir. Böylece bu adımda n tane nokta k tane kümeye atanmış olur.

3) Kümelerin ağırlık merkezleri bu kümelere atanmış noktalar kullanılarak yeniden hesaplanır. Ağırlık merkezi hesaplaması ortalama yoluyla yapılmaktadır. Yani o kümedeki tüm noktaların kendi aralarında ortalaması bulunur. (Örneğin 3 boyutlu uzayda bu ortalama kümedeki tüm noktaların x'lerinin, y'lerinin ve z'lerinin kendi aralarında ortalaması bulunarak hesaplanmaktadır.) Böylece yeni ve daha uygun bir ağırlık merkezi oluşturulmuş olur.

4) Bundan sonra yine tüm noktalar yeni ağırlık merkezlerine uzaklıklarına göre yeniden kümelenir. Bu kümleme bir öncekinden farklı olabilmektedir. (Yani yeni ağırlık merkezleri dikkate alındığında bazı noktalar küme değiştirebilmektedir.)

5) Yeni kümelerin ağırlık merkezleri yeniden ortalama yöntemiyle bulunur. Ve işlemler böyle devam ettirilir.

6) Yeniden kümleme sonucunda önceki kümelerle yeni kümeler arasında eleman bakımından hiç fark yoksa artık işlemler sonlandırılır. Yani artık hiçbir nokta küme değiştirmiyorsa algoritma devam etmenin bir anlamlı yoktur.

Algoritmanın işleyişini kartezyen koordinat sisteminde noktalarla açıklamaya çalışalım. Örneğimizde toplam 12 nokta olsun. Bu 12 nokta iki kümede kümelenmek istensin. Başlangıçtaki noktalar şöyledir:

Sıra	X	Y
1	7	8
2	2	4
3	6	4
4	3	2
5	6	5
6	5	7
7	3	3
8	1	4
9	5	4
10	7	7
11	7	6
12	2	1

İki küme için rastgele ağırlık merkezleri şöyle olsun: $C_1(1, 4)$, $C_2(7, 8)$. Şimdi bu 12 noktanın bu ağırlık merkezlerine Öklid uzaklıklarını hesaplayıp bu 12 noktayı iki kümeye dağıtalım:

Sıra	X	Y	C_1 Uzaklığı	C_2 Uzaklığı	Atanan Küme
1	7	8	7.21	0	Cluster-2
2	2	4	1.00	6.40	Cluster-1
3	6	4	5.00	4.12	Cluster-2
4	3	2	2.83	7.21	Cluster-1
5	6	5	5.10	3.16	Cluster-2
6	5	7	5.00	2.24	Cluster-2
7	3	3	2.24	6.40	Cluster-1
8	1	4	0.00	7.21	Cluster-1
9	5	4	4.00	4.47	Cluster-1
10	7	7	6.71	1.00	Cluster-2
11	7	6	6.32	2.00	Cluster-2
12	2	1	3.16	8.60	Cluster-1

Bu işlemin sonucunda Cluster-1 kümesinde $\{2, 4, 7, 8, 9, 12\}$, Cluster-2 kümesinde ise $\{1, 3, 5, 6, 10, 11\}$ noktaları bulunacaktır. Şimdi yeni ağırlık merkezlerini hesaplayalım. Bu hesaptan yeni ağırlık merkezleri $C_1 = (2.67, 3.00)$ ve $C_2 = (6.33, 6.17)$ biçiminde elde edilir. Şimdi bu noktaların hepsinin yeni ağırlık merkezlerine göre uzaklıklarını hesaplayarak yeniden kümleme yapmamız gereklidir.

Sıra	X	Y	C_1 Uzaklığı	C_2 Uzaklığı	Atanan Küme
1	7	8	7.21	0	Cluster-2

1	7	8	6.61	1.95	Cluster-2
2	2	4	1.20	4.84	Cluster-1
3	6	4	3.48	2.19	Cluster-2
4	3	2	1.05	5.34	Cluster-1
5	6	5	3.88	1.22	Cluster-2
6	5	7	4.63	1.57	Cluster-2
7	3	3	0.33	4.60	Cluster-1
8	1	4	1.95	5.75	Cluster-1
9	5	4	2.54	2.55	Cluster-1
10	7	7	5.89	1.07	Cluster-2
11	7	6	5.27	0.69	Cluster-2
12	2	1	2.11	6.74	Cluster-1

Burada kümelerdeki elemanlar değişmediği için artık algoritmaya son verilecektir.

K-Means algoritmasında ilk başta alınan rastgele noktalardan dolayı algoritmanın farklı çalıştırılmalarında farklı kümeler elde edilebilmektedir. Burada genellikle izlenen yol algoritmayı n defa çalıştmak ve bu n çalışmadan elde edilen kümeleri karşılaştırarak en iyi olanını bulmaktadır. Peki洒 kümeleri neye göre karşılaştırmalıyız? İşte burada izlenen yol genellikle her kümenin noktalarının kendi ağırlık merkezlerine uzaklıklarının toplamı en küçük olan kümelemeyi tercih etmektedir. Bu da aslında toplam varyansın en küçük olduğu kümenin seçilmesi anlamına gelmektedir.

Şimdi biz Python'da K-Means algoritmasını bir fonksiyon olarak kendimiz yazalım. Fonksiyonumuzun parametrik yapısı şöyledir olsun:

```
kmeans(dataset, nclusters, centroids=None)
```

Fonksiyonun birinci parametresi olan dataset k tane sütundan ve n tane satırda oluşan noktaların kümesini belirtmektedir. Örneğin iki boyutlu kartezyen koordinat sistemi için sütun sayısı 2 olacaktır. İkinci parametre toplam kaç kümenin kullanılacağını belirtir. Üçüncü parametre default None değerini almıştır. Programcı isterse başlangıç centroid değerlerini kendisi verebilir. Eğer bu değerleri kendisi vermezse rastgele olarak alınacaktır. centroids dizisi iki boyutludur. Bunun sütun sayısı veri kümesinin sütun sayısı kadar, satır sayısı ise nclusters kadar olmalıdır.

Fonksiyonumuzda işin başında her küme için rastgele bir ağırlık merkezin atanması gerekmektedir. Bu işlemi yapan aşağıdaki gibi bir fonksiyon yazılmıştır:

```
def rand_centroids(dataset, nclusters):
    ncols = dataset.shape[1]
    centroids = np.zeros((nclusters, ncols), dtype=np.float32)
    for i in range(ncols):
        maxi = np.max(dataset[:, i])
        mini = np.min(dataset[:, i])
        rangei = maxi - mini
        centroids[:, i] = mini + rangei * np.random.random(nclusters)

    return centroids
```

Burada dataset matrisinin sütunları tek tek ele alınmış sütunların en küçük ve en büyük değerleri arasında rastgele değerlerden noktalar oluşturulmuştur. Bu fonksiyon kmeans tarafından çağrılacaktır. Şimdi de kmeans fonksiyonunu yazalım:

```
from scipy.spatial.distance import euclidean

def kmeans(dataset, nclusters, centroids=None):
    nrows = dataset.shape[0]
    clusters = np.full(nrows, -1)
    if centroids == None:
        centroids = rand_centroids(dataset, nclusters)
```

```

change_flag = True
while change_flag:
    change_flag = False
    for i in range(nrows):
        min_val = np.inf
        min_index = -1
        for k in range(nclusters):
            if not np.any(np.isnan(centroids[k])):
                result = euclidean(dataset[i], centroids[k])
                if result < min_val:
                    min_val = result
                    min_index = k
        if clusters[i] != min_index:
            change_flag = True
        clusters[i] = min_index

    for i in range(nclusters):
        idataset = dataset[clusters == i]
        centroids[i] = np.mean(idataset, axis=0) if len(idataset) else np.nan

dataset_clusters = []
inertias = []
for i in range(nclusters):
    idataset = dataset[clusters == i]
    dataset_clusters.append(idataset)
    inertias.append(np.sum((idataset - centroids[i]) ** 2) if len(idataset) else np.nan)

return clusters, dataset_clusters, centroids, inertias

```

Buradaki kmeans fonksiyonumuz dörtlü bir demete geri dönmektedir. Fonksiyonun geri döndürdüğü demetin ilk elemanı her bir noktanın hangi kümeye atandığını gösteren bir ndarray nesnesidir. Buradaki küme numaraları 0'dan başlamaktadır. Geri döndürülen demetin ikinci elemanı hangi kümelerin hangi noktalara sahip olduğunu veren bir ndarray listedir. Bu listenin uzunluğu küme sayısı kadardır ve her elemanı o kümeye ilişkin noktaları tutmaktadır. Tabii büyük uygulamalarda büyük dizilerin bu biçimde geri döndürülmesi uygun olmayabilir. Geri döndürülen demetin üçüncü elemanı kümelerin ağırlık merkezlerinin bulunduğu ndarray nesnesidir. Geri döndürülen demetin son elemanı ise nihai durumdaki noktaların kendi ağırlık merkezlerine uzaklıklarının kareleri toplamıdır. Bu da istatistiksel bakımdan kümeler içerisindeki varyans toplamları anlamına gelmektedir. Bu varyans toplamlarına bu bağlamda "atalet (inertia)" denilmektedir. Daha önceden de belirtildiği gibi kmeans algoritması bir kez değil n kez çalıştırılmalı ve ev küçük atalet değerine sahip olan çalıştırmanın sonuçları nihai sonuç olarak ele alınmalıdır.

Yukarıdaki kodun daha önce verdigimiz örnek değerlerle iki küme için 5 kez çalıştırılmaları sonucunda aşağıdaki saçılım grafikleri elde edilmiştir

```

import matplotlib.pyplot as plt

dataset = np.loadtxt('points.csv', delimiter=',', dtype=np.float32)

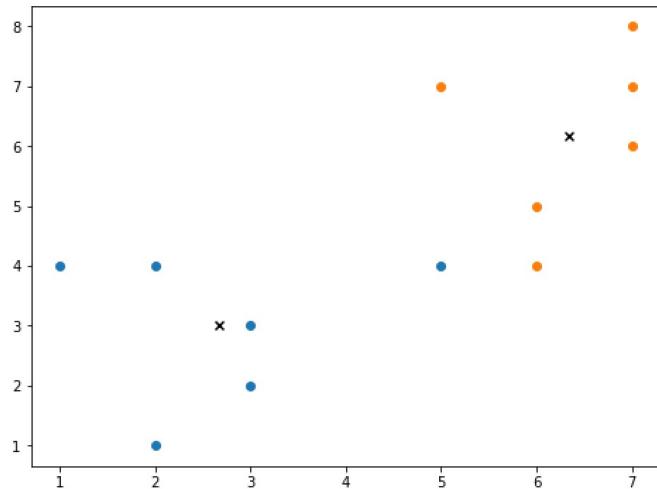
for i in range(5):
    clusters, dataset_clusters, centroids, inertias = kmeans(dataset, 2)

    plt.title('Clustered Points')
    figure = plt.gcf()
    figure.set_size_inches((8, 6))
    for i in range(2):
        plt.scatter(dataset[clusters == i, 0], dataset[clusters == i, 1])

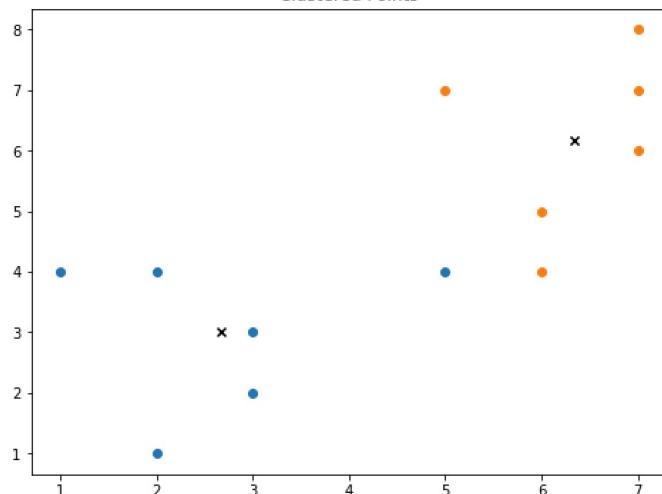
    plt.scatter(centroids[:, 0], centroids[:, 1], color='black', marker='x')
plt.show()

```

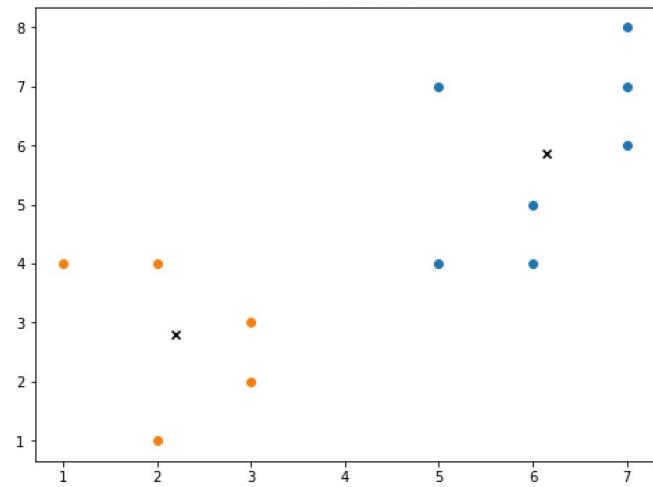
Clustered Points

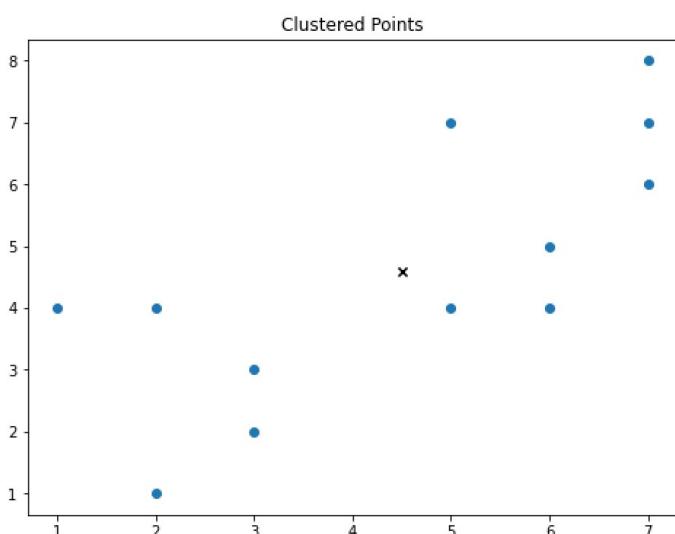
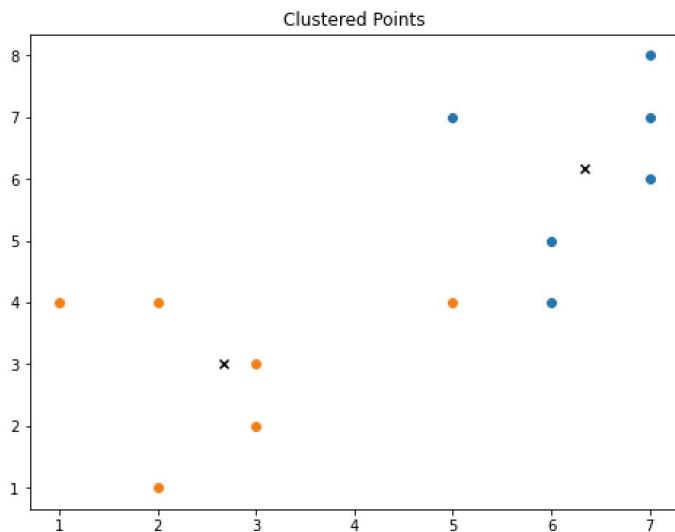


Clustered Points



Clustered Points





Algoritmanın her çalıştırılması sonucunda aynı noktaların aynı kümeler içerisinde bulunmayabileceğine dikkat ediniz. Burada son kümelemede biz iki küme oluşturulmasını istediğimiz halde tek küme oluşturulmuştur. Bunun nedeni kullandığımız algoritmada bir kusurdan kaynaklanmaktadır. Biz başlangıçta ağırlık merkezleri için rastgele noktalar aldık. Bu ağırlık merkezlerine ilişkin bir kümede hiçbir eleman yoksa artık o kümeye herhangi bir elemanın girme olasılığı kalmamaktadır. K-Means algoritmasında bu nedenle başlangıç ağırlık merkezlerinin rastgele seçiminde çeşitli teknikler kullanılmaktadır. Bunların en çok tercih edilenlerinden biri K-Means++ yöntemidir. Biz burada bu aürlük merkezlerinin rastgele seçiminde kullanılan teknikler üzerinde durmayacağız.

Örnekte kullandığımız "points.csv" dosyasının içeriği şöyledir:

```
7,8
2,4
6,4
3,2
6,5
5,7
3,3
1,4
5,4
7,7
7,6
2,1
```

Yukarıda da belirtildiği gibi algoritmanın n kez çalıştırılıp en iyi değerin (toplam ataletin en düşük olduğu değerin) alınması tipik uygulanan yöntemdir. Aşağıda bu yöntem uygulanmıştır:

```

REPEAT_COUNT = 20

dataset = np.loadtxt('points.csv', delimiter=',', dtype=np.float32)

min_inertia = np.inf
for i in range(REPEAT_COUNT):
    result = kmeans(dataset, 2)
    total = np.sum(inertias)
    if total < min_inertia:
        result_best = result

clusters, dataset_clusters, centroids, inertias = result_best

print(f'Best clusters: {clusters}')
print(f'Best dataset_clusters: {dataset_clusters}')
print(f'Best centroids: {centroids}')
print(f'Best inertias: {inertias}')

plt.title('Clustered Points')
figure = plt.gcf()
figure.set_size_inches((8, 6))
for i in range(2):
    plt.scatter(dataset[clusters == i, 0], dataset[clusters == i, 1])

plt.scatter(centroids[:, 0], centroids[:, 1], color='black', marker='x')
plt.show()

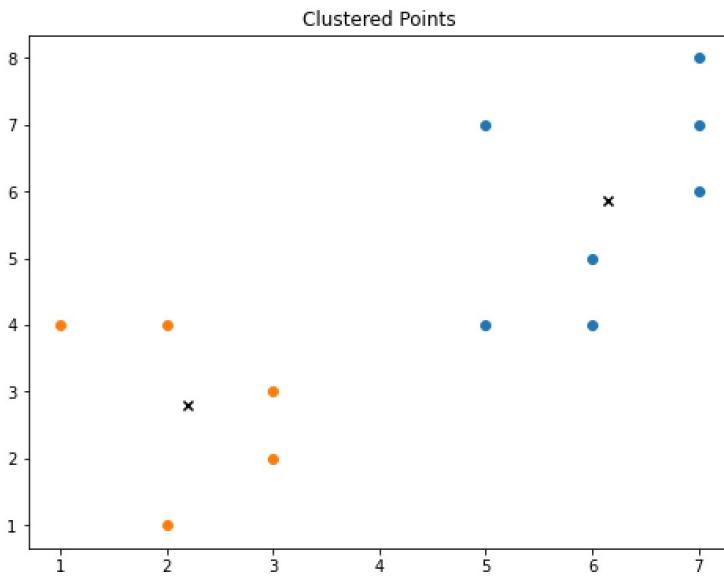
```

Buradan şöyle bir çıktı elde edilmiştir:

```

Best clusters: [0 1 0 1 0 0 1 1 0 0 0 1]
Best dataset_clusters: [array([[7., 8.],
   [6., 4.],
   [6., 5.],
   [5., 7.],
   [5., 4.],
   [7., 7.],
   [7., 6.]]), dtype=float32], array([[2., 4.],
   [3., 2.],
   [3., 3.],
   [1., 4.],
   [2., 1.]]), dtype=float32]
Best centroids: [[6.142857 5.857143]
 [2.2      2.8      ]]
Best inertias: [19.714285, 9.599999]

```



Biz yukarıdaki örnekte noktaları toplamda iki kümeye ayırdık. Pekiyi noktaların iki kümeye ayrılacağını nasıl biliyorduk? Aslında biz böyle bir bilgiye sahip değildik. Önce noktaların saçılma grafiğini çizdik. Bu grafikte noktaların kendi aralarında iki farklı toplasmaya sahip olduğunu gördük. Tabii noktalar iki boyutlu düzlemede gözle kolay biçimde algılanabilmektedir. Ancak ikiden fazla özelliğe (yani sütuna) sahip olan veri kümelerinde böyle bir grafik çizme imkanımız yoktur. O halde biz elimizdeki veri kümelerindeki satırların kaç kümeye ayrılacağını işin başında nasıl belirleyebiliriz? İşte izleyen bölümlerde bu konu ele alınacaktır. Şimdi yukarıdaki örnekteki noktaları 3 kümeye ayırtıralım:

```

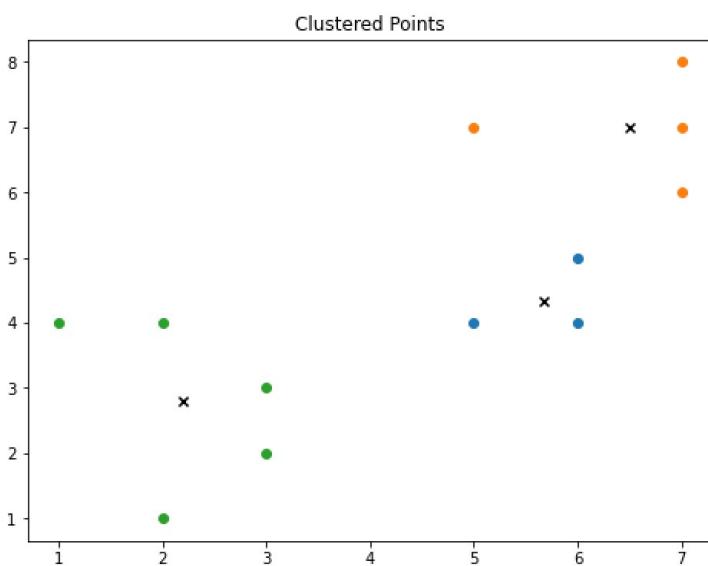
clusters, dataset_clusters, centroids, inertias = kmeans(dataset, 3)

plt.title('Clustered Points')
figure = plt.gcf()
figure.set_size_inches((8, 6))
for i in range(3):
    plt.scatter(dataset[clusters == i, 0], dataset[clusters == i, 1])

plt.scatter(centroids[:, 0], centroids[:, 1], color='black', marker='x')
plt.show()

```

Elde edilen saçılma grafiğini inceleyiniz:



Scikit-learn Kütüphanesindeki KMeans Sınıfının Kullanımı

K-Means kümeleme işlemleri için scikit-learn kütüphanesinde sklearn.cluster modülü içerisinde KMeans isimli bir sınıf bulunmaktadır. Sınıfın `__init__` metodunun parametreleri şöyledir:

```
KMeans(n_clusters=8, init='kmeans++', n_init=10, max_iter=300, tol=0.0001, precompute_distances='auto', verbose=0, random_state=None, copy_x=True, n_jobs=None, algorithm='auto')
```

Fonksiyonun `n_clusters` parametresi noktaların ayırtılacağı küme sayısını belirtmektedir. `n_init` parametresi en iyi değerin bulunması için uygulanacak çalışma sayısıdır. Bu parametrenin default durumda 10 değerini aldığına görürorsunuz. Yani sınıf default durumda rastgele ağırlık merkezleriyle 10 çalışma yapıp bunun en iyisini bize vermektedir. `max_iter` parametresi belli bir çözüm için yineleme yapılrken çözümün uzun süre bulunmaması durumunda işlenen maksimum iterasyon sayısını belirtmektedir. Bilindiği gibi K-Means algoritmasında her yinelemede (iterasyonda) kümelerdeki elemanlar yer değiştirebilmektedir. Zaten kümelerde hiçbir değişiklik olmadığı durumda sonuç elde edilmiş olmaktadır. `tol` parametresi farklı çalışırmalardaki yakınsama değerlerinin (yani toplam `inertia`'nın) duyarlığını belirtmektedir. Yani bir çalışırmayı diğerlerinden iyi kabul edilmesi için o çalışırmadaki toplam atletin (`inertia`) burada belirtilen değerden daha iyi olması gereklidir. Fonksiyonun diğer parametrelerini scikit-learn dokümanlarından inceleyebilirsiniz.

KMeans sınıfının `fit` isimli metodu algoritmayı işleyen asıl metottur. `fit` metodu bizden veri kümelerini parametre olarak alır, algoritmayı işletir, sonuçları sınıfın örnek özniteliklerine yerleştirir ve KMeans nesnesinin kendisine geri döner. Örneğin:

```
dataset = np.loadtxt('points.csv', delimiter=',', dtype=np.float32)

from sklearn.cluster import KMeans

km = KMeans(n_clusters=2)
km.fit(dataset)
```

Mademki `fit` metodu da aslında KMeans nesnesinin kendisine geri dönmektedir. O halde yukarıdaki işlem tek bir satırda şöyle de yapılabilirdi:

```
km = KMeans(n_clusters=2).fit(dataset)
```

Pekiye çözümün sonuçları nasıl elde edilecektir? İşte `fit` metodu sonuçları KMeans sınıfının çeşitli örnek özniteliklerine yerleştirmektedir. Bu örnek öznitelikleri ve tuttuğu değerler şunlardır:

labels_: Her noktanın atandığı küme numarasına ilişkin numpy dizisidir. Küme numarası 0'dan başlatılmaktadır. KMeans sınıfı bize kümelerde hangi noktaların bulunduğu değil, noktaların hangi kümede olduğunu vermektedir. Tabii biz bu `labels_` örnek özniteliği sayesinde kümelerdeki elemanları aşağıdaki gibi kolay bir biçimde elde edebiliriz:

```
from sklearn.cluster import KMeans

km = KMeans(n_clusters=2)
km.fit(dataset)

print(km.labels_)

print('İlk grubun noktaları:')
print(dataset[km.labels_ == 0])

print('İkinci grubun noktaları:')
print(dataset[km.labels_ == 1])
```

Şöyledir bir çıktı elde edilmiştir:

```
[1 0 1 0 1 1 0 0 1 1 1 0]
```

ilk grubun noktaları:

```
[[2. 4.]  
 [3. 2.]  
 [3. 3.]  
 [1. 4.]  
 [2. 1.]]
```

ikinci grubun noktaları:

```
[[7. 8.]  
 [6. 4.]  
 [6. 5.]  
 [5. 7.]  
 [5. 4.]  
 [7. 7.]  
 [7. 6.]]
```

cluster_centers_: Nihai çözümdeki kümelerin ağırlık merkezlerine ilişkin numpy dizisidir. Örneğin:

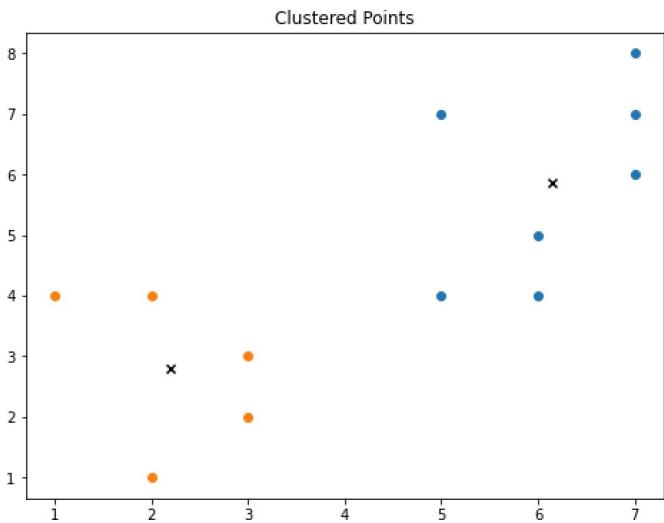
```
In [11]: km.cluster_centers_  
Out[11]:  
array([[2.2      ,  2.7999997],  
       [6.142857 ,  5.857143 ]], dtype=float32)
```

inertia_: Tüm noktaların kendi ağırlık merkezlerine uzaklıklarının karelerinin toplamını verir. Bu öznitelik tek bir elemandan oluşan toplam değerini vermektedir. (Halbuki kendi yaptığımız örnekte kmeans fonksiyonun geri döndürdüğü inertia değerinin her kümenin kendi elemanlarına ilişkin inertia değeri olduğuna dikkat ediniz.)

n_iter_: En iyi çözümün bulunması için uygulanan yineleme sayısını belirtmektedir.

Şimdi de yukarıdaki noktalar için scikit-learn KMeans sınıfı ile elde edilen çözümün saçılma grafiğini çizelim:

```
import numpy as np  
from sklearn.cluster import KMeans  
  
dataset = np.loadtxt('points.csv', delimiter=',', dtype=np.float32)  
km = KMeans(n_clusters=2)  
km.fit(dataset)  
  
import matplotlib.pyplot as plt  
  
plt.title('Clustered Points')  
figure = plt.gcf()  
figure.set_size_inches(8, 6)  
for i in range(2):  
    plt.scatter(dataset[km.labels_ == i, 0], dataset[km.labels_ == i, 1])  
  
plt.scatter(km.cluster_centers_[:, 0], km.cluster_centers_[:, 1], color='black', marker='x')  
plt.show()
```



KMeans sınıfının predict isimli metodu yeni bir verinin sınıfını bize vermektedir. Aslında predict Örneğin:

```

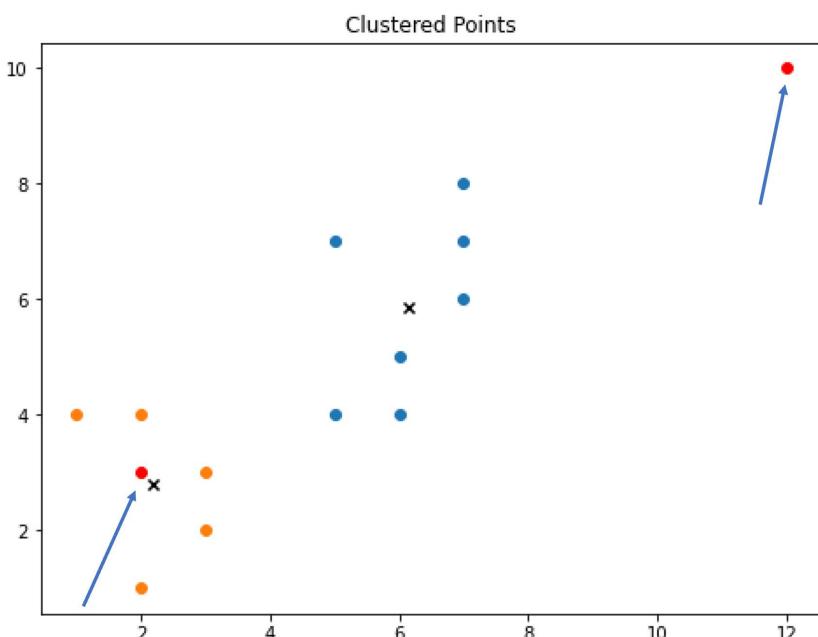
predict_data = np.array([[2, 3], [12, 10]], dtype=np.float32)
predict_result = km.predict(predict_data)

print(predict_result)

plt.title('Clustered Points')
figure = plt.gcf()
figure.set_size_inches((8, 6))
for i in range(2):
    plt.scatter(dataset[km.labels_ == i, 0], dataset[km.labels_ == i, 1])

plt.scatter(km.cluster_centers_[:, 0], km.cluster_centers_[:, 1], color='black', marker='x')
plt.scatter(predict_data[:, 0], predict_data[:, 1], color='red')
plt.show()

```



Oluşturduğumuz örnek iki noktanın kestiriminde şu sonuç elde edilmiştir:

[1 0]

Burada mavi 0 numaralı küme, kavuniçi 1 numaralı kümedir.

K-Means Kümeleme Yönteminde Özellik Ölçeklemesinin Önemi

K-Means kümeleme yönteminde eğer sütunlardaki değerlerin skalaları birbirlerinden farklısa noktalar arasındaki Öklit uzaklıklar hesaplanırken skalası büyük olan sütunlardaki etki skalası küçük olan sütunlara göre yüksek olacaktır. Bu da kümelemenin yanlış oluşturulmasına yol açabilecektir. Örneğin sütunlardan birisi üç basamaklı değerlerden diğerinin tek basamaklı değerlerden oluşuyor olsun. Uzaklık hesabında üç basamaklı değerlere ilişkin sütun çok baskın hale gelecektir değil mi? İşte bu tür durumlarda sütunlardaki büyülüklerin birbirilerine yaklaştırılması için ölçekleme gerekmektedir. Daha önceden de belirttiğimiz gibi bu tür durumlarda eğer sütun özellikleri normal dağılıyorsa standart ölçeklendirmesi değilse min-max ölçeklemesi tercih edilebilmektedir.

K-Means Yönteminin Zambak (Iris-Plant) Veri Kmesine Uygulanması

Zambak (Iris) resimleri sınıflandırma için eğitim amacıyla en fazla kullanılan veri kümelerinden biridir. Zambak veri kümесini aşağıdaki adresten indirebilirsiniz:

<https://www.kaggle.com/uciml/iris>

Veri tablosunda zambak resimleri alınarak bunların aşağıdaki özellikleri sayısal biçimde kodlanmıştır:

- Çanak yaprakların santimetre cinsinden uzunluğu
- Çanak yaprakların santimetre cinsinden genişliği
- Taç yaprakların santimetre cinsinden uzunluğu
- Taç yaprakların santimetre cinsinden genişliği

Bu verilerden hareketle zambaklar üç sınıfa ayrılmaktadır: Iris Setosa, Iris Versicolor ve Iris Virginica. Zambak veri kümesi 150 zambak bilgisinden oluşmaktadır. Her zambak için onun bulunduğu sınıf da veri tablosunda yer almaktadır (yani bu tablo eğitimli ağlarda da kullanılabilmektedir). Ancak biz burada K-Means yöntemini kullanarak eğitsiz biçimde kümeleme yapacağız.

Zambak veri tablosu bir CSV dosyası biçiminde indirilebilir. Dosyanın görünümü aşağıdaki gibidir:

```
5.1,3.5,1.4,0.2,Iris-setosa  
4.9,3.0,1.4,0.2,Iris-setosa  
4.7,3.2,1.3,0.2,Iris-setosa  
4.6,3.1,1.5,0.2,Iris-setosa  
5.0,3.6,1.4,0.2,Iris-setosa  
5.4,3.9,1.7,0.4,Iris-setosa  
4.6,3.4,1.4,0.3,Iris-setosa  
5.0,3.4,1.5,0.2,Iris-setosa  
4.4,2.9,1.4,0.2,Iris-setosa  
4.9,3.1,1.5,0.1,Iris-setosa  
5.4,3.7,1.5,0.2,Iris-setosa  
.....
```

Bu dosyadan hareketle veri kümесini dataset_x ve dataset_y biçiminde aşağıdaki gibi ikiye ayıralım:

```
import pandas as pd  
  
df = pd.read_csv('iris.csv')  
  
dataset_x = df.iloc[:, :-1].to_numpy()  
  
from sklearn.preprocessing import LabelEncoder  
  
le = LabelEncoder()  
dataset_y = le.fit_transform(df.iloc[:, 4])
```

Aslında Keras ve scikit-learn paketlerinde bu zambak verilerini bu biçimde bize veren sınıflar ve fonksiyonlar zaten bulunmaktadır. Yani benzer işlemler scikit-learn'de şöyle de yapabiliyoruz:

```
from sklearn.datasets import load_iris

iris = load_iris()

dataset_x = iris.data
dataset_y = iris.target
```

Biz burada verileri dataset_x ve dataset_y biçiminde iki gruba ayırdık. Ancak tabii hedefimiz bir eğitim uygulamak değildir. dataset_y'yi biz kümelemenin başarısını test etmek için ayrıca kullanacağız.

Şimdi veriler üzerinde min-max ölçeklendirmesi yapalım:

```
from sklearn.preprocessing import MinMaxScaler

mms = MinMaxScaler()
mms.fit(dataset_x)
scaled_dataset_x = mms.transform(dataset_x)
```

Şimdi de KMeans sınıfını kullanarak bu verileri üç kümeye ayıralım:

```
from sklearn.cluster import KMeans

km = KMeans(n_clusters=3)
km.fit(scaled_dataset_x)

print(km.labels_)

for i in range(3):
    cluster = dataset_x[km.labels_ == i]
    print('-----')
    print(f'Cluster-{i}')
    print('-----')
    print(cluster)
```

Veri kümemizdeki özellik sayısı 4 olduğu için noktaları kartezyen koordinat sisteminde gösteremiyoruz. Ancak sütunları 2'ye indirgerek böyle bir grafiği çizebiliriz. Sütun indirmeleri kursumuzda ilerideki bölümlerde ele alınmaktadır. Ancak burada biz henüz görmemiş olsak da sütun indirmemesi uygulayacağımız:

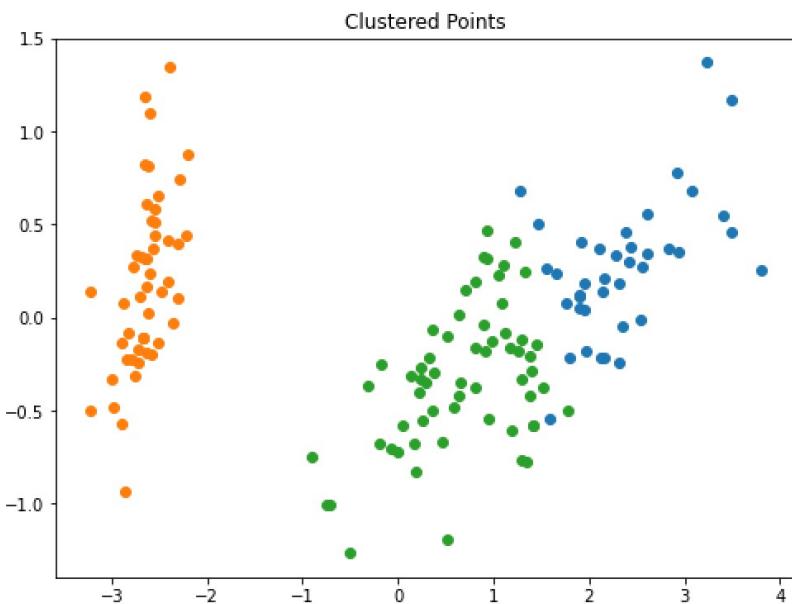
```
pca = PCA(n_components=2)
reduced_dataset_x = pca.fit_transform(dataset_x)

import matplotlib.pyplot as plt

plt.title('Clustered Points')
figure = plt.gcf()
figure.set_size_inches((8, 6))
for i in range(3):
    plt.scatter(reduced_dataset_x[km.labels_ == i, 0], reduced_dataset_x[km.labels_ == i, 1])

plt.show()
```

Şöyledir bir grafik elde edilmiş:



Şimdi de bir zambak bilgisi uydurup onun hangi sınıfı ilişkin olabileceğini kestirelim:

```
import numpy as np

predict_data = np.array([[4.9, 3.1, 1.5, 0.2]])
predict_data = mms.transform(predict_data)
predict_result = km.predict(predict_data)

print(predict_result[0])
```

Buradan şu sonuç elde edilmiştir:

1

Tabii burada bize predict ile verilen yalnızca bir küme numarasıdır. Hangi küme numarasının hangi zambak çeşidine karşılık geldiğini algoritma bilemez. Bunu bizim belirlememiz gerekdir. Başka bir deyişle K-Means algoritması sayısal bilgilere göre kümelieme yapmaktadır. O kümelerin ne anlam ifade ettiği hakkında bize bir bilgi vermemeştir.

Bazen uygulamacılar fit işlemi sonrasında labels_ özniteliğine bakmak yerine doğrudan asıl veri kümesiyle predict işlemi de yapabilmektedir. Örneğin:

```
labels = km.fit(dataset_x).predict(dataset_x)
```

Burada result ile elde edilen sonuç zaten labels_ özniteliği ile de alınmaktadır. fit metodunun KMeans nesnesinin kendisine geri döndüğünü anımsayınız.

Pekiyi mademki zambakların gerçek sınıfları (dataset_y) elimizde var. O halde biz K-Means algoritmasından elde ettiğimiz sonucun başarısını test edebiliriz. Ancak burada KMeans sınıfı ile elde ettiğimiz sınıf numaralarının dataset_y'deki sınıf numaralarıyla aynı olmayabileceğine dikkat etmek gereklidir. Bu nedenle önce gözle inceleyerek bu sınıf numaralarını dönüştürmek zorunda kalabiliriz. Aşağıda km.labels_ ile kümelenen zambaklarla onların gerçek kümeleri ekrana yazıdırılmıştır:

In [14]: dataset_y

Out[14]:

In [15]: km.labels_

Out[15]:

Burada gördüğünüz gibi aslında 0 numaralı küme bizde 1 numaralı olarak, 1 numaralı küme bizde 2 numaralı olarak, 2 numaralı küme ise bizde 0 numaralı olarak bulunmaktadır. Şimdi bunları dönüştürüp küeelenmenin performansını değerlendirelim:

```
converted_labels = np.zeros(dataset_y.shape)
converted_labels[km.labels_ == 2] = 1
converted_labels[km.labels_ == 0] = 2

performance_ratio = np.mean(converted_labels == dataset_y)
print(performance_ratio)
```

Burada 0.886 gibi bir oran elde edilmektedir. Yani bu örneğimizde zambakların %88.6'sı doğru bir biçimde kümelere ayrılmıştır. Burada elde edilen kümeye numaralarının dolayısıyla da yapılan dönüştürmenin programın her çalıştırılmasında farklı olabileceği dikkat ediniz.

Yukarıdaki kodun tamamını aşağıda veriyoruz. (Ancak performans kısmını koda dahil etmiyoruz. Çünkü bu kısmı siz gözle bakarak o anki çalışmaya göre oluşturmanız gereklidir):

```
import pandas as pd

df = pd.read_csv('iris.csv')

dataset_x = df.iloc[:, :-1].to_numpy()

from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
dataset_y = le.fit_transform(df.iloc[:, 4])

from sklearn.preprocessing import MinMaxScaler

mms = MinMaxScaler()
mms.fit(dataset_x)
scaled_dataset_x = mms.transform(dataset_x)

from sklearn.cluster import KMeans

km = KMeans(n_clusters=3)
km.fit(scaled_dataset_x)
```

```

print(km.labels_)

for i in range(3):
    cluster = dataset_x[km.labels_ == i]
    print('-----')
    print(f'Cluster-{i}')
    print('-----')
    print(cluster)

from sklearn.decomposition import PCA

pca = PCA(n_components=2)
reduced_dataset_x = pca.fit_transform(dataset_x)

import matplotlib.pyplot as plt

plt.title('Clustered Points')
figure = plt.gcf()
figure.set_size_inches((8, 6))
for i in range(3):
    plt.scatter(reduced_dataset_x[km.labels_ == i, 0], reduced_dataset_x[km.labels_ == i, 1])

plt.show()

import numpy as np

predict_data = np.array([[4.9, 3.1, 1.5, 0.2]])
predict_data = mms.transform(predict_data)
predict_result = km.predict(predict_data)

print(predict_result[0])

```

K-Means Kümeleme Yönteminin MNIST Veri Kümesine Uygulanması

Şimdi de rakamları sınıflandırmak için kullandığımız MNIST veri kümesini K-Means kümeleme yöntemine uygulayalım. Önce MNIST verilerini yükleyip min-max ölçeklendirmesi yapalım:

```

from tensorflow.keras.datasets import mnist

(training_dataset_x, training_dataset_y), (test_dataset_x, test_dataset_y) = mnist.load_data()

training_dataset_x = training_dataset_x.reshape(-1, 28 * 28)
training_dataset_x = training_dataset_x / 255
test_dataset_x = training_dataset_x.reshape(-1, 28 * 28)
test_dataset_x = test_dataset_x / 255

```

Şimdi KMeans işlemini uygulayalım:

```

from sklearn.cluster import KMeans

km = KMeans(n_clusters=10)
km.fit(training_dataset_x)

print(km.labels_)

```

Şimdi her sınıfın rastgele 9 tane resmi çizdirelim:

```

import matplotlib.pyplot as plt
import numpy as np

for i in range(10):
    random_indices, = np.where(km.labels_ == i)

```

```

random_images = training_dataset_x[np.random.choice(random_indices, 9)]

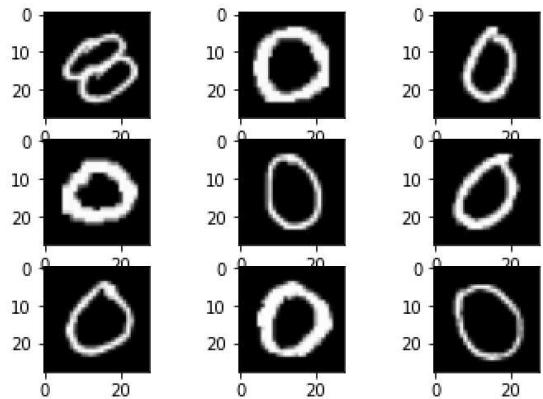
print('-----')
print(f'Cluster {i}')
print('-----')

for k in range(9):
    plt.subplot(3, 3, k + 1)
    plt.imshow(random_images[k].reshape(28, 28), cmap='gray')
plt.show()

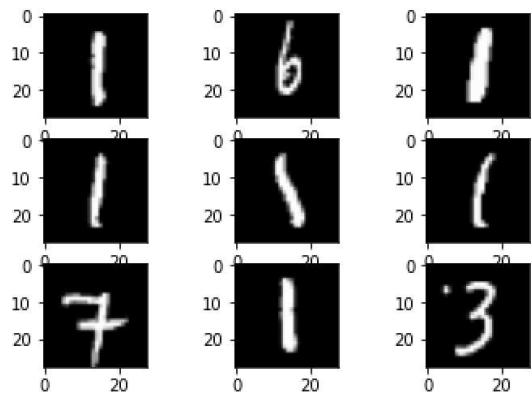
```

Bu sınıfların bazıları şöyledir:

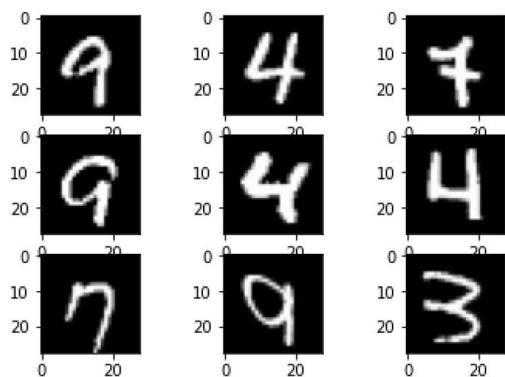
Cluster 2



Cluster 5



Cluster 6



MNIST veri kümesi için yapay sinir ağları ile yapılan sınıflandırmanın K-Means ile yapılan kümelendirmeden oldukça iyi olduğu bu sonuçlardan da görülmektedir.

Kümelemede Küme Sayılarının İşin Başında Tahmin Edilmesi

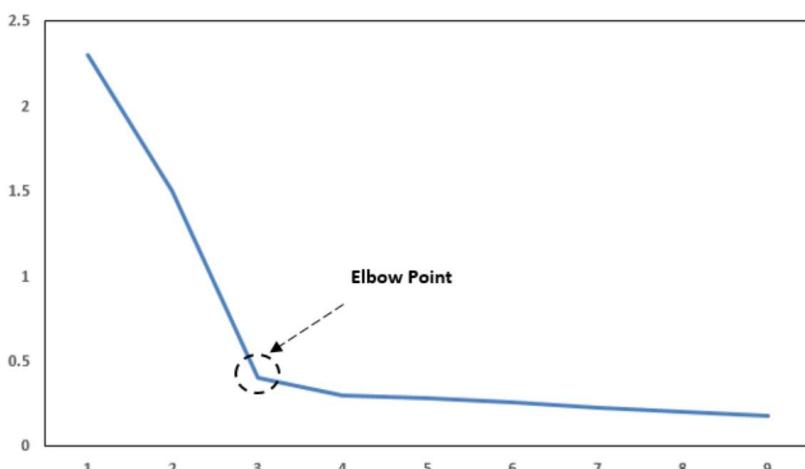
Biz K-Means yöntemini kullanırken verileri işin başında kaç kümeye ayıracagımızı biliyor olmamız gerekmektedir. Aslında bu bilgi bazen problemin kendisinde bulunmaktadır. Örneğin birtakım meyve fotoğrafları olabilir. Bu fotoğraftaki meyveleri sınıflandırmak isteyebiliriz. İşte eğer fotoğraflarda toplam kaç çeşit meyve olduğunu zaten biliyorsak küme sayısını da biliyoruz demektir. Peki ya resimlerde toplamda kaç çeşit meyve olduğunu bilmiyorsak? İşte bu tür durumlarda K-Means yöntemini kullanmadan önce verileri inceleyerek toplam kaç kümeyi söz konusu olacağına ilişkin bir çıkarımın yapılması gerekecektir.

K-Means yönteminde işin başında küme sayısının bulunması temelde iki yöntemle yapılmaktadır:

- 1) Dirsek (Elbow) Yöntemi
- 2) Silhouette Yöntemi

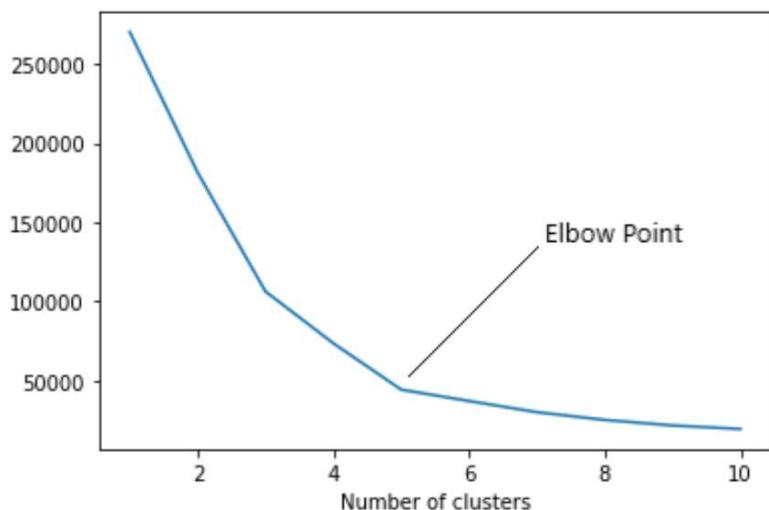
Her iki yöntemde de biz deneysel olarak küme sayılarını 1'den belli bir sayıya kadar oluşturarak kümelemeyi yaparız. Sonra birtakım değerleri karşılaştırarak en uygun küme sayısını belirleriz.

Dirsek yönteminde her kümedeki elemanın kendi kümelerinin ağırlık merkezlerine uzaklıklarının toplamı hesaplanır. Buna "atalet (inertia)" denildiğini anımsayınız. Sonra bu toplam uzaklıkların grafiği çizilir. Bu grafikte eğim düşümlerinin azlığı yere ilişkin olan küme sayısı (bu dirseğin bulunduğu yerdir) en iyi küme sayısı olarak seçilir. Bu yöntem görsel biçimde karar vermeye dayalıdır. Aşağıda örnek dirsek grafiğini inceleyiniz:



Alıntı Notu: GörSEL <https://www.oreilly.com/library/view/statistics-for-machine/9781788295758/c71ea970-0f3c-4973-8d3ab09a7a6553c1.xhtml> adresinden alınmıştır.

Burada dirsek noktası 3'tür. Aşağıdaki dirsek grafiğinde dirsek noktası 5'tir.



Alıntı Notu: GörSEL <https://www.analyticsvidhya.com/blog/2021/01/in-depth-intuition-of-k-means-clustering-algorithm-in-machine-learning/> adresinden alınmıştır.

Grafiklerden de görüldüğü gibi dirsek noktası eğrinin kırılıp yataylaşmaya başladığı noktadır. Şimdi biz ilk örneğimizdeki noktalardan küme sayısını tahmin etmek için dirsek yöntemini kullanalım. Noktalarımız şöyleydi:

```
import numpy as np

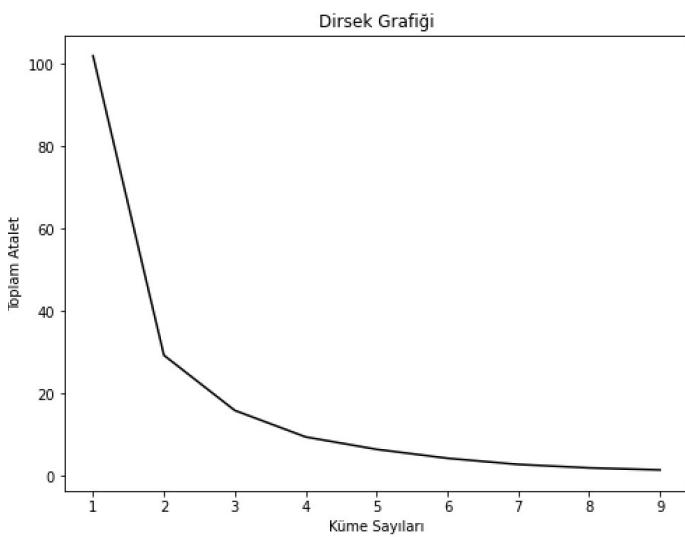
dataset = np.loadtxt('points.csv', delimiter=',', dtype=np.float32)

from sklearn.cluster import KMeans

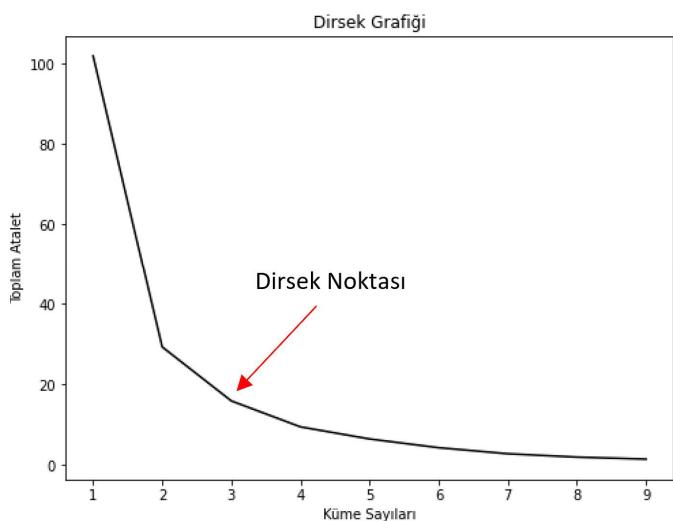
total_inertias = [KMeans(n_clusters=i).fit(dataset).inertia_ for i in range(1, 10)]

import matplotlib.pyplot as plt

plt.title('Dirsek Grafiği')
figure = plt.gcf()
figure.set_size_inches((8, 6))
plt.xlabel('Küme Sayıları')
plt.ylabel('Toplam Atalet')
plt.plot(range(1, 10), total_inertias, c='black')
plt.show()
```



Bu grafiğe baktığımızda yataya geçme eğiliminin 3'ten başladığı görülmektedir. Yani grafikteki eğrinin kırılma noktası 3 gibi gözükmektedir. O halde dirsek yöntemine göre bu noktalar üç kümeye ayrılmalıdır.



Şimdi noktaları üç kümeye ayırarak çözümü yeniden bulalım:

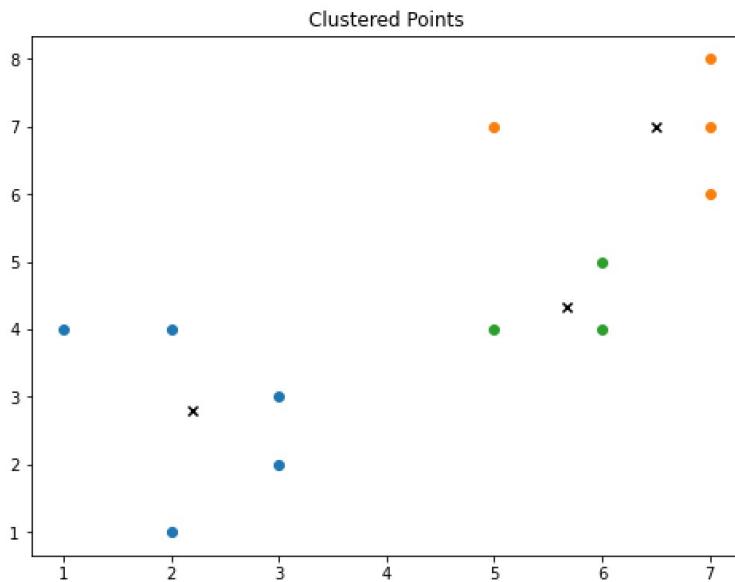
```
import numpy as np
from sklearn.cluster import KMeans

dataset = np.loadtxt('points.csv', delimiter=',', dtype=np.float32)
km = KMeans(n_clusters=3)
km.fit(dataset)

import matplotlib.pyplot as plt

plt.title('Clustered Points')
figure = plt.gcf()
figure.set_size_inches(8, 6)
for i in range(3):
    plt.scatter(dataset[km.labels_ == i, 0], dataset[km.labels_ == i, 1])

plt.scatter(km.cluster_centers_[:, 0], km.cluster_centers_[:, 1], color='black', marker='x')
plt.show()
```



Şimdi de zambak verileri için dirsek grafiğini çizelim:

```
from sklearn import datasets

iris = datasets.load_iris()
dataset_x = iris.data
dataset_y = iris.target

from sklearn.preprocessing import StandardScaler

ss = StandardScaler()
ss.fit(dataset_x)
transformed_dataset_x = ss.transform(dataset_x)

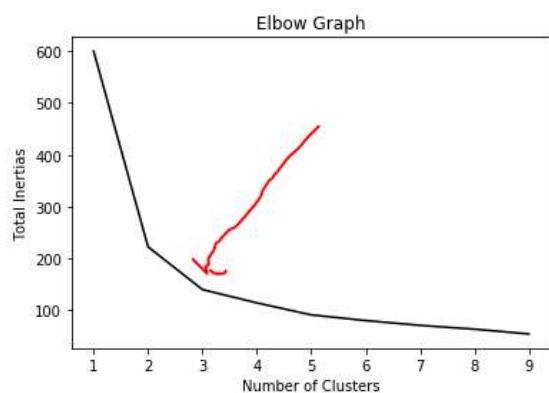
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

from sklearn.cluster import KMeans

total_inertias = [KMeans(n_clusters=i).fit(transformed_dataset_x).inertia_ for i in range(1, 10)]

import matplotlib.pyplot as plt

plt.title('Elbow Graph')
plt.xlabel('Number of Clusters')
plt.ylabel('Total Inertias')
plt.plot(range(1, 10), total_inertias, c='black')
```



Grafikten de görüldüğü gibi eğrinin yataya geçtiği küme sayısı 3'tür.

Silhouette yöntemi nümerik bir yöntemdir. Yöntemin ayrıntıları burada ele alınmayacaktır. Ancak bu yöntemi uygulayan hazır bir fonksiyon vardır. Bu fonksiyon sklearn.metrics modülündeki silhouette_score isimli fonksiyondur. Bu yöntemde kümeler için tek tek silhouette_score değerleri bulunur. Bu değerin en yüksek olduğu küme sayısından 1 fazlası nihai küme sayısı olarak tespit edilir. Yukarıdaki "point.csv" dosyasındaki veriler için en uygun küme sayısını bulmaya çalışalım:

```
import numpy as np

dataset = np.loadtxt('points.csv', delimiter=',', dtype=np.float32)

from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

kms = [KMeans(n_clusters=i).fit(dataset) for i in range(2, 10)]
sscores = [silhouette_score(dataset, km.labels_) for km in kms]

for k, sscore in enumerate(sscores):
    print('{:}->{}'.format(k + 2, sscore))
print(np.argmax(sscores) + 3)
```

Elde edilen çıktı şöyledir:

```
2---->0.5544097423553467
3---->0.47607627511024475
4---->0.4601795971393585
5---->0.4254012405872345
6---->0.3836685121059418
7---->0.29372671246528625
8---->0.21625618636608124
9---->0.114889957010746
3
```

Görüldüğü gibi burada 2 kümeli çözümdeki silhouette değeri en yüksektir (0.5544). O halde en uygun küme sayısı bu değerden bir fazla yani 3 olacaktır. Yukarıdaki örnekte de gördüğünüz gibi küme sayıları 2'den başlatılmıştır. 1 kğme için silhouette değeri söz konusu olmaz.

Örneğin Iris örneği için Silhouette değeri şöyle hesaplanabilir:

```
from sklearn import datasets

iris = datasets.load_iris()
dataset_x = iris.data
dataset_y = iris.target

from sklearn.preprocessing import StandardScaler

ss = StandardScaler()
ss.fit(dataset_x)
transformed_dataset_x = ss.transform(dataset_x)

from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

kms = [KMeans(n_clusters=i).fit(transformed_dataset_x) for i in range(2, 10)]
sscores = [silhouette_score(transformed_dataset_x, km.labels_) for km in kms]

for k, sscore in enumerate(sscores):
    print('{:}->{}'.format(k + 2, sscore))
```

```
print(np.argmax(sscores) + 3)
```

Bu örnekte 2'den başlayarak 10'a kadar tek tek küme sayıları için silhouette_score değerleri hesaplanmıştır. Değerler şu biçimdedir:

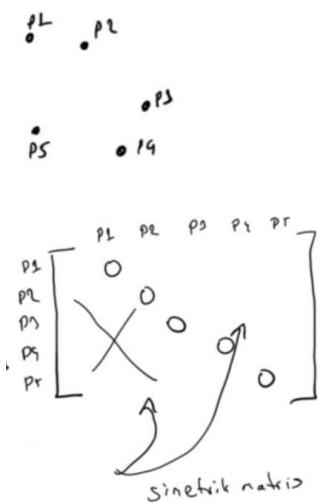
```
2---->0.5817500491982808  
3---->0.45994823920518635  
4---->0.41511334907493763  
5---->0.34551099599809465  
6---->0.3304816878968672  
7---->0.32456665698942505  
8---->0.34135763346672265  
9---->0.3316593073430383  
3
```

Sonra bu değerler içerisindeki en yüksek değer bulunur. Onun küme sayısının bir fazlası (örneğimizde 3) en iyi küme sayısı olarak tespit edilir.

Hiyerarşik Kümeleme (Hierarchical Clustering) Yöntemleri

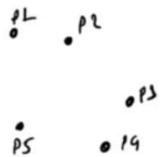
K-Means kümeleme yönteminin yanı sıra daha pek çok kümeleme yöntemi de vardır. Bazı yöntemler bazı yöntemlerin biraz değiştirilmiş ve probleme göre genişletilmiş biçimleridir. Hiyerarşik kümeleme ise grup olarak özgün bir yöntem grubudur. Hiyerarşik kümeleme kendi aralarında "Agglomerative" ve "Divisive" olmak üzere ikiye ayrılmaktadır. Agglomerative yöntem tümevarımsal, divisive yöntem tümenden gelimseldir. Daha çok uygulamada Agglomerative yöntem tercih edilmektedir. Agglomerative hiyerarşik kümeleme algoritması şöyle işletilmektedir:

- 1) Önce her nokta ayrı bir küme varsayılarak işleme başlanır.
- 2) Sonra her noktanın her noktadan uzaklığı hesaplanır. Uzaklık hesaplamada çeşitli yöntemler ve metrikler kullanılmaktadır. Burada yöntem uzaklık hesaplamanın neye dayalı olarak yapılacağını belirtir. Metrik ise hesaplama biçimini belirtmektedir. Örneğin "öklit uzaklığı" bir yöntem değil metriktir. Ancak iki kümenin en yakın noktalarını hesaplamada kullanmak bir yöntemdir. Başka bir deyişle "yöntem birden fazla noktanın tek nokta olarak ifade edilmesiyle ilgilidir, metrik ise iki noktanın uzaklığının hesaplanmasıdır". İleride ele alınacağı gibi uzaklık hesaplamada çeşitli yöntemler tercih edilebilmektedir. Bu adımın sonunda bir uzaklık matrisi elde edilir. Örneğin 5 nokta söz konusu olsun. Uzaklık matrisi şöyle bir yapıda olacaktır:

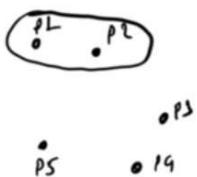


- 3) Uzaklık matrisinde en yakın iki nokta tespit edilir ve bu iki nokta yeni bir küme oluşturacak biçimde birleştirilir. Sonra yeniden tüm uzaklıklar yeni duruma göre hesaplanır. Aynı işlemler devam ettirilir. Her yinelemede bir eleman diğer bir küme ile birleştirilecektir. Bu işlem azrzu edilen küme sayısına gelinene kadar devam ettirilir.

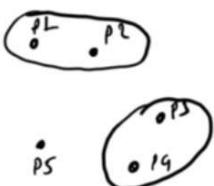
Örneğin işin başında aşağıdaki gibi 5 nokta olsun:



Burada tüm noktalar arasındaki uzaklıklar hesaplandığında P1 ile P2'nin en yakın noktalar olduğunu varsayıyalım. Bu durumda bu iki noktayı tek bir küme olarak birleştiririz.



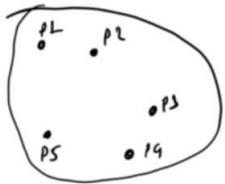
Artık burada 4 küme vardır. Bu 4 kümenin birbirlerine uzaklılığı hesaplanarak yeni bir uzaklık matrisi oluşturulur. Buradaki sorunlardan biri birden fazla noktadan oluşan kümeler için uzaklık yerinin neresi alınacağıdır. İşte bunun için izleyen kısımda açıklayacak olduğumuz birkaç yöntem kullanılmaktadır. Örneğin bu kümelerin en yakın noktalarının kullanılması yöntemlerden biridir. Bu haliyle en yakın iki nokta P3 ve P4 noktalarıdır. O halde bu noktalar da birleştirilir:



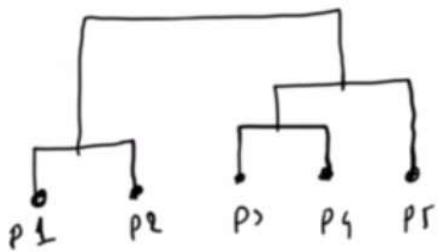
Bu yeni durumda P5 ile {P3, P4} kümelerinin mesafelerinin en yakın olduğunu varsayıyalım. O halde bunlardan yeni bir küme yapılacaktır:



Artık burada iki tane küme kalmıştır. Bu aşamada onlar da birleştirilir:



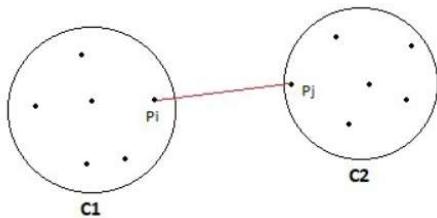
Hangi kümelerin hangi kümelerle birleştirildiğini gösteren grafiğe "dendrogram" denilmektedir.



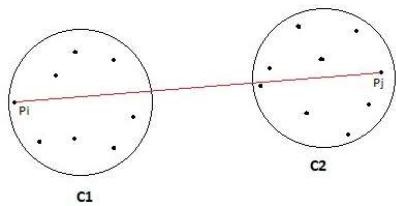
Divisive yöntemde tamamen işlemler tersten yapılmaktadır. Yani işin başında sanki tek bir küme varmış gibi başlanır. Sonra her defasında bu kümeden ayırtımalar yapılır. Başka bir deyişle agglomerative yöntem tümeyerimsal (bottom-up), Divisive yöntem ise tümdengelimsel (top-down) biçimdedir.

Birden fazla noktadan oluşan kümeler için uzaklık hesaplamasında şu yöntemler kullanılmaktadır:

Min Yöntemi: Burada uzaklık olarak iki kümenin en yakın iki noktası arasındaki uzaklık alınır. Örneğin:



Max Yöntemi: Burada uzaklık olarak iki kümenin en uzak iki elemanı arasındaki uzaklık alınır.



Grup Ortalaması Yöntemi: Bu yöntemde iki kümenin tüm iki noktalarının uzaklıklarının ortalaması alınır. Örneğin kümelerden birinin 10 noktası diğerinin 5 noktası olsun. Mümkün uzaklıkların sayısı $10 * 5 = 50$ 'dir. Bu 50 uzaklık toplanıp 50'ye bölünür.

Ward Yöntemi: Bu yöntem grup ortalaması yönteminin çok benzeridir. Tek farkı noktalar arasındaki uzaklıkların karelerinin ortalamasının alınmasıdır. Uygulamaların çoğu default olarak bu yöntem tercih edilmektedir.

Hiyerarşik Kümeleme Yönteminin Scikit-Learn Kullanılarak Gerçekleştirilmesi

Hiyerarşik kümeleme için Scikit-Learn içerisindeki `sklearn.cluster` modülündeki `AgglomerativeClustering` isimli sınıf kullanılmaktadır. Sınıfın `__init__` metodunun parametrik yapısı şöyledir:

```
sklearn.cluster.AgglomerativeClustering(n_clusters=2, affinity='euclidean', memory=None, connectivity=None, compute_full_tree='auto', linkage='ward', pooling_func='deprecated', distance_threshold=None)
```

Fonksiyonun `n_clusters` parametresi ayrılacak küme sayısını, `linkage` parametresi kullanılacak yöntemi belirtir. Bunun default olarak 'ward' olduğuna dikkat ediniz. `affinity` parametresi kullanılacak metriki belirtmektedir. Bu parametre de default olarak 'euclidean' biçimindedir. Fonksiyon bize bir sınıf nesnesi verir. Buradan alınan nesneyle tipki KMeans sınıfında olduğu gibi `fit` metodu çağrılır. Benzer biçimde yine kümeler sınıfın `labels_` isimli özniteligidinden elde edilmektedir. Şimdi yukarıdaki "points.csv" örneğini üç sınıf için `AgglomerativeClustering` sınıfını kullanarak çözelim:

```
import numpy as np
```

```

points = np.loadtxt('points.csv', delimiter=',', dtype=np.float32)

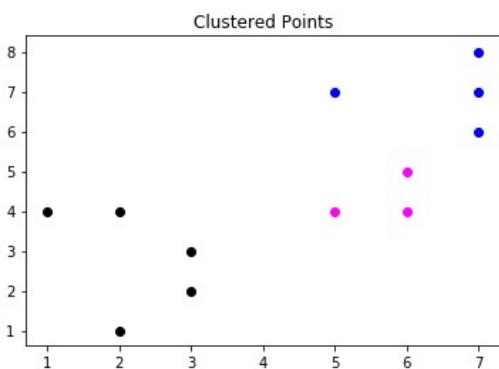
from sklearn.cluster import AgglomerativeClustering

ac = AgglomerativeClustering(n_clusters=3)
ac.fit(points)

import matplotlib.pyplot as plt

plt.title('Clustered Points')
plt.scatter(points[ac.labels_ == 0, 0], points[ac.labels_ == 0, 1], c='black')
plt.scatter(points[ac.labels_ == 1, 0], points[ac.labels_ == 1, 1], c='blue')
plt.scatter(points[ac.labels_ == 2, 0], points[ac.labels_ == 2, 1], c='magenta')

```



AgglomerativeClustering sınıfının clusters_centers_ biçiminde bir örnek özniteliği yoktur. Çünkü hiyerarşik kümelemede ağırlık merkezi (centroid) kullanılmamaktadır. Benzer biçimde yine bu sınıfın predict metodu da yoktur. Çünkü hiyerarşik sınıflamada bu anlamda bir kestirim yapılamaz. Yeni bir nokta için tüm işlemlerin baştan başlatılması gerekmektedir.

Şimdi de Iris örneğini hem K-Means yönetemi ile hem de hiyerarşik yöntemle çözüp sonuçları kontrol edelim.

```

from sklearn.datasets import load_iris

iris = load_iris()
dataset_x = iris.data
dataset_y = iris.target

from sklearn.preprocessing import StandardScaler

ss = StandardScaler()
transformed_dataset_x = ss.fit_transform(dataset_x)

from sklearn.cluster import KMeans, AgglomerativeClustering

km = KMeans(n_clusters=3)
km.fit(transformed_dataset_x)

ac = AgglomerativeClustering(n_clusters=3)
ac.fit(transformed_dataset_x)

print(km.labels_)
print(ac.labels_)

km_labels = [{1: 0, 2: 1, 0: 2}[label] for label in km.labels_]
ac_labels = [{1: 0, 2: 1, 0: 2}[label] for label in ac.labels_]

import numpy as np

km_success_ratio = np.sum(km_labels == dataset_y) / len(dataset_y)

```

```

ac_success_ratio = np.sum(ac_labels == dataset_y) / len(dataset_y)

print('KMeans success ration: {}'.format(km_success_ratio))
print('Agglomerative success ration: {}'.format(ac_success_ratio))

ratio = np.sum(km.labels_ == ac.labels_) / len(ac.labels_)
print('Identical class ratio: {}'.format(ratio))

```

Bu programda biz K-Means yöntemi ile Agglomerative yöntemin çıktılarını karşılaştırdık. Buradaki başarı oranı şöyle bulunmuştur:

```

KMeans success ration: 0.8333333333333334
Agglomerative success ration: 0.8266666666666667

```

Göründüğü gibi KMeans çok az daha başarılı olmuşsa da değerler birbirlerine çok yakındır.

Agglomerative Yöntemde Dendrogram Çizilmesi

Anımsanacağı gibi dendrogram hiyerarşik agglomerative kğmelemede hangi noktaların hangi noktalarla birleştirildiğini gösteren bir ağaç grafiğiydi. Dendrogram denilen ağaç grafiğinin oluşturulması için scipy kütüphanesinde scipy.cluster.hierarchy modülünde dendrogram isimli bir fonksiyon bulunmaktadır. Bu fonksiyon çizimin kendisini yapar. Ancak bu fonksiyon çizimin nasıl yapılacağını parametresiyle aldığı matrise bakarak tespit etmektedir. Bu matris de aynı modüldeki linkage isimli fonksiyon tarafından elde edilmektedir. Yani linkage fonksiyonu aslında agglomerative kümemeleme işlemini yapıp birleştirme bilgilerini bize matris olarak verir. Biz de bu matrisi dendrogram fonksiyonuna veririz.

Points.csv verileri üzerinde dendrogram şöyleden çizilebilir:

```

import numpy as np

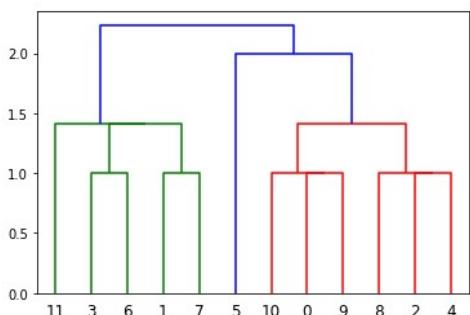
points = np.loadtxt('points.csv', delimiter=',', dtype=np.float32)

from scipy.cluster.hierarchy import linkage, dendrogram

import matplotlib.pyplot as plt

m = linkage(points)
dendrogram(m)

```



Iris verileri için de dendrogram şöyleden çizilebilir:

```

from sklearn import datasets
from scipy.cluster.hierarchy import linkage, dendrogram

import matplotlib.pyplot as plt

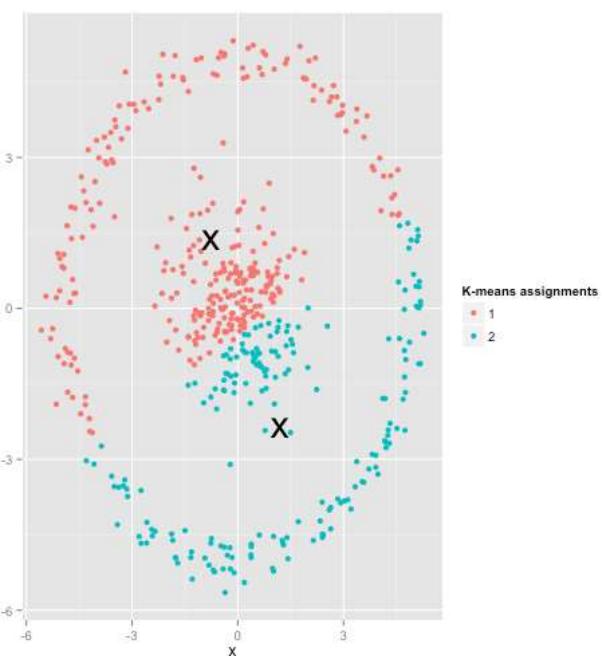
plt.figure(figsize=(25, 10))
m = linkage(transformed_dataset_x)
dendrogram(m)

```

K-Means Yöntemiyle Hiyerarşik (Agglomerative) Yöntemin Karşılaştırılması

Aslında her iki yöntemin de başarısı birbirine benzer olabilmektedir. Ancak iki yöntemin arasındaki farklılıklar şöyle özetlenebilir:

- Yüksek sayıda nokta söz konusu olduğunda hiperarşik (agglomerative) yöntem daha fazla zaman almaktadır. Çünkü işin başında tüm noktaların ayrı kümeler olarak ele alınması ve gitgide birleştirilmesi uzun hesaplama zamanına yol açmaktadır.
- Hiperarşik yöntemde algoritma rastgele bir biçimde başlatılmamaktadır. Dolayısıyla algoritmanın her çalıştırılmasında aynı sonucun bulunmasına yol açar. Halbuki K-Means yönteminde başlangıç ağırlık merkezleri rastgele alındığı için algoritmanın her çalıştırılmasında farklı sonuçlar elde edilebilmektedir.
- Hiperarşik yöntem daha esnektir. Çünkü kümeler arasındaki uzaklık hesaplama yöntemi (similarity) değiştirilebilmektedir. Oysa K-Means yönteminde yalnızca metrik değer değiştirilebilmektedir.
- Hiperarşik yöntemde dendrogram çizilebilmekte dolayısıyla hangi kümelerin hanki kümelerle birleştirildiği gözle görülebilmektedir. Dendrograma bakılarak küme uzaklıklarına dayalı küme sayıları bulunabilmektedir.
- KMeans yöntemi küresel (spherical) olmayan nokta dağılımlarında iyi çalışmaz. Küresel nokta dağılımları noktaların bir merkez etrafında toplanması biçiminde oluşan dağılımlardır. Küresel olmayan nokta dağılımlarında hiperarşik yöntem daha iyi sonuç vermektedir. Aslında küresel olmayan noktalar için en iyi yöntem grubu yoğunlu tabanlı (density based) olanlardır. Aşağıda küresel olmayan nokta dağılımlarındaki kmeans kümelerini görüyorsunuz:

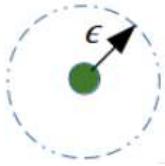


Şekil <http://varianceexplained.org/r/kmeans-free-lunch/> makalesinden alınmıştır.

Burada belki de dış dairesel noktaların ayrı bir küme, iç dairesel noktaların ayrı bir kme olarak değerlerndirilmesi daha uygun olabilecektir. Ancak KMeans hiçbir zaman buna yönelik bir kümeleme yapamamaktadır.

DBSCAN Kümeleme Yöntemi

DBSCAN (Density-Based Spatial Slustering of Applications with Noise) yoğunluk tabanlı kümeleme yöntemlerinin en çok kullanılanlarından biridir. Algoritma 90'lı yıllarda son haline getirilmiştir. DBSCAN yönteminde yoğunluk (density) en önemli unsurdur. Yoğunluk belli bir alanda bulunan nokta sayısı ile ilgilidir. Yani belli bir alanda çok nokta varsa o alan yoğundur, az nokta varsa o alan yoğun değildir. Yoğunluk için kullanılan alan genellikle daireseldir ve bu dairenin yarı çapı epsilon ile gösterilmektedir.



Şekil <https://towardsdatascience.com/dbscan-algorithm-complete-guide-and-application-with-python-scikit-learn-d690cbae4c5d> makalesinden alınmıştır.

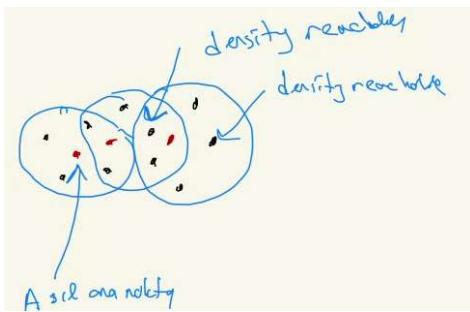
DBSCAN algoritmasında iki önemli parametre vardır: "Yarıçap (epsilon)" ve "en az nokta sayısı (minPts)". Yarıçap yukarıda belirtildiği gibi ilgili noktadan çizilen çemberin yarıçapıdır. En az nokta sayısı ise bir alanın yoğun kabul edilebilmesi için o çemberin içerisinde en az kaç noktanın bulunması gerektiğini belirtir. Örneğin "epsilon = 0.5, minPts = 15" demek, 0.5 yarıçaplı çizilen çemberin içerisinde 15 ya da daha fazla nokta varsa orası yoğun" demektir. Algoritmada yoğunluk nokta temelinde değerlendirilmektedir. Yani bir nokta çemberin merkezi varsayılarak o noktanın oluşturduğu alanın yoğun olup olmadığına bakılmaktadır.

DBSCAN algoritmasında anlatımı kolaylaştırmak amacıyla birkaç kavram kullanılmaktadır. Şimdi bunları açıklayalım:

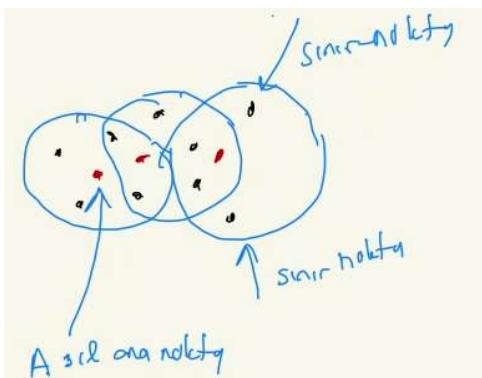
Ana Noktalar (Core Points): Eğer bir noktanın epsilon yarıçaplı çemberi içerisinde minPts ya da daha fazla nokta kalıyorsa, yani o noktanın belirttiği alan yoğun ise böyle noktalara ana noktalar (core points) denilmektedir.

Doğrudan Erişilebilir Noktalar (Direct Reachable Points): Bir ana noktanın oluşturduğu çemberin içerisindeki noktalara o ana noktanın doğrudan erişilebilen noktaları denilmektedir.

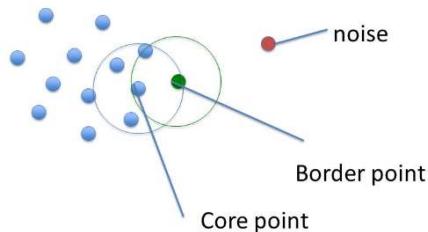
Yoğunluk Yoluyla Erişilebilir Noktalar (Density Reachable Points): Bir ana noktanın doğrudan erişilebilen (yani onun çemberi içerisinde kalan) bir noktası da ana nokta ise o noktanın doğrudan erişilenen noktası asıl noktanın "yoğunluk yoluyla erişilebilen" noktasıdır. Yani yoğunluk yoluyla erişilebilen noktalar "arkadaşımın arkadaşı arkadaşımındır" önermesine benzetilebilir. Tabii geçişlik özelliği devam etmektedir. Yani Ana noktanın doğrudan erişilebilen ana noktasının doğrudan erişilebilen ana noktaları da ilk noktanın yoğunluk yoluyla erişilebilen noktalarıdır. (Yani "arkadaşımın arkasının arkadaşları da benin arkadaşımındır.") Örneğin:



Sınır Noktalar (Border Points): Bir noktanın yoğunluk yoluyla erişilebilen ancak ana nokta olmayan noktalara sınır noktalar (border points) denilmektedir. Buradaki sınır noktalar ismi bu noktaların ilgili kümenin üç noktaları olması nedeniyle verilmiştir. Örneğin:

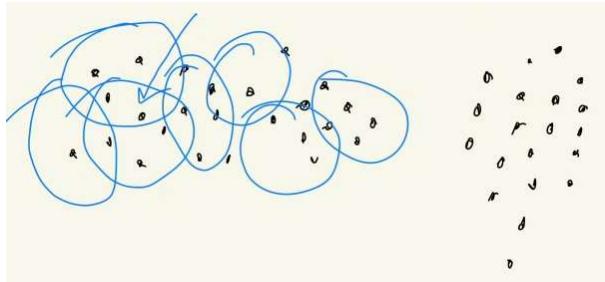


Gürültü Noktaları (Noise Points): Ana nokta ve sınır nokta olmayan noktalara gürültü noktaları denilmektedir. Bu noktalar bir kümenin içerisinde dahil edilmezler ve anomali olarak değerlendirilirler. Anomali tespit yöntemlerinde buna benzer gürültü noktaları üzerinde durulmaktadır. Örneğin:

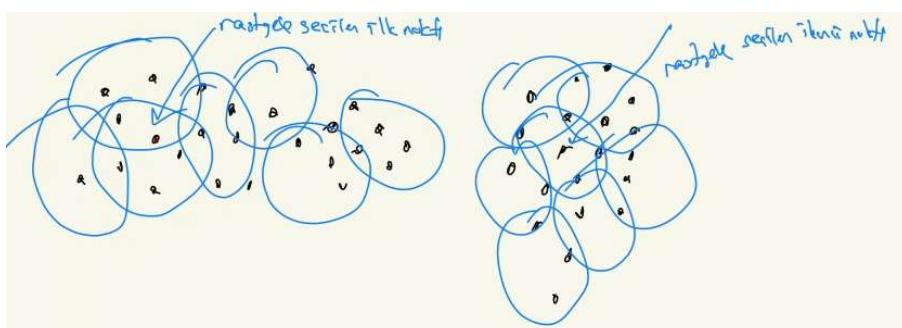


Bu kavramlardan sonra şimdi de DBSCAN algoritmasını açıklayalım. Epsilon ve minPts değerlerinin algoritma girdisi olarak verildiğini varsayıyalım:

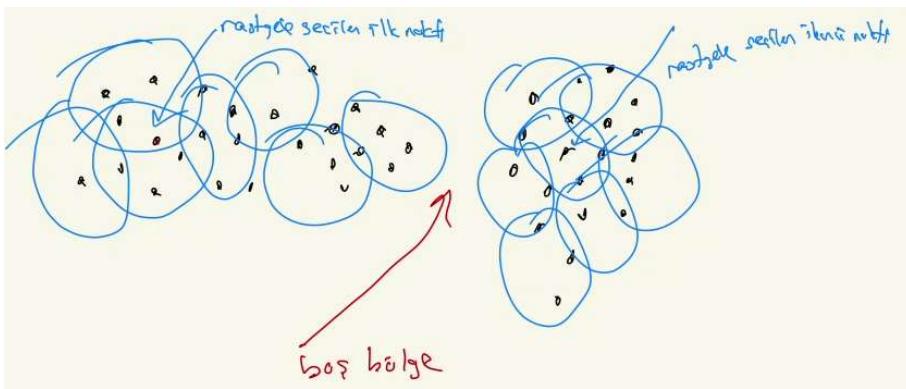
- 1) Kalan noktalar kümesinde (başlangıçta tüm noktalar kalan noktalar kümesindedir) rastgele bir nokta alınır. Bu noktanın ana nokta olup olmadığına (yani etrafının yoğun olup olmadığına) bakılır. Eğer bu nokta ana nokta değilse gürültü noktası biçiminde işaretlenir. Tabii önce gürültü noktası biçiminde işaretlenen bir nokta daha sonda bir kümeye dahil edilebilmektedir.
- 2) Eğer rastgele seçilen nokta bir ana noktaya onun doğrudan erişilebilen (yani onun çemberi içerisinde kalan) noktaları bir kişiye yaratılarak ona dahil edilir. Sonra bu noktalar gözden geçirilir. Doğrudan erişilebilen noktalar içerisinde ana noktalar olup olmadığına bakılır. Doğrudan erişilebilen noktaların içerisindeki ana noktaların doğrudan erişilebilen noktaları da kişiye dahil edilir. Böylece özyineleemeli biçimde işlemler devam ettirilerek ilk kümenin bütün noktaları belirlenmiş olur. Yani artık ilk noktanın yoğunluk yoluyla erişilebilen noktalarındaki bütün ana noktalar kişiye dahil edilmiştir.



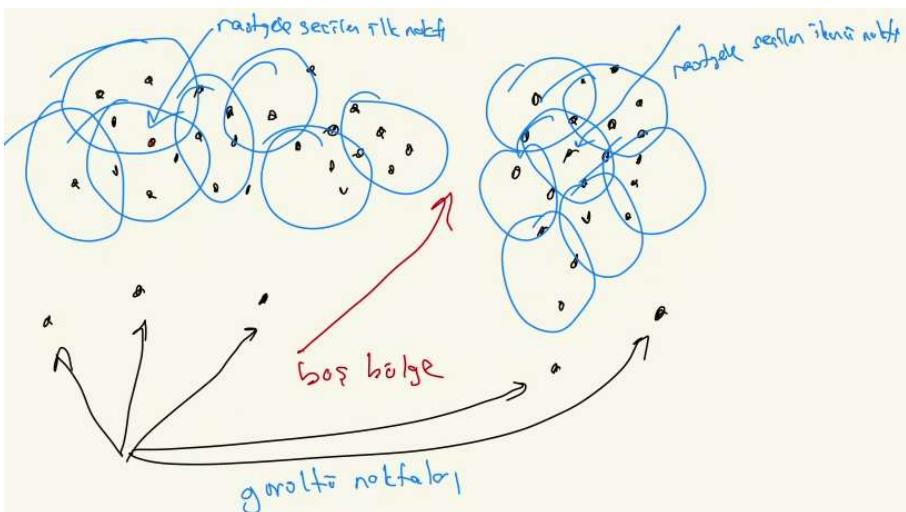
Bu işlem yalnızca bir kümeyi tespit edebilmiştir. Burada algoritma 1'inci adıma geri dönerken bir kümeye sokulmamış ve gürültü olarak işaretlenmemiş noktalar arasından yeni rastgele bir nokta seçer. Aynı şeyleri o nokta için de yapar. Böylece diğer kümeler elde edilmiş olur.



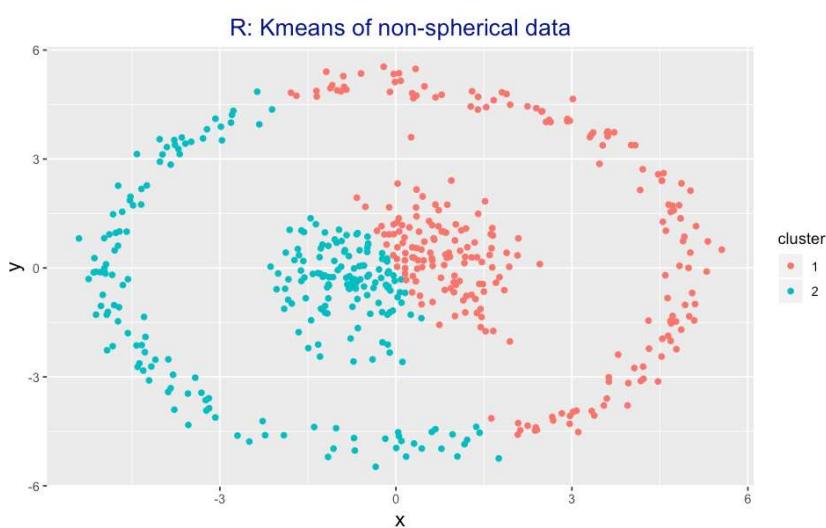
Burada görüldüğü gibi kümelerin birbirlerinden ayrılması için arada boş bir bölgenin olması gereklidir:

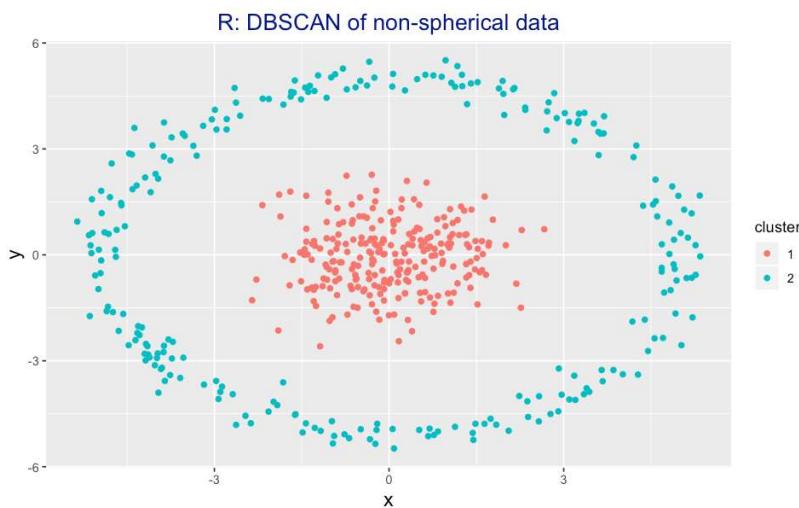


Algoritmanın sonunda hiçbir kümeye dahil edilememiş gürültü noktaları kalabilir. Örneğin:



Burada birkaç noktaya dikkatinizi çekmek istiyoruz. DBSCAN algoritmasında biz algoritmaya küme sayısını vermemekteyiz. Algoritma kümeleri epsilon ve minPts değerlerine dayalı olarak kendisi belirlemektedir. Diğer önemli bir nokta da algoritmanın tüm noktaları kümelendirmediği gürültü noktalarını kümelerin dışında bırakıldığıdır. Bu iki durum daha önce görmüş olduğumuz KMeans ve Agglomerative (hiyerarşik) kğmeleme yöntemlerinden farklıdır. Ayrıca DBSCAN küresel olmayan noktaları da iyi biçimde sınıflandırabilmektedir. Örneğin:





Şekiller https://datascience-enthusiast.com/Python/DBSCAN_Kmeans.html makalesinden alınmıştır.

DBSCAN Kümeleme Yönteminin Scikit-Learn Kütüphanesi Kullanılarak Uygulanması

DBSCAN kümeleme yöntemi scikit-learn kütüphanesindeki DBSCAN isimli sınıfla temsil edilmiştir. Sınıfın `__init__` metodunun parametrik yapısı şöyledir:

```
class sklearn.cluster.DBSCAN(eps=0.5, *, min_samples=5, metric='euclidean', metric_params=None,
                             algorithm='auto', leaf_size=30, p=None, n_jobs=None)
```

Fonksiyonun iki önemli parametresi `eps` ve `min_samples` parametreleridir. `eps` parametresi epsilon değerini, `min_samples` parametresi de minPts değerini belirtir. Bunların default değer aldığına dikkat ediniz. Ayrıca fonksiyonun `metric` parametresi de bir noktanın çember içerisinde kalıp kalmadığını anlamak için kullanılan uzaklık hesap biçimini belirtmektedir. Bu değerin default olarak "euclidean" alındığına dikkat ediniz. Diğer yöntemlerde olduğu gibi burada da DBSCAN nesnesi yaratıldıktan sonra `fit` metoduyla algoritmayı çalıştırmak gereklidir. `fit` metodu kümeleneyecek verileri parametre olarak almaktadır. Kümeleme yapıldıktan sonra hangi noktaların hangi kümelere dahil edildiği yine sınıfın `labels_` isimli özniteliğinden elde edilmektedir. `labels_` özniteliğinde kümeler 0'dan başlanarak numaralandırılmıştır. -1 değeri gürültü noktaları anlamına gelmektedir.

Şimdi "points.csv" dosyasındaki noktaları DBSCAN algoritmasıyla kümelendirelim:

```
import numpy as np

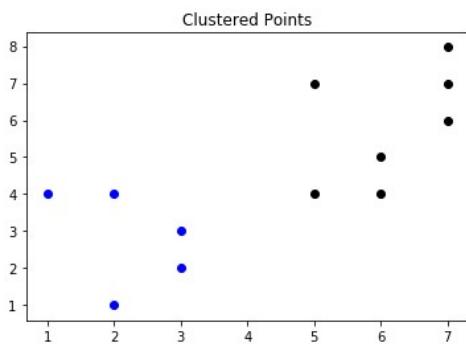
points = np.loadtxt('points.csv', delimiter=',', dtype=np.float32)

from sklearn.cluster import DBSCAN

dbs = DBSCAN(eps=1.5, min_samples=3)
dbs.fit(points)

import matplotlib.pyplot as plt

plt.title('Clustered Points')
plt.scatter(points[dbs.labels_ == 0, 0], points[dbs.labels_ == 0, 1], c='black')
plt.scatter(points[dbs.labels_ == 1, 0], points[dbs.labels_ == 1, 1], c='blue')
plt.scatter(points[dbs.labels_ == 2, 0], points[dbs.labels_ == 2, 1], c='magenta')
```



DBSCAN'in $\text{eps} = 1.5$, $\text{min_sample} = 3$ parametreleriyle noktaları iki kümeye ayırdığını görüyorsunuz.

Şimdi de Iris verileri ile DBSCAN yöntemini kullanalım:

```
from sklearn.datasets import load_iris

iris = load_iris()
dataset_x = iris.data
dataset_y = iris.target

from sklearn.preprocessing import StandardScaler

ss = StandardScaler()
transformed_dataset_x = ss.fit_transform(dataset_x)

from sklearn.cluster import DBSCAN

dbs = DBSCAN(eps=1.5, min_samples=3)
dbs.fit(transformed_dataset_x)

print(dbs.labels_)
```

İşlemler sonucunda elde edilen değerler şöyledir:

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1]
```

Göründüğü gibi DBSCAN epsilon=1.5, minPts=3 parametreleriyle Iris verilerini iki sınıfaya ayırmıştır. Bu parametrelerle hiç gürültü oluşmadığına dikkat ediniz. Şimdi algoritmayı default parametreler olan epsilon=0.5, minPts=5 değerleriyle çalıştırıp sonucuna bakalım:

```
[ 0  0  0  0  0  0  0  0  0  0  0  0  0 -1 -1  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0 -1 -1  0  0  0  0  0 -1  0  0  0  0  0  0  0  0
  0  0  1  1  1  1  1  1 -1 -1  1 -1 -1  1 -1  1  1  1  1  1 -1  1  1
 -1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
 1  1 -1  1 -1  1  1  1  1 -1 -1 -1 -1  1  1  1  1  1 -1  1  1  1 -1 -1
 1  1 -1  1  1 -1  1  1  1 -1 -1 -1  1  1  1  1 -1 -1  1  1  1  1  1  1
 1  1  1  1 -1  1]
```

Çemberi genişletip yoğunluk koşulunu 5'e çektiğimizde gürültü noktalarının arttığını görüyorsunuz. Biz daha önce kullandığımız yöntemlerde Iris verilerini 3 kümeye ayırmıştık. DBSCAN yönteminde küme sayısının belirtilmemiğini epsilon ve minPts değerlerine göre otomatik oluşturulduğunu anımsayınız.

Varyans, Kovaryans ve Korelasyon Kavramları

Varyans istatistikteki en önemli kavramlardan biridir. Genel olarak dağılımdaki değerlerin ortalamadan uzaklığını ölçmek için kullanılır. Örneğin ortalamaları aynı olan değerlerin varyanslarına bakıldığından varyansı yüksek olan daha fazla yayılmıştır. Varyansı düşük olanın değerleri ortalamaya daha yakındır. Ya da örneğin ortalama yıllık gelirin 10000\$ olduğu iki ülkeden hangisinde varyans daha o ülkede gelir adeletsizliği diğerinden daha yüksektir. Varyansın karekökü standart sapmayı verir. Bu anlamda varyans ile standart sapma aynı amaçla kullanılmaktadır. Varyans değerlerin ortalamadan farklarının karelerinin toplamının ortalamasıyla hesaplanır. Varyans formülünde ortalama alınırken genel olarak ana kütle (population) için n 'e örneklem için $n - 1$ 'e bölme uygulanır.

$$\frac{\sum (x_i - \bar{x})^2}{n-1}$$

$$\frac{\sum (x_i - \bar{x})^2}{N}$$

Değerlerin ortalamadan uzaklıklarını hesaplamak için kare aldığımızda değerleri negatiflikten kurtarmış olmaktadır. Kare almak yerine mutlak değer kullanmak da bir yöntem olabilmektedir. Ancak kare almak burada açıklayamayacağımız bazı nedenlerden dolayı çok daha etkin bir yöntemdir.

Varyans ve standart sapma için Python'ın standart kütüphanesindeki statistics modülünün içinde bulunan stddev, variance, pstdev, pvariance fonksiyonları kullanılmaktadır. Bu fonksiyonlar dolaşılabilir herhangi bir nesneyi parametre olarak alabilmektedir. stdev ve variance örneklem için (yeni $n - 1$ bölümü yapmaktadır) pstdev ve pvariance ise ana kütle (population) için (yani N 'ye bölme yapmaktadır) kullanılmaktadır. Örneğin:

```
>>> import statistics
>>> a = [1, 3, 7, 2, 4]
>>> statistics.variance(a)
5.3
>>> statistics.pvariance(a)
4.24
>>> statistics.stdev(a)
2.3021728866442674
>>> statistics.pstdev(a)
2.0591260281974
```

Numpy kütüphanesinde ise varyans ve standart sapma için var ve std fonksiyonları kullanılmaktadır. Bu fonksiyonların parametrelerindeki ddof (delta degrees of freedom) parametresi ile bölümdeki değer belirlenebilmektedir. Bu parametre 0 olarak girilirse (default durum) bölüm N 'ye göre 1 olarak girilirse ($n - 1$)'e göre yapılmaktadır. Örneğin:

```
>>> import numpy as np
>>> a = [1, 3, 7, 2, 4]
>>> np.var(a)
4.24
>>> np.var(a, ddof=1)
5.300000000000001
>>> np.std(a)
2.0591260281974
>>> np.std(a, ddof=1)
2.302172886644268
```

Tabii numpy kütüphanesinde mean, var, std gibi fonksiyonların hepsi çok boyutlu dizilerle de çalışabilmektedir. Bu durumda axis parametresi ile satırsal ya da sütunsul işlem yapılabacağı belirlenir. axis=0, sütunsal, axis=1 satırsal işlem anlamına gelmektedir. Eğer axis değeri None yapılrsa (default durum) bu durumda tüm değerler üzerinde işlemler yapılmaktadır. Örneğin:

```
a = np.array([[1, 2, 3], [4, 5, 2], [5, 9, 1]], dtype=np.float32)
```

```

np.var(a, axis=0)
Out[21]: array([2.888889 , 8.22222 , 0.6666667], dtype=float32)
np.var(a, axis=1)
Out[23]: array([ 0.6666667,  1.5555557, 10.666667 ], dtype=float32)
np.var(a)
Out[24]: 5.8024693

```

Kovaryans birden fazla değişkenin olduğu durumda (multivariate) bu değişkenlerin birbirleri arasındaki varyansı hesaplamak için yani birindeki değişimin diğerini etkileme biçimini incelemek için kullanılmaktadır.

x ve y 'nin kovaryansları olan $\text{Cov}(x, y)$ şu formülle hesaplanmaktadır:

$$\frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{N \text{ (ya da } n-1\text{)}}$$

$\text{Cov}(x, y)$ ile $\text{Cov}(y, x)$ değerlerinin aynı olacağına dikkat ediniz. Ayrıca $\text{Cov}(x, x)$ ile $\text{Var}(x)$ tamamen eşdeğer, benzer biçimde $\text{Cov}(y, y)$ ile $\text{Var}(y)$ değerleri de tamamen aynıdır. Örneğin iki değişken için manuel kovaryans işlemni şöyle yapabiliriz:

```

>>> covxy = np.sum((x - np.mean(x)) * (y - np.mean(y))) / (len(x) - 1)
>>> covxy
24.450000000000003

```

Kovaryans hesabı numpy kütüphanesinde `cov` isimli fonksiyonla yapılmaktadır. Biz `cov` fonksiyonunda iki değişken için ilk iki parametreyi kullanabiliriz. Fakat daha fazla değişken söz konusu olursa birinci parametreyi çok boyutlu dizi biçiminde alabiliriz. `cov` fonksiyonu çok boyutlu dizinin her bir satırını ayrı bir değişken biçiminde ele almaktadır. (Bu fonksiyon `axis` parametresi kullanılmamaktadır.) Birden çok değişken söz konusu olduğunda kovaryans hesabı hepsinin birbirlerine göre ikişerli kovaryanslarını içerecek biçimde matrisel bir hal almaktadır. Örneğin:

x	y	z
5	3	1
7	9	6
2	8	5
1	9	5
3	3	7

Burada kovaryans hesabı $\text{Cov}(x, y)$, $\text{Cov}(x, z)$, $\text{Cov}(y, z)$ biçiminde ayrı ayrı yapılmaktadır. Bu kovaryans değerleri genellikle simetrik bir matris ile ifade edilir. Bu matrise kovaryans matrisi denilmektedir:

$$\begin{bmatrix} \text{cov}(x, x) & \text{cov}(x, y), \text{cov}(x, z) \\ \text{cov}(y, x) & \text{cov}(y, y), \text{cov}(y, z) \\ \text{cov}(z, x) & \text{cov}(z, y), \text{cov}(z, z) \end{bmatrix}$$

İki değişken için kovaryans matrisi de şöyle olacaktır:

$$\begin{bmatrix} \text{Cov}(x,x) & \text{Cov}(x,y) \\ \text{Cov}(y,x) & \text{Cov}(y,y) \end{bmatrix}$$

İşte numpy kütüphanesindeki cov isimli fonksiyon bize ürün olarak kovaryans matrisini vermektedir. Yani başka bir deyişle cov fonksiyonu bize tüm değişkenlerin birbirlerine göre kovaryanslarını bir matris olarak verir. cov fonksiyonu kovaryans hesaplamasında default olarak n - 1'e bölme işlemini uygular. Ancak yine fonksiyonun ddof parametresi 0 yapılarak n'e bölme işlemi uygulanabilir. Örneğin:

```
x = [5, 7, 2, 1, 3]
y = [3, 4, 8, 9, 3]
z = [1, 6, 5, 4, 7]

a = np.array([x, y, z], dtype=np.float32)

cov = np.cov(a)
print(cov)
```

Sonuç şöyledir:

```
[[ 5.8 -5.05 -0.2 ]
 [-5.05  8.3 -0.05]
 [-0.2   -0.05  5.3 ]]
```

Matrisin simetrik olduğuna dikkat ediniz. Örneğin matrisin [0, 3] indeksli elemanı Cov(x, z) değerini vermektedir. Bu değeri şöyle sinyayabiliriz:

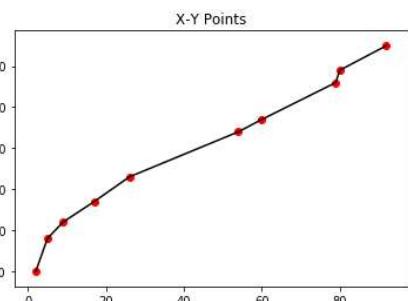
```
result = np.sum((x - np.mean(x)) * (z - np.mean(z))) / (len(x) - 1)
print(result)
```

Kovaryans iki değişkenin birlikte değişimlerinin biçimini anlamakta kullanılıktadır. Yani değişkenlerden birisi artarken diğerinin ne olmaktadır? İki değişkenin kovaryansları pozitif bir değerde ise bunların değişimleri aynı yöndedir. Yani birisi artarken diğer de artmaktadır. Kovaryans negatif ise bunların değişimleri farklı yönlerdedir. Yani biri artarken diğer azalmaktadır. Aşağıdaki x, y noktalarının grafiğini çizelim:

```
x = [2, 5, 9, 17, 26, 54, 60, 79, 80, 92]
y = [10, 18, 22, 27, 33, 44, 47, 56, 59, 65]
```

```
import matplotlib.pyplot as plt

plt.title('X-Y Points')
plt.plot(x, y, c='black')
plt.scatter(x, y, c='red')
```



Şimdi Cov(x,y) değerine bakalım:

```
cov = np.cov([x,y])
print(cov)
```

Sonuç şöyledir:

```
[[1190.93333333 643.4
 [ 643.4 355.21111111]]]
```

Göründüğü gibi $\text{Cov}(x, y) = 643.4$ gibi pozitif bir değerdedir. O halde x artarken y de artmaktadır. Eğer iki değişken arasında artmalı azalmalı bir ilişki yoksa kovaryans değeri 0'a yaklaşmaktadır. Kovaryans değerinin yüksekliği x ve y arasında artış ilişkisinin bağlantısı ile ilgilidir. Ancak bu değer doğrudan yorumlanabilecek bir değer değildir.



Şekil <https://www.geeksforgeeks.org/mathematics-covariance-and-correlation/> makalesinden alınmıştır.

Korelasyon kovaryansın normalize edilmiş bir biçimidir. Korelasyon da tipki kovaryans gibi iki değişken arasındaki ilişkiyi aşağı çıkartır. Ancak korelasyonda değerler -1 ile +1 arasında normalize edilmiştir. Yani bir değişken yükseltirken diğer de benzer biçimde yükseliyorsa bunların arasındaki korelasyon 1'e yaklaşır fakat bir değişken yükseltirken diğer düşüyorsa bunlar arasındaki korelasyon -1'e yaklaşmaktadır. Korelasyonun 0 civarında olması demek iki değişken arasında ilişkinin çok zayıf olması demektir. Tabii iki değişken arasında korelasyon olması bu iki değişkenin neden sonuç ilişkisi içerisinde olduğu anlamına gelmemektedir. Örneğin dondurma yeme miktarı ile boğulma miktarı arasında pozitif bir korelasyon bulunuyor olabilir. Ancak bu durum dondurma yemenin boğulmaya yol açacağı anlamına gelmez. Korelasyon katsayısının hesaplanması için değişik yöntemler kullanılabilmektedir. Ancak en yaygın kullanılan yöntem "Pearson korelasyon katsayı yöntemi"dir. Bu yöntemaslında iki değişkenin kovaryanslarının bu iki değişkenin standart sapmalarının çarpımı bölümü ile hesaplanır.

$$\text{Cor}(x,y) = \frac{\text{Cov}(x,y)}{\text{std}(x) * \text{std}(y)}$$

Numpy'da korelasyon katsayısı corrcoef isimli fonksiyonla hesaplanmaktadır. Bu fonksiyon bize yine matrisel bir sonuç verir. Örneğin:

```
x = [2, 5, 9, 17, 26, 54, 60, 79, 80, 92]
y = [10, 18, 22, 27, 33, 44, 47, 56, 59, 65]
```

```
import matplotlib.pyplot as plt
```

```
plt.title('X-Y Points')
plt.plot(x, y, c='black')
plt.scatter(x, y, c='red')

corr = np.corrcoef([x, y])
print(corr)
```

Burada x ve y'nin korelasyonlarının 0.98 gibi çok yüksek bir değerde olduğunu görüyoruz. Şimdi aynı işlemi kovaryans hesabı ile yapalım:

```

cov = np.cov([x, y])
corr = cov / (np.std(x, ddof=1) * np.std(y, ddof=1))
print(corr)

```

Korelasyon matrisinde matrisin ana köşegenindeki değerinin 1 olmadığına dikkat ediniz. Bu değerlerin 1 olması için bölümde aynı değişkenin standart sapmasının kullanılması gereklidir. Sosyal bilimlerde iki değişkenin arasındaki korelasyon için şunlar söylemektedir:

- 0 - 0.2 ise çok zayıf ilişki ya da korelasyon yok
- 0.2-0.4 arasında ise zayıf korelasyon
- 0.4-0.6 arasında ise orta şiddette korelasyon
- 0.6-0.8 arasında ise yüksek korelasyon
- 0.8 – 1 > ise çok yüksek korelasyon

Doğrusal bir ilişkinin mükemmel bir ilişki olduğunu söyleyebiliriz. Örneğin:

```

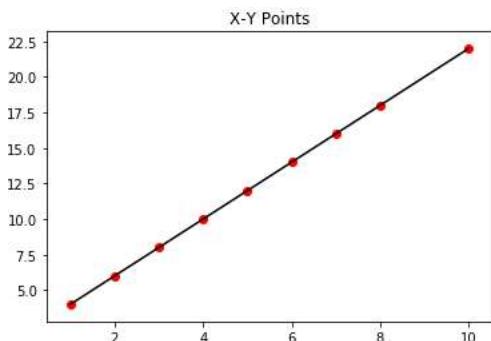
x = np.array([1, 2, 3, 4, 5, 6, 7, 8, 10], dtype=np.float32)
y = 2 * x + 2

import matplotlib.pyplot as plt

plt.title('X-Y Points')
plt.plot(x, y, c='black')
plt.scatter(x, y, c='red')

cor = np.corrcoef([x, y])
print(cor)

```



Elde edilen korelasyon sonucu şöyledir:

```

[[1. 1.]
 [1. 1.]]

```

Örneğin:

```

import numpy as np

x = np.array([3, 5, 8, 3, 17, 19, 3, 2, 10, 21])
y = np.array([2, 11, 12, 19, 20, 5, 6, 16, 4, 9])

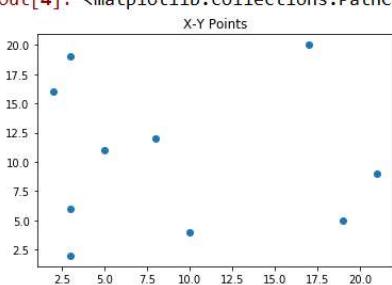
cor = np.corrcoef([x, y])
print(cor)

import matplotlib.pyplot as plt

plt.title('X-Y Points')
plt.scatter(x, y)

```

```
[[ 1.          -0.04398404]
 [-0.04398404  1.          ]]
Out[4]: <matplotlib.collections.PathCollection at 0x23ed6d11bc8>
```



Buradaki grafikten x ve y arasındaki korelasyonun çok düşük olduğu zaten görülmektedir. Örneğin:

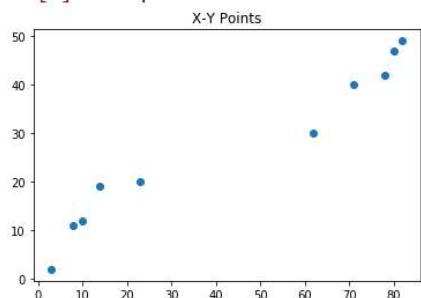
```
x = np.array([3, 8, 10, 14, 23, 62, 71, 78, 80, 82])
y = np.array([2, 11, 12, 19, 20, 30, 40, 42, 47, 49])
```

```
cor = np.corrcoef([x, y])
print(cor)
```

```
import matplotlib.pyplot as plt
```

```
plt.title('X-Y Points')
plt.scatter(x, y)
```

```
[[1.          0.97306728]
 [0.97306728  1.          ]]
Out[5]: <matplotlib.collections.PathCollection at 0x23ed6d88408>
```



Buradaki grafikten de x ve y arasında kuvvetli bir korelasyon olduğu anlaşılmaktadır.

Numpy'da Lineer Cebir İşlemleri

Numpy'da lineer cebir işlemleri için doğrudan numpy içerisindeki bazı fonksiyonlar ve linalg modülündeki fonksiyonlar kullanılmaktadır. Önemli fonksiyonlar şunlardır:

- numpy.dot isimli fonksiyon "dot product" işlemi yapmaktadır. Örneğin:

```
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])
```

```
c = np.dot(a, b)
print(c)
```

- numpy.matmul fonksiyonu matris çarpması yapmaktadır. Örneğin:

```
import numpy as np
```

```
a = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]], dtype='float32')
b = np.array([[1, 1, 2], [2, 2, 3], [4, 5, 2]], dtype='float32')
```

```
c = np.matmul(a, b)
print(c)
```

```
[[17. 20. 14.]
 [38. 44. 35.]
 [59. 68. 56.]]
```

- numpy.linalg.det fonksiyonu determinant hesaplamakta kullanılır. Örneğin:

```
import numpy as np

a = np.array([[1, 20, 3], [4, 50, 6], [7, 12, 9]], dtype='float32')
c = np.linalg.det(a)
print(c)
```

- numpy.linalg.inv fonksiyonu matris tersi almakta kullanılır. Örneğin:

```
import numpy as np

a = np.array([[1, 2, 3], [3, 2, 4], [1, 5, 3]], dtype=np.float32)
b = np.linalg.inv(a)
print(b)
print()
c = np.matmul(a, b)
print(c)

[[ -9.3333334e-01 6.0000002e-01 1.3333334e-01]
 [-3.3333334e-01 1.2810266e-18 3.3333334e-01]
 [ 8.6666667e-01 -2.0000000e-01 -2.6666668e-01]]

[[ 1.0000000e+00 1.4901161e-08 -2.9802322e-08]
 [ 0.0000000e+00 1.0000000e+00 0.0000000e+00]
 [-1.1920929e-07 1.4901161e-08 1.0000000e+00]]
```

- numpy.linalg.solve fonksiyonu lineer denklem sistemini çözmektedir. Örneğin:

```
3x1 + 5x2 - x3 = 10
-x1 - 3x2 + x3 = 8
x1 + x2 + x3 = 6
```

```
import numpy as np

a = np.array([[3, 2, -5], [2, -2, 7], [1, 1, 1]], dtype='float32')
b = np.array([10, 12, 5])
c = np.linalg.solve(a, b)
print(c)

[[1.]
 [2.]
 [3.]]
```

Öz Değerler ve Öz Vektörler (Eigen Values and Eigen Vectors)

Öz değerler ve öz vektörler konusu pek çok alanda kullanılmaktadır. Öz değer ve öz vektör bir kare matrise dayalı olarak hesaplanmaktadır. Hesaplama için genel eşitlik öyledir:

$$AX = \lambda X$$

Burada A ilgili kare matrisi belirtmektedir. X bir sütun vektördür. Bu X vektörüne öz vektör denir. Lambda ise bir skalerdir. Bu lambda skalerine de öz değer denilmektedir. Buradaki eşitliğin daha belirgin gösterimi şöyle yapılabilir:

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1k} \\ a_{21} & a_{22} & \cdots & a_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ a_{k1} & a_{k2} & \cdots & a_{kk} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_k \end{bmatrix} = \lambda \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_k \end{bmatrix},$$

Yukarıdaki eşitlikten şöyle bir anlam çıkmaktadır: Biz bir vektörü (eşitlikteki x) bir matrisle çarptığımızda yine bir vektör elde ederiz. Ancak elde ettiğimiz vektör ilkiyle aynı doğrultuda yalnızca büyülü olarak farklıdır. Yani biz bir vektörü bir matrisle çarptığımızda vektör doğrultu değiştirmemektedir. Yalnızca vektörün büyülüğu değişmektedir. Tabii biz bir vektörü her türlü matrisle çarparsak böyle bir sonuç elde edemeyiz. Bu vektörün matrise uygun seçilmesi gereklidir. İşte bu vektöre matrisin öz vektörü buradaki skaler lambda değerine de matrisin özdegeri denilmektedir. Yalnızca kare matrislerin öz değerleri ve öz vektörleri vardır. Ayrıca her kare matrisin de öz değer ve öz vektörü olmak zorunda değildir. Genel olarak nxn'lik bir kare matrisin n tane öz değeri ve öz vektörü vardır.

Öz değerler ve öz vektörler lineer cebirde aşağıdaki biçimde elde edilmektedir. Bu yöntemde önce Lambda değerlerini (yani öz değerleri) buluruz.

$$\begin{aligned} Ax &= \lambda x \\ \cancel{A}x - \cancel{\lambda}x &= \emptyset \\ (A - \cancel{\lambda})x &= \emptyset \\ \downarrow \\ (A - I\cancel{\lambda})x &= \emptyset \\ A - I\cancel{\lambda} &= \emptyset \end{aligned}$$

Bu denklem aslında A nxn'lik bir kare matris olmak üzere n'inci dereceden bir denklem oluşturur. Bu n'inci dereceden denklemin de n tane kökü vardır. Örneğin aşağıdaki matris için lambda öz değerlerini bulmak isteyelim:

$$A = \begin{bmatrix} 2 & 3 \\ 2 & 1 \end{bmatrix}$$

Örneğin:

$$\begin{bmatrix} 2 & 3 \\ 2 & 1 \end{bmatrix} - \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \emptyset$$

$$\begin{bmatrix} 2-\lambda & 3 \\ 2 & 1-\lambda \end{bmatrix} = \emptyset$$

$$(2-\lambda)(1-\lambda) - 6 = \emptyset$$

$$2 - 2\lambda - \lambda^2 + \lambda^2 - 6 = \emptyset$$

$$\lambda^2 - 3\lambda - 4 = \emptyset$$

$$\lambda = \{-1, 4\}$$

Göründüğü gibi buradan öz değerler 4 ve -1 biçiminde bulunmuştur. Aslında biz ödeğer bulma işlemini numpy.linalg modülündeki eigvals fonksiyonuyla da yapabilirdik:

```
>>> import numpy as np
>>> a = np.array([[2, 3], [2, 1]])
>>> np.linalg.eigvals(a)
array([ 4., -1.])
```

Pekiyi öz vektörler nasıl elde edilmektedir? İşte elimizde öz değerler (yani lamda'lar) varsa biz buradan öz vektörleri elde edebiliriz:

$$\begin{bmatrix} 2 & 3 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \lambda \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$$2x_1 + 3x_2 = \lambda x_1$$

$$2x_1 + x_2 = \lambda x_2$$

$$(2-\lambda)x_1 + 3x_2 = 0$$

$$2x_1 + (1-\lambda)x_2 = 0$$

Şimdi burada lamda yerine 4 koyarak denklemi çözmeye çalışalım:

$$-2x_1 + 3x_2 = 0$$

$$2x_1 - 3x_2 = 0$$

Göründüğü gibi denklemin sonsuz sayıda kökü vardır. Bunlardan birisi $[3, 2]$ kökleridir. İşte $[3, 2]$ vektörü bir öz vektördür. Ama bunun gibi sonsuz sayıda öz vektör vardır. Lineer cebirde bu denklemi sağlayan bütün vektörlere öz vektör uzayı denilmektedir. Fakat pratikte biz bu sonsuz sayıda öz vektörle ilgilenmemiz. Mademki öz vektör bizim için bir doğrultu belirtmektedir. Biz uzunluğu 1 olan normalize edilmiş öz vektörlerle işlemlerimizi yaparız. Bir vektörü normalize etmek için pisagor teoreminden hareketle şu işlemi uygularız:

$$x_1 / \sqrt{x_1^2 + x_2^2}$$

$$x_2 / \sqrt{x_1^2 + x_2^2}$$

Tabii burada nokta sayısı 3 tane olsaydı biz karekök içerisine üçüncü bileşenin de karesini ekleyecektik. Normalize edilmiş değerler şöyledir:

```
>>> 3 / np.sqrt(3 ** 2 + 2 ** 2)
0.8320502943378437
>>> 2 / np.sqrt(3 ** 2 + 2 ** 2)
0.5547001962252291
```

Şimdi aynı işlemi -1 için yapalım:

$$\begin{bmatrix} 2 & 3 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = -1 \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$$2x_1 + 3x_2 = -x_1$$

$$2x_1 + x_2 = -x_2$$

$$\begin{aligned} 3x_1 + 3x_2 &= 0 \\ 2x_1 + 2x_2 &= 0 \end{aligned} \quad \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

$$1\sqrt{2}, -1\sqrt{2}$$

```
>>> 1 / np.sqrt(1 ** 2 + (-1) ** 2)
0.7071067811865475
>>> -1 / np.sqrt(1 ** 2 + (-1) ** 2)
-0.7071067811865475
```

Aslında bu normalize edilmiş öz vektörleri biz Numpy'in linalg modülündeki eig fonksiyonuyla elde edebilmekteyiz. Örneğin:

```
>>> a = np.array([[2, 3], [2, 1]])
>>> np.linalg.eig(a)
(array([ 4., -1.]), array([[ 0.83205029, -0.70710678],
   [ 0.5547002 ,  0.70710678]]))
```

eig fonksiyonun bize verdiği vektörler sütun vektörü biçimindedir. Yani yukarıdaki çözümdeki vektörler aslında şöyledir:

$$\begin{bmatrix} 0.83205029 \\ 0.5547002 \end{bmatrix} \begin{bmatrix} -0.70710678 \\ 0.70710678 \end{bmatrix}$$

Öz vektörler bir birine diktir.

BOYUTSAL ÖZELLİK İNDİRİGEMESİ (Dimensionality Feature Reduction)

Makine öğrenmesindeki veri tabloları sütunlardan oluşmaktadır. Pek çok uygulamada sütun sayısı çok fazla olabilmektedir. Ancak bu sütunlar analiz edildiğinde aslında örneğin bazı sütunun birbirleriyle ilişkili olduğu görülebilmektedir. Örneğin iki sütun söz konusu olsun. Biri diğerinin iki katı değerlere sahip olsun. Aslında öğrenme algoritmaları için bu iki sütunun aynı tabloda bulunması bir kazanç sağlamayacağı gibi "overfitting" gibi, "zaman kaybı", ve "işlem miktarı" gibi dezavantajlar doğurmaktadır. Boyutsal özellik indirgemesi demek veri tablosundaki n tane sütun yerine bu n tane sütunu temsil edebilecek $k < n$ tane sütun oluşturmak demektir. Örneğin üç sütunlu bir tabloda bu üç sütun yerine bu üç sütunu temsil edeceğine inandığımız iki sütun oluşturabiliriz.

Boyutsal özellik indirgemesi aynı zamanda çok sütunlu öğelerin grafiklerinin çizilmesinde de tercih edilmektedir. İnsan olarak bizler iki değişkenli yani iki boyutlu grafikleri güzel bir biçimde algılayabilmekteyiz. Çünkü iki boyutlu grafikler kağıt üzerinde görüntülenebilmektedir. Üç boyutlu grafiklerin kağıt üzerinde görüntülenmesi mümkün olabiliyorsa da çok zahmetlidir. Üç boyuttan yüksek boyuta sahip olan veri tablolarının grafikleri çizilemez. İşte özellik indirgemesi yardımıyla örneğin biz 4 boyutlu (sütunlu) bir veri tablosunu iki boyuta indirgeyip onun grafiğini iki boyutlu olarak çizebiliriz.

Özetlersek boyutsal özellik indirgemesi makine öğrenmesinde şu amaçlarla kullanılmaktadır:

- Veri tablosundaki sütunların (özelliklerin) sayısını azaltarak hesaplamalar için gereken zamanı ve bellek alanını azaltmak yani performans kazancı sağlamak.
- Overfitting durumunu azaltmak
- Veri tablosunun grafiksel olarak görüntülenmesini sağlamak.

Boyutsal veri indirgemesi için pek çok yöntem kullanılabilmektedir. Örneğin:

- Missing Value Ratio
- Low Variance Filter
- High Correlation Filter
- Random Forest
- Backward Feature Elimination
- Forward Feature Selection
- Factor Analysis
- Principal Component Analysis
- Independent Component Analysis
- Methods Based on Projections
- t-Distributed Stochastic Neighbor Embedding (t-SNE)
- UMAP

Bu yönetmelerden en fazla kullanılan temel bileşenler analizi (principal component analysis) dir.

Boyutsal özellik indirgemesi yöntemleri çeşitli bakımlardan gruplandırılabilmektedir. Biz burada bu yöntemleri iki gruba ayıracagız:

- 1) Orijinal n tane sütundan bazı sütunları atarak k tane ($k < n$) sütun elde etmeye çalışan yöntemler.
- 2) Orijinal n tane sütunun yerine bunlarla ilişkili olan yeni k tane sütun ($k < n$) elde etmeye çalışan yöntemler.

Birinci grup yöntemlerde n tane sütundaki bazı sütunlar çeşitli istatistiksel ölçütler temelinde atılmaktadır. Kalan sütunlar orijinal tablodaki k tane sütundur. İkinci grup yöntemlerde n tane sütunun tamamı başka bir k tane sütunla değiştirilmektedir. Örneğin temel bileşenler analizi (principal component analysis) ikinci grup yöntemlere bir

örnektir. Temel bileşenler analizinde biz n tane sütun yerine onları temsil edecek k tane ($k < n$) yeni sütunlar elde ederiz. Şimdi bu iki grup yöntemlerden bazılarını kısaca burada tanıtacağız. Ancak yukarıda da belirtildiği gibi bu alanda en çok tercih edilen yöntem "temel bileşenler analizi (principal component analysis)" yöntemidir. Bu yöntem üzerinde ayrı bir başlıkta daha geniş duracağız.

Eksik Değerli Sütunların Atılması Yöntemi (Missing Value Ratio): Veri tablosundaki bazı sütunlar eksik veriler içerebilmektedir. Örneğin çok kişisel ya da politik birtakım soruların yanıtlarını insanlar vermek istemeyebilirler. Bu durumda bu sorulara ilişkin sütunlarda bilgi eksikliği bulunabilir. İşte bilgi eksikliği belirli bir oranda olan sütunlar tümenden atılıp boyutssal bir özellik indirgemesi oluşturulabilmektedir.

Düşük Varyans Filtrelemesi (Low Variance Filtering): Veri tablosundaki bir sütundaki bilgilerin hep aynı olduğunu düşünelim. Böyle bir sütunun tabloda bulunmasının bir faydası olabilir mi? Tabii ki olmaz. Tüm değerleri aynı olan sütunun varyansı 0'dır. Demekki bir sütunun kestirimde bir faydasının olup olmaması o sütunun varyansıyla da ilgilidir. İşte bu yöntemde bir eşik değeri belirleyip varyansı düşük olan sütunlar tablodan atılabilir. Tabii varyanslar için bir eşik değerinin oluşturulması ayrı bir problemdir. Bu tür durumlarda işlemlerin kolay yürütülebilmesi için önce bir ölçeklendirme (normalizasyon) yapılmamaktadır. Örneğin elimizde aşağıdaki gibi bir data.csv dosyası bulunuyor olsun:

```
10,180,3,1345  
10,100,3,3456  
13,125,2,2340  
12,200,9,5250  
10.01,170,7,1980  
10,160,8,2900  
10,120,5,5200
```

Şimdi biz sütunları MinMax ölçeklendirmesine göre ölçeklendirip varyanslarına bakalım:

```
import numpy as np  
  
dataset = np.loadtxt('data.csv', delimiter=',', dtype=np.float32)  
  
from sklearn.preprocessing import MinMaxScaler  
  
mms = MinMaxScaler()  
  
transformed_dataset = mms.fit_transform(dataset)  
col_vars = np.var(transformed_dataset, axis=0)  
print(col_vars)
```

Aşağıdaki gibi bir sonuç elde edilmiştir:

```
[0.14943445 0.11316326 0.13244483 0.1314027 ]
```

Burada atılmaaya en uygun sütun birinci indeksli en küçük varyansa sahip sütundur. Tabii iki sütun atmak isersek birinci ve üçüncü indeksli sütunları atmalıyız. Örneğin:

```
reduced_dataset = np.delete(transformed_dataset, [1, 3], axis=1)  
print(reduced_dataset)  
  
[[0. 0.14285716]  
[0. 0.14285716]  
[1. 0.]  
[0.66666665 1.0000001]  
[0.00333333 0.71428573]  
[0. 0.8571429]  
[0. 0.42857143]]
```

Bu yöntem scikit-learn kütüphanesinde VarianceThreshold isimli sınıf ile gerçekleştirilmiştir. Bu fonksiyon 0 ile 1 arasında elimine edilecek sütunların düşük varyans limitini parametre olarak almaktadır. Daha sonra elde edilen sınıf nesnesi ile fit ve transform işlemleri yapılabilir. (Bu işlemler tek adımda fit_transform metoduyla da yapılabilmektedir.) Örneğin:

```
import numpy as np

dataset = np.loadtxt('data.csv', delimiter=',', dtype=np.float32)
from sklearn.preprocessing import MinMaxScaler

mms = MinMaxScaler()
transformed_dataset = mms.fit_transform(dataset)

from sklearn.feature_selection import VarianceThreshold

vt = VarianceThreshold(0.13)
reduced_dataset = vt.fit_transform(transformed_dataset)
print(vt.variances_)
print(reduced_dataset)

[0.14943445 0.11316326 0.13244484 0.1314027 ]
[[0.          0.14285716 0.          ]
 [0.          0.14285716 0.540589  ]
 [1.          0.          0.25480154]
 [0.66666665 1.0000001  1.          ]
 [0.00333333 0.71428573 0.16261205]
 [0.          0.8571429  0.39820746]
 [0.          0.42857143 0.98719597]]
```

VarianceThreshold sınıfının kendisi ölçeklendirme yapmamaktadır. Dolayısıyla düşük varyanslı sütunların karşılaştırılmış olarak elimine edilmesi için bizim ölçeklendirme yaptıktan sonra bu sınıfı kullanmamız gereklidir.

Yüksek Korelasyon Filtrelemesi Yöntemi (High Correlation Filtering): Yüksek korelasyona sahip iki sütunun aynı tabloda bulunması makine öğrenmesi modellerinde önemli bir fayda sağlamamaktadır. Örneğin bir sütun diğer sütunun iki katından dört fazla olsun. Bu durumda bu iki sütun arasında doğrusal bir ilişki vardır. Dolayısıyla bu iki sütunun korelasyon katsayısı 1'dir. Bu iki sütunun tablomuzda bir arada bulunmasının kestirimler için bir faydası yoktur. Bunlardan biri tablodan atılabilir. Yüksek korelasyon demekle ne kastedildiği veri bilimcисine bağlıdır. Genellikle burada kastedilen 0.90 ve yukarısıdır. Bu yöntemi Python'da şöyle uygulayabiliriz. Aşağıdaki gibi bir "data.csv" dosyası bulunuyor olsun:

```
10,180,3,302,20,21
10,1790,3,205,20.1,20.2
26,600,2,200,26.2,53
12,1925,9,920,24.1,25
10.1,192,7,710,20.100,20.3
18,28,8,820,20,27
10,890,5,520,20,21
```

```
import numpy as np

dataset = np.loadtxt('data.csv', delimiter=',', dtype=np.float32)
corr = np.corrcoef(dataset, rowvar=False)

reduced_features = []

for i in range(corr.shape[0]):
    for k in range(corr.shape[1]):
        if i != k and i not in reduced_features and np.abs(corr[i, k]) > 0.90:
            reduced_features.append(i)
```

```

print(reduced_features)

reduced_dataset = np.zeros((dataset.shape[0], len(reduced_features)))

for i, col in enumerate(reduced_features):
    reduced_dataset[:, i] = dataset[:, col]

print(reduced_dataset)

```

Burada reduced_features listesi içerisindeki sütun numaraları indirgenmiş olan sütunları belirtmektedir. Yani biz buradaki reduced_features sütunlarını alıp bunlardan yeni bir dataset oluşturmalıyız. Yukarıdaki programdan elde edilen sonuçlar şöyledir:

```

[0, 2, 3, 5]
[[ 10.      3.      302.      21.      ]
 [ 10.      3.      205.      20.20000076]
 [ 26.      2.      200.      53.      ]
 [ 12.      9.      920.      25.      ]
 [ 10.10000038 7.      710.      20.29999924]
 [ 18.      8.      820.      27.      ]
 [ 10.      5.      520.      21.      ]]

```

Göründüğü gibi orijinal dataset'teki iki sütun diğer iki sütunla yüksek bir korelasyona sahip olduğu için atılmıştır. scikit-learn kütüphanesinde doğrudan yüksek varyans indirgemesini yapan bir sınıf ya da fonksiyon bulunmamaktadır.

Rassal Ormalar Yöntemi (Random Forests): Rassal ormanlar yöntemi kursumuzda ilerideki konularda ele alınmaktadır.

Geriye Doğru Özellik İndirgemesi Yöntemi (Backward Feature Elimination): Bu yöntem eğitimli öğrenme modellerinde uygulanabilmektedir. Bu yöntemde n tane sütun ile modelin başarısı test edilir. Sonra sütunlardan bir tanesi atılarak yeniden modelin başarısına bakılır. Eğer sütun atıldıktan sonra modelin başarısı çok değişmemişse bu sütunun gerçekten atılabilecek bir sütun olduğunu karar verilir. Bu biçimde tüm sütunlar gözden geçirilir. Tabii bu yöntem çok uzun bir bilgisayar zamanına mal olmaktadır. Bu nedenle pratikte pek tercih edilmemektedir.

İleriye Doğru Özellik İndirgemesi Yöntemi (Forward Feature Elimination): Bu yöntem geriye doğru özellik indirgemesi yönteminin tersidir. Yani modelin başarısı tek bir sütunla başlatılır. Sonra başarıyı dah çok yükselten sütun dahil edilir. Tabii yöntem de eğitimli öğrenmede kullanılabilecek bir yöntemdir. Yine bu yöntem de yüksek bir bilgisayar zamanına yol açmaktadır.

Faktör Analizi Yöntemi (Factor Analysis): Faktör analizi istatistikte özellikle sosyal bilimlerde çok kullanılan yöntemlerden biridir. Faktör analizinde bir olguya açıklayabilecek n tane özellik aralarındaki kovaryanslara göre k tane özelliğe indirgenmektedir. Yani birbirlerine benzeyen özellikler atılarak onlar yerine o özellikleri temsil eden bir özellik sınıfı oluşturulmaktadır. Faktör analizi ayrı ve önemli bir konudur. Temel Bileşenler Analizi yöntemi de aslında bir bakıma faktör analizi yöntemi olarak değerlendirilebilmektedir.

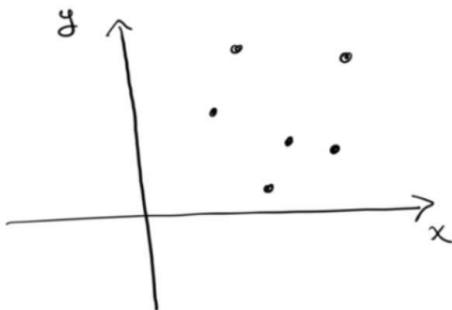
Biz kursumuzda boyutsal özellik indirgemesi yöntemi olarak "temel bileşenler analizi (principal component analysis) üzerinde ağırlıklı biçimde duracağız. Bu yöntem izleyen başlıkta açıklanmaktadır.

Temel Bileşenler Analizi (Principal Component Analysis)

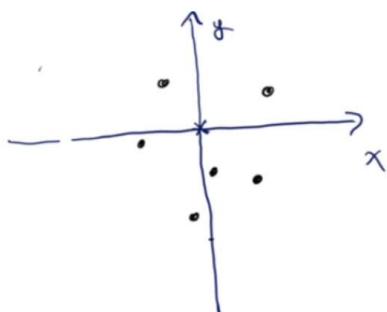
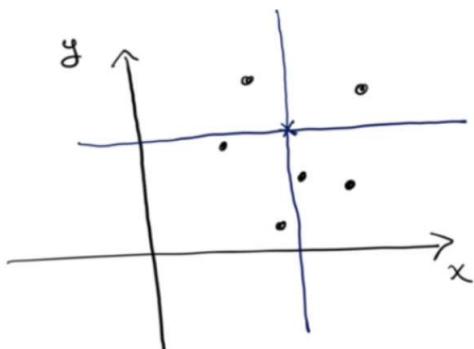
Yukarıda da belirtildiği gibi boyutsal özellik indirgemesi (feature reduction) için en yaygın kullanılan yöntem "temel bileşenler analizi"dir. Numpy'da temel bileşenler analizini uygulayan yetenekli bir sınıf vardır. Bu sınıf sayesinde aslında veri bilimcisi burada açıklanacak manuel hesapları yapmak zorunda kalmaz. Ancak biz burada temel bileşen analizinin manuel bir biçimde nasıl yapıldığı üzerinde biraz duracağız.

Örneğin iki sütunlu bir veri tablomuz olsun ve bu veri tablosunu tek sütuna indirmek isteyelim. Bu işlemi nasıl yapabiliriz? Buradaki temel mantık iki sütunlu (yani iki bileşene sahip) veri yerine onu temsil edebilecek tek sütunlu

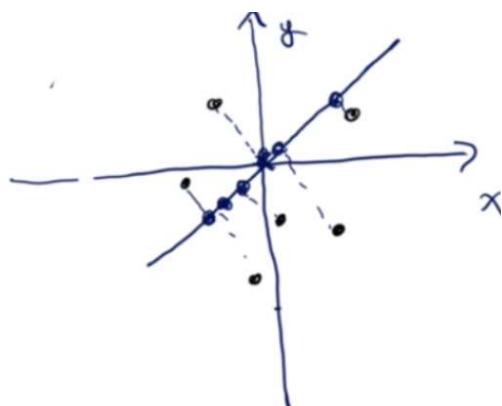
(yani tek bileşene sahip) bir veri tablosu oluşturmaktır. Bu yapılrken de orijinal noktaların yeni eksendeki projeksiyonlarının varyansının yüksek tutulması gerekmektedir. Örneğin:



Biz burada x ve y değerlerine sahip iki sütunlu bilgilerin nokta grafiğini çizdik. Amacımız da bu iki sütunlu bilgi yerine tek sütunlu (yani tek eksenli) değerleri oluşturmak. Bunun için uygun bir eksenin seçilmesi gereklidir. İlk yapılacak şey eksenlerin orta noktaya ötelebilmesidir. Bu ötemele sonucunda grafiğin şu hale geldiğini varsayıyalım:



Bundan sonra iki boyutlu noktaların tek boyutlu eksende oluşturulması için (buna "noktaların projekte edilmesi" denilmektedir) uygun bir eksenin seçilmesi gereklidir. Orijin noktasından geçen sonsuz sayıda eksen olabilir. Bizim seçeceğimiz eksenin söyle bir özelliği olmalıdır: Noktalar bu eksene projekte edildiğinde toplam varyansların en yüksek olması istenir. Yani noktaların orijine olan uzaklıklarını toplamının maksimum olması arzu edilmektedir. Böylece iki boyutlu bilginin daha az kayıpla tek boyuta indirgenmesi sağlanmış olur. Aşağıda böyle bir eksen çizilmiştir:



Pekiyi böyle bir doğrunun denklemi nedir ve projeksiyon bu doğru üzerine nasıl yapılacaktır? İşte bu işlemler daha önce görmüş olduğumuz kovaryans matrisi ve öz değer vektörleri kullanılarak yapılmaktadır.

Örneğin N tane sütuna sahip bir veri tablosundan k tane sütuna sahip ($k < n$) bir veri tablosunu temel bileşenler analizi yöntemiyle elde etmek isteyelim. İşlem adımları şöyle gerçekleştirilir:

- 1) N sütunlu veriler üzerinde özellik ölçeklendirmesi (normalizasyon) uygulanır.
- 2) N sütunlu matristen $N \times N$ 'lik kovaryans matrisi elde edilir.
- 3) Bu kovaryans matrisinden N tane öz vektör bulunur.
- 4) Bu N tane özvektör arasından k tane'si seçilerek (seçimin nasıl yapılacağı belirtilecektir) asıl matrisle çarpılır ve böylece sonuçta k tane sütuna indirgenmiş veri tablosu elde edilir.

Şimdi bu işlemleri adım adım Python'da yapalım. Bu örneğimizde iki sütunlu tabloyu tek sütuna indirmeye çalışalım. İki sutunlu tablonun bilgileri şöyle olsun:

x1	x2
0.72	0.13
0.18	0.23
2.5	2.3
0.45	0.16
0.04	0.44
0.13	0.24
0.30	0.03
2.65	2.1
0.91	0.91
0.46	0.32

Bu bilgiler "data.csv" dosyası içerisinde bulunmaktadır.

Şimdi ilk yapılacak şey normalize etmek yani orijin noktasını değerlerin ortasına kaydurmaktır. Bu işlem şöyle yapılabilir:

```
import numpy as np

dataset = np.loadtxt('data.csv', delimiter=',', dtype=np.float32)
dataset = dataset - np.mean(dataset, axis=0)
```

Şimdi bu normalize edilmiş 2 boyutlu tablodan 2×2 'lik kovaryans matrisi elde edilir:

```
cmat = np.cov(dataset, rowvar=False)
print(cmat)
```

Elde edilen covaryans matrisi şöyledir:

```
array([[0.91316003, 0.75931776],
       [0.75931776, 0.69689329]])
```

Bundan sonra bu kovaryans matrisinin öz değerleri ve öz vektörleri bulunur:

```
evals, evecs = np.linalg.eig(cmat)
print(evals)
print(evecs)
```

Elde edilen öz değerler ve öz vektörler şöyledir:

```
[1.57200534 0.03804799]
[[ 0.75530992 -0.65536778
 [ 0.65536778  0.75530992]]]
```

Şimdi bizim projeksiyon işlemini yapmamız gereklidir. Biz asıl ölçeklendirilmiş asıl matrisimizi (biz bu örnekte ölçeklendirme yapmadık, çünkü gerekmedi) $n \times n$ 'lik özvektör matrisinin k tanesiyle çarptığımızda artık k tane sütunlu bir matris elde ederiz. Buradaki amacımız iki sütunu tek sütuna indirgemektir. Demek ki biz tek bir özvektörlü çarpma yaparak tek sütunumuzu elde edeceğiz. Pekiyi n tane öz vektör arasından hangi k tane özvektörü bu çarpma işlemine sokmalıyız? İşte öz değeri yüksek vektörlerin bu işlem için seçilmesi gerekmektedir. Çünkü öz değeri yüksek olan öz vektörler daha yüksek varyansa yol açmaktadır. Yukarıdaki örneğimizde birinci özvektör (1.57 olan) diğerinden daha yüksektir. O halde biz birinci öz vektörü asıl matrisle çarpmalıyız:

```
reduced_dataset = np.matmul(dataset, evecs[:, 0].reshape((-1, 1)))
```

Elde edilen değerler şöyledir:

```
[[ -0.45048978
 [-0.79282036
 [ 2.3161099 ]
 [-0.63476248
 [-0.76093652
 [-0.8240322 ]
 [-0.83325676
 [ 2.29833288
 [ 0.20420599
 [-0.5223505 ]]
```

Aslında temel bileşenler analizi scikit-learn kütüphanesinde decomposition modülündeki PCA sınıfıyla kolay bir biçimde yapılmaktadır. PCA sınıfının `__init__` metodunun parametrik yapısı şöyledir:

```
sklearn.decomposition.PCA(n_components=None, copy=True, whiten=False, svd_solver='auto', tol=0.0, iterated_power='auto', random_state=None)
```

Buradaki `n_components` indirgenme sonucunda elde edilecek sütun sayısını belirtmektedir. Diğer parametrelerin çok önemi yoktur.

PCA sınıfının `fit` metodu gerçek hesaplamayı yapmaktadır. Veri tablosu bu metoda parametre olarak girilir. Ancak `fit` metodu indirgenmiş değerleri bize vermez. (Bu metot PCA nesnesinin yine kendisini bize vermektedir.) İndirgenmiş değerleri almak için ayrıca PCA sınıfının `transform` metodunun çağrılmaması gerekmektedir. Fakat `fit_transform` isimli metot bu ikisini zaten bir arada yapmaktadır. `inverse_transform` metodu ise işlemin tersini yapmaktadır. Yani bu metot az sütunlu bir tabloyu çok sütunlu hale getirmektedir. Bu işlemin amaçsız olduğu düşünülebilir. Ancak bazı uygulamalarda (örneğin anomalileri tespit etme uygulamalarında) bu metot kullanılmaktadır. Sınıfın iki önemli özniteligi `explained_variance_` ve `explained_varience_ratio` öznitelikleridir. Bu öznitelikler belirlenen sütun sayısına göre asıl tablonun varyansının ne kadarının indirgenmiş tabloya yansıtıldığını belirtmektedir. `explained_variance_` bir değer olarak bunu verirken `explained_variance_ratio` bir yüzde olarak bunu vermektedir.

Şimdi yukarıda elle yaptığımız örneği PCA sınıfı ile yapalım:

```
import numpy as np

dataset = np.loadtxt('data.csv', delimiter=',', dtype=np.float32)
print(dataset, '\n')

from sklearn.decomposition import PCA

pca = PCA(n_components=1)
reduced_dataset = pca.fit_transform(dataset)
print(reduced_dataset, '\n')
```

```
print('Explained variance: {}'.format(pca.explained_variance_))
print('Explained variance ratio: {}'.format(pca.explained_variance_ratio_))
```

Burada elde edilen sonuçlar şöyledir:

```
[[0.72 0.13]
 [0.18 0.23]
 [2.5  2.3 ]
 [0.45 0.16]
 [0.04 0.44]
 [0.13 0.24]
 [0.3  0.03]
 [2.65 2.1 ]
 [0.91 0.91]
 [0.46 0.32]]

[[-0.45048997]
 [-0.7928204 ]
 [ 2.31611  ]
 [-0.6347626 ]
 [-0.7609365 ]
 [-0.82403225]
 [-0.8332568 ]
 [ 2.298333  ]
 [ 0.20420602]
 [-0.52235055]]
```

```
Explained variance: [1.572005]
Explained variance ratio: [0.9763685]
```

Açıklanan varyans oranının %97 civarında olduğu görülmektedir. Bu %97 değeri kabaca şu anlamda gelmektedir: Biz iki sütunlu bu tabloyu tek sütuna indirdiğimizde yalnızca %3'lük bir kayıp oluşturduk. Tabii şüphesiz indirgeme sonrasında elde edilecek sütun sayısı azaltıldıkça bu oran düşecektir. Bu oranın iyi bir noktada kalacağı biçimde asıl tablonun kaç sütuna indirgenmesi gereği izleyen başlıkta ele alınmaktadır.

Anımsanacağı gibi boyutsal özellik indirgemesi çok boyutlu olguların iki ya da üç boyutta indirgenerek grafiklerinin çizilmesi için de kullanılıyordu. Şimdi biz Iris veritabanında sınıflandırma sonucunda elde edilen 4 bsütunlu verileri iki sütuna indirgerek grafiğini çizelim:

```
import numpy as np

from sklearn.datasets import load_iris

iris = load_iris()

dataset_x = iris.data
dataset_y = iris.target

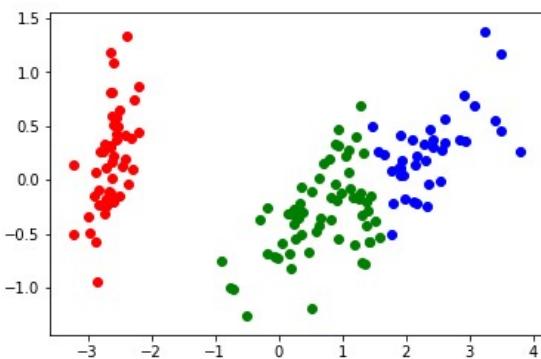
from sklearn.cluster import KMeans

km = KMeans(n_clusters=3)
km.fit(dataset_x)

from sklearn.decomposition import PCA
pca = PCA(n_components=2)
reduced_dataset_x = pca.fit_transform(dataset_x)

import matplotlib.pyplot as plt

plt.scatter(reduced_dataset_x[km.labels_ == 0][:, 0], reduced_dataset_x[km.labels_ == 0][:, 1],
color='red')
plt.scatter(reduced_dataset_x[km.labels_ == 1][:, 0], reduced_dataset_x[km.labels_ == 1][:, 1],
color='green')
plt.scatter(reduced_dataset_x[km.labels_ == 2][:, 0], reduced_dataset_x[km.labels_ == 2][:, 1],
color='blue')
```



Biz bu örnekte aslında 4 özelliğe (sütuna) sahip olan Iris verilerini 3 kümeye ayırdık. 4 özelliğin grafiğini çizemeyeceğimizden dolayı onu PCA yöntemiyle iki özelliğe indirgeyip grafiğini çizdik.

PCA İşleminde İndirgenecek Sütun Sayısının Belirlenmesi

Elimizde n sütunlu bir veri tablosu olsun. Biz n sütunu k sütuna indirmek isteyelim. Peki bu k değeri ne olmalıdır? Yani örneğin elimizde 100 sütunlu bir veri tablosu varsa biz bunu kaç sütuna önemli bir kayıp olmadan indirgemeliyiz? Şüphesiz sezgisel olarak bu 100 sütunun 99 sütuna indirgenmesi ile 10 sütuna indirgenmesi arasında bir farklılık olduğu anlaşılabılır. Biz kabaca indirgeme ne kadar yüksek yapılıyorsa orijinal verilerin o kadar bozulacağını söyleyebiliriz. O halde "optimum indirgeme için sütun sayısı ne olmalıdır" sorusu gündeme gelmektedir. İşte bunu belirleyebilmek için iki yöntem önerilmektedir:

1) Açıklanan varyans oranının en çok düşüğü sütun sayısını bulmak. Yani eğrinin eğiminin atay eksene stabil olarak kaldığı noktanın başlangıcı bulmak.

2) Açıklanan varyansın kabaca arzu edilen yüzdede olduğu (örneğin %80) sütun sayısını bulmak.

Birinci yöntem büyük ölçüde gözle kontrol edebilecek bir yöntemdir. İkinci yöntemi programlamak çok kolaydır. İkinci yöntemdeki yüzde değeri duruma göre değiştirilebilir. Örneğin yüzde değeri %80 alınırsa bu durum "orijinal verilerdeki bozulmanın %20 düzeyinde kaldığı en çok sütunlu durumun elde edileceği" anlamına gelmektedir. Şimdi biz bu yöntemleri Mnist verilerinde kullanmaya çalışalım.

```
from tensorflow.keras.datasets import mnist
(training_dataset_x, training_dataset_y), (test_dataset_x, test_dataset_y) = mnist.load_data()
dataset_x = training_dataset_x.reshape(-1, 28 * 28)
dataset_x = dataset_x / 255

import numpy as np
from sklearn.decomposition import PCA

ratios = []
optimal_feature = None
for i in range(100):
    pca = PCA(n_components=i)
    pca.fit(dataset_x)
    ratios.append(np.sum(pca.explained_variance_ratio_))
    if not optimal_feature and ratios[-1] >= 0.80:
        optimal_feature = i
    print('{}---> {}'.format(i, ratios[-1]))

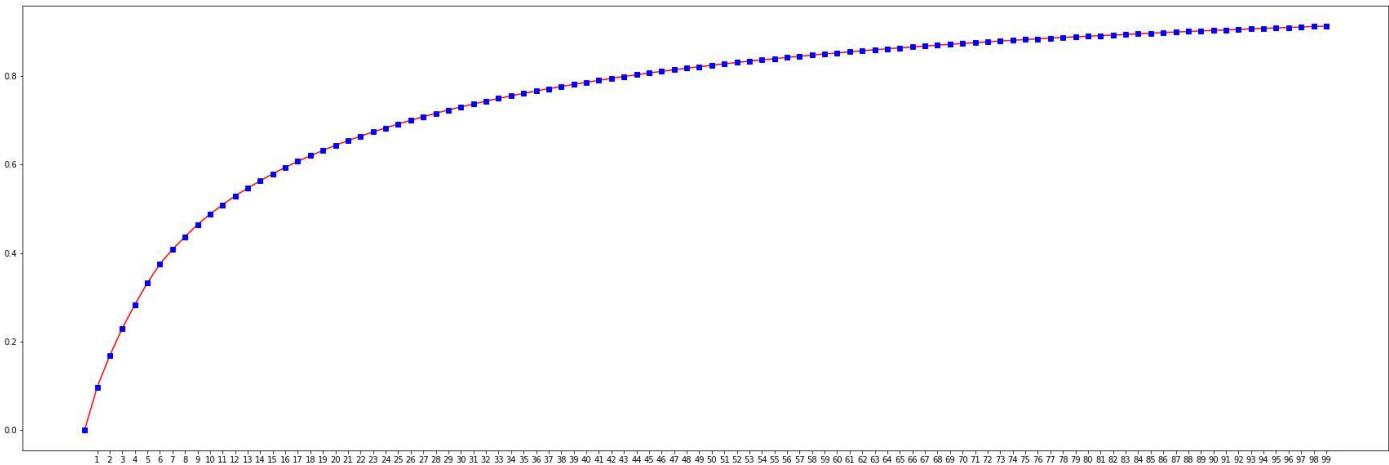
import matplotlib.pyplot as plt
plt.figure(figsize=(30, 10))
plt.plot(ratios, 'r')
plt.plot(ratios, 'bs')
plt.xticks(range(1, 100))
```

```
plt.pause(1)  
print(optimal_feature)
```

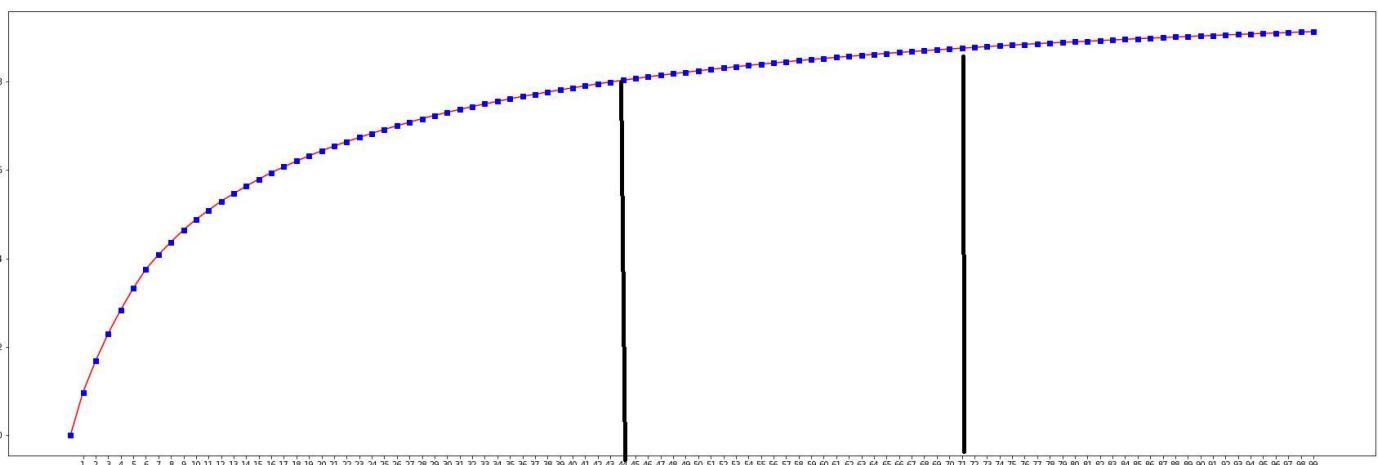
Bu programdan elde edilen veriler aşağıdaki gibidir:

```
0---> 0.0  
1---> 0.09704664359411821  
2---> 0.16800588405555902  
3---> 0.22969677170660122  
4---> 0.28359096140493656  
5---> 0.33227893102047446  
6---> 0.3754012485672624  
.....  
25---> 0.6917966534434756  
26---> 0.7001939613296784  
27---> 0.7083190671384518  
28---> 0.7161844162745432  
29---> 0.723623901698142  
30---> 0.7305381991707998  
31---> 0.737103904740151  
.....  
43---> 0.7993004114124839  
44---> 0.8032580490108351  
45---> 0.8071143109151905  
46---> 0.8108497261629446  
47---> 0.81449252518249  
48---> 0.8180094304518465  
.....  
59---> 0.850151959089285  
60---> 0.8525806719701093  
61---> 0.8550143536639044  
62---> 0.8573916039952458  
63---> 0.8597069516137059  
64---> 0.8619171933388575  
.....  
80---> 0.8902214119703338  
81---> 0.8916782708014838  
82---> 0.892996569814878  
83---> 0.8943441647586491  
84---> 0.8957344823217974  
85---> 0.8972051043024051  
86---> 0.8985277299960197  
.....  
95---> 0.9091421162637744  
96---> 0.9101542304308556  
97---> 0.9111446067892256  
98---> 0.9123344433204897  
99---> 0.9132516893488988
```

Burada %80 varyans için elde edilen sütun sayısı 44 olarak bulunmuştur. Elde edilen grafik de şöyledir:



Gözle tespitin yapıldığı birinci yöntem bu grafikte nasıl uygulanabilir? Grafikte artık kazancın iyici düşüğü ve eğrinin eğiminin yatay eksene göre stabil hale geldiği nokta gözle tespit edilebilir. Aşağıdaki çizimde gözle tespit edilen yer ile %80'lik yer gösterilmiştir:



Aslında optimal sütun sayısının elde edilmesi için şöyle bir fonksiyon da yazabiliriz:

```
def optimal_reduction(x, ratio=0.80):
    for i in range(1, x.shape[1]):
        pca = PCA(n_components=i)
        pca.fit(x)
        if np.sum(pca.explained_variance_ratio_) >= ratio:
            return i
    return x.shape[1]

result = optimal_reduction(dataset_x)
print(result)
```

MNIST Örneğinde PCA İşleminden sonra K-Means Kümeleme Yönteminin Kullanılması

PCA bir önişlem (preprocessing) faaliyeti olarak değerlendirilebilir. Yani biz sütun indirmemesi yaptıktan sonra başka diğer işlemlere geçebiliriz. Burada Mnist verileri üzerinde PCA işleminden sonra K-Means kümelemesi yapacağız.

```
import numpy as np
from tensorflow.keras.datasets import mnist

(training_dataset_x, training_dataset_y), (test_dataset_x, test_dataset_y) = mnist.load_data()

dataset_x = training_dataset_x.reshape(-1, 28 * 28).astype(np.float32)
dataset_x = dataset_x / 255.0
```

```
from sklearn.decomposition import PCA  
  
pca = PCA(n_components=44)  
reduced_dataset_x = pca.fit_transform(dataset_x)  
  
from sklearn.cluster import KMeans  
  
kmeans = KMeans(n_clusters=10)  
kmeans.fit(reduced_dataset_x)  
print(kmeans.labels_)
```

ANOMALİLERİN TESPİT EDİLMESİ (ANOMALY DETECTION)

Denetimsiz makine öğrenmesinde popüler konulardan biri de anomalilerin tespit edilmesidir. Bu sayede hiçbir niteliksel bilgiye sahip olmadan hileli işlemleri ve verileri diğerlerinden belli bir güvenilirlikle ayırt edebilmekteyiz. Dolayısıyla anomalilerin tespit edilmesi şüpheli işlemler, bilgisayar güvenliği, bilgisayar virüsleri, kötü niyetli yazılımların belirlenmesi gibi işlemlerde kullanılabilmektedir. Anomalilerin tespit edilmesi anlamına gelen İngilizce pek çok terim uydurulmuştur. Örneğin: "Anomaly detection", "outliers", "novelties", "noise", "deviations", "exceptions".

Anomalilerin tespit edilmesi için pek çok yöntem kullanılmaktadır. Bunlardan bazıları şunlardır:

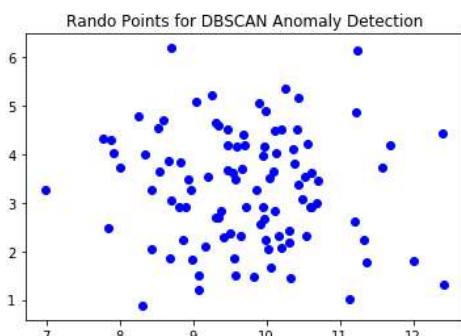
- Yoğunluk Tabanlı Yöntemler (Isolation Forest, K-Nearest Neighbor, vs.)
- Destek Vektör makineleri (support vector machines)
- Bayesian Networks
- Saklı Markov Modelleri (Hidden Markov Models)
- Kümeleme Esasına Dayanan Yöntemler (K-Means, DBSCAN vs.)
- Bulanık Mantık Kullanılan Yöntemler

DBSCAN Kümeleme Yöntemiyle Anomalilerin Tespit Edilmesi

Anımsanacağı gibi DBSCAN yönteminde yoğunluk esas alınmaktadır. Böylece bu yöntem küresel kümelerin dışında örneğin eliptik gibi diğer kümeleri de tespit edebiliyordu. DBSCAN yönteminde noktaların ilişkilendirileceği kume sayısını biz vermiyoruz. Bu kume sayısı zaten algoritmanın işleyışı sırasında otomatik olarak belirleniyordu. Yine bu yöntemde bazı noktalar hiçbir kümeye dahil edilemiyordu. Bu noktalara biz "gürültü (noise)" diyoruz. İşte DBSCAN yöntemiyle anomali tespitinde uygulamacı sanki bir kümeleme işlemi yapıyormuş gibi işlemlerini yürütür. Gürültü noktaları anomali olarak değerlendirilir.

DBSCAN yönteminin uygulamak için `make_blobs` fonksiyonuyla rastgele veri elde edelim. `make_blobs` fonksiyonu `sklearn.datasets` modülü içerisindeindir. Bu fonksiyon kümeleme işlemleri için istenilen kume sayısına uygun istenilen miktarda rastgele nokta üretmektedir. Biz burada anomali tespiti için kume sayısını 1 alacağız. Örneğin:

```
from sklearn.datasets import make_blobs  
  
dataset_x, dataset_y = make_blobs(100, centers=1, cluster_std=1)  
  
import matplotlib.pyplot as plt  
  
plt.title('Rando Points for DBSCAN Anomaly Detection')  
plt.scatter(dataset_x[:, 0], dataset_x[:, 1], color='blue')
```



Şimdi DBSCAN algoritmasını çalıştırarak gürültü noktalarını belirleyelim:

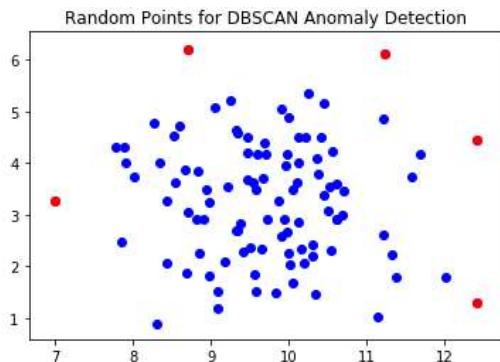
```
from sklearn.cluster import DBSCAN  
  
dbs = DBSCAN(eps=1)
```

```

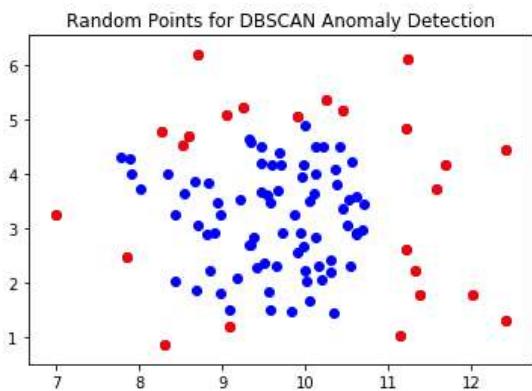
dbs.fit(dataset_x)
plt.pause(1)

plt.title('Random Points for DBSCAN Anomaly Detection')
plt.scatter(dataset_x[:, 0], dataset_x[:, 1], color='blue')
plt.scatter(dataset_x[dbs.labels_ == -1, 0], dataset_x[dbs.labels_ == -1, 1], color='red')

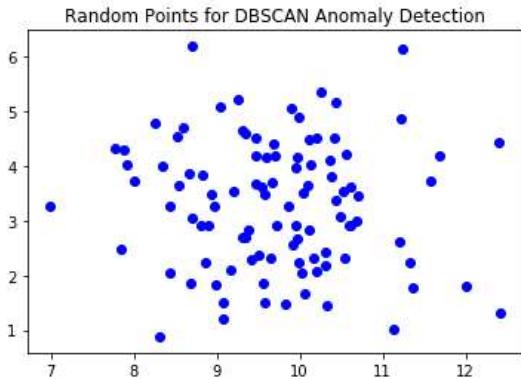
```



Tabii DBSCAN yöntemini uygularken yoğun belirten epsilon değerinin iyi bir biçimde belirlenmesi gereklidir. Anımsanacağı gibi epsilon değeri küçüldükçe gürültü noktalarının sayıları artmaktadır, epsilon değeri yükseldikçe gürültü noktalarının sayıları azalmaktadır. Yukarıdaki örnekte epsilon 1 alınmıştır. Epsilon değerini 0.5'e küçültüp örneği bir daha çalışıralım:



Şimdi de epsilon'u 1.5 alarak kodumuzu çalışıralım:



Göründüğü gibi epsilon parametresinin uygulamacı tarafından ayarlanması gerekmektedir. Epsilon değeri küçültüldükçe şüpheli durumların sayısı artacaktır. Bu da şüpheli durumların değerlendirilmesi için daha fazla kaynağın harcanmasına yol açacaktır.

Şimdi de DBSCAN yerine K-Means algoritmasıyla anomalileri belirlemeye çalışalım. Anımsanacağı gibi K-Means yönetminde algoritmanın noktaları ayıracığı küme sayısını başlangıçta biz veriyorduk. Algoritma da tüm noktaları bir

kümeye dahil ediyordu. (Yani noktalar ağırlık merkezlerine çok uzak olsalar bile K-Means algoritması yine de bu noktaları bir kümeye dahil etmektedir.) K-Means algoritmasının 1 küme için çalıştırılması anlamsız gelebilir. Gerçekten de önceki konularda bizim yazdığımız K-Means kodu tek kümeye için tek bir iterasyonla işlemini sonlandırmaktadır. Ancak K-Means algoritmasının ince versiyonları vardır. Örneğin scikit-learn kütüphanesinin kullandığı K-Means algoritmasında küme sayısı 1 olsa bile algoritma o tek kümeye rastgele aldığı ağırlık merkezini iyileştirmektedir. İşte K-Means yoluyla anomali tespitinde algoritma 1 kümeye için çalıştırılır. Algoritma tüm noktaları o tek kümeye dahil etmeye birlikte o kümeyi ağırlık merkezini iyi bir noktaya çeker. Böylece uygulamacı tüm noktaların bu ağırlık merkezine uzaklıklarını hesaplayarak en uzak n tane noktayı anomali olarak belirleyebilir.

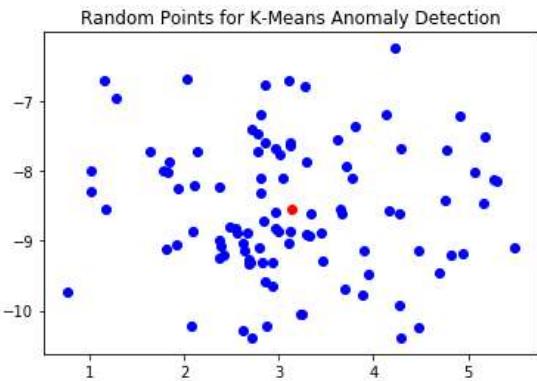
Örneğin make_blobs fonksiyonuyla bir kümeye için ağırlık merkezini bularak noktaların grafiğini çizdirelim:

```
from sklearn.datasets import make_blobs
dataset_x, dataset_y = make_blobs(100, centers=1, cluster_std=1)
import matplotlib.pyplot as plt
plt.title('Rando Points for DBSCAN Anomaly Detection')
plt.scatter(dataset_x[:, 0], dataset_x[:, 1], color='blue')

from sklearn.cluster import KMeans
km = KMeans(n_clusters=1)
km.fit(dataset_x)

plt.pause(1)

plt.title('Random Points for k-MEANS Anomaly Detection')
plt.scatter(dataset_x[:, 0], dataset_x[:, 1], color='blue')
plt.scatter(km.cluster_centers_[0][0], km.cluster_centers_[0][1], color='red')
```



Grafkte noktalar ve ağırlık merkezi gösterilmiştir. Uygulamacının tüm noktaların ağırlık merkezine olan uzaklıklarını hesaplayıp en uzak n tane noktayı anomali olarak tespit etmesi gereklidir. Aslında KMeans sınıfının transform metodunu zaten verdiğimiz noktaların ağırlık merkezlerine uzaklığını bize vermektedir. Yani biz noktaların ağırlık merkezlerine uzaklıkları için doğrudan bu metottan faydalananabiliriz:

```
import numpy as np
distances = km.transform(dataset_x).ravel()
sorted_distances = np.argsort(distances)
outliers = sorted_distances[-5:]
```

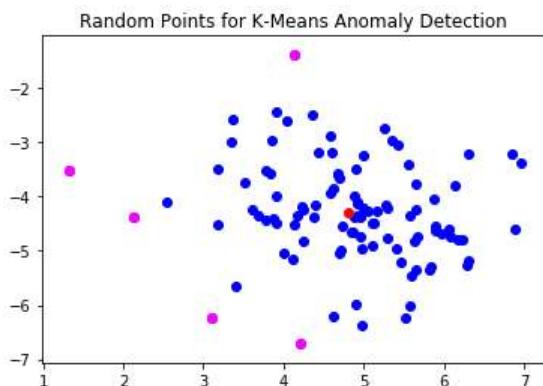
ndarray sınıfının ravel metodunu çok boyutlu vektörü tek boyut haline getirmektedir. np.argsort fonksiyonu ise asıl vektörü (yani örneğimizdeki distances vektörünü) değil o vektörün indislerini sıraya dizmektedir. Böylece sorted_distances vektöründeki son n tane eleman aslında ağırlık merkezlerine en uzak n noktanın indisleridir. Şimdi biz bu noktaları da grafikte belirtelim:

```

plt.pause(1)

plt.title('Random Points for K-Means Anomaly Detection')
plt.scatter(dataset_x[:, 0], dataset_x[:, 1], color='blue')
plt.scatter(km.cluster_centers_[0][0], km.cluster_centers_[0][1], color='red')
plt.scatter(dataset_x[outliers, 0], dataset_x[outliers, 1], color='magenta')

```



Şimdi de özellik indirgemesi ve yükseltilmesi yöntemiyle anomali tespiti üzerinde duralım. Bu yöntemde n tane sütundan oluşan veri tabloları üzerinde özellik indirgemesi yapılarak bu n tane sütun k tane sütuna indirgenir. Sonra bu k tane sütun ters dönüşümle yeniden n tane sütuna yükseltilir. Daha sonra orijinal veri tablosuyla indirgenmiş yükseltilmiş veri tablosu arasında karşılaştırma yapılır. Bu yöntemde önce indirgeme sonra yükselme yapıldığında anomali içeren satırlarda daha ciddi bir farklılaşma oluşmaktadır. Yöntemde indirgeme için PCA ya da diğer bazı indirgeme yöntemleri de kullanılabilirlerdir.

Pekiyi indirgenip yükseltilmiş verilerle orijinal verilerin arasındaki farka nasıl bakılmaktadır? Yaygın kullanılan bir yöntem iki vektörel nokta arasındaki farkların karelerinin toplamlarını [0, 1] arasında normalize ederek karşılaştırmaktadır. Örneğin aşağıdaki gibi bir uzaklık fonksiyonu yazılabilir:

```

def anomaly_scores(original_data, manipulated_data):
    loss = np.sum((original_data - manipulated_data) ** 2, axis=1)
    loss = (loss - np.min(loss)) / (np.max(loss) - np.min(loss))
    return loss

```

Fonksiyonun sonunda değerleri 0 ile 1 arasına getirdik. Bu işlem yapılmasa da olurdu. Buradan amaç oransal bir değerin elde edilmesidir. Bu da seçim işleminde faydalı olabilmektedir.

Şimdi özellik indirgemesi ve yükseltilmesi yoluyla anomali tespit edilmesi yöntemini kredi kart işlemlerine ilişkin bir veri tablosu üzerinde uygulayalım. Veri tablosunu "Kaggle" sitesinden indirerek kullanacağız. Siz de bu tabloyu CSV dosyası olarak <https://www.kaggle.com/mlg-ulb/creditcardfraud> adresinden indirebilirsiniz. (Kursumuzda bu dosya "creditcard.csv" ismiyle indirilmiştir.) "creditcard.csv" dosyası 30 sütundan oluşmaktadır. Dosyanın başında bir başlık kısmı vardır. Eğer dosya doğrudan numpy.loadtxt fonksiyonuyla okunacaksa bu başlık kısmı geçilmelidir. Dosyanın 30'uncu sütunu iki tür mak işerisinde "0" ya da "1" olan yazılarından oluşmaktadır. Bu yazılar ilgili işlemin geçerli mi (0 değeri) yoksa geçersiz mi (1 değeri) olduğunu belirtmektedir. Dolayısıyla bu sütun diğer sütunlardan ayırtılmalıdır. Toplam 284807 tane işlem arasında sahte olanların sayısı 492 tanedir. "creditcard.csv" dosyasının yüklenip kullanıma hazır hale getirilmesi şöyle yapılabilir:

```

import numpy as np

dataset_x = np.loadtxt('creditcard.csv', dtype=np.float32, delimiter=',', skiprows=1,
usecols=range(30))
dataset_y = np.loadtxt('creditcard.csv', dtype='str', delimiter=',', skiprows=1, usecols=[30])

from sklearn.preprocessing import StandardScaler

ss = StandardScaler()
dataset_x = ss.fit_transform(dataset_x)

```

Buradan dataset_x ve dataset_y isminde iki ndarray elde etmiş olduk. Şimdi dataset_x verilerini PCA ile indirgeyip sonra yükseltelim:

```
from sklearn.decomposition import PCA
pca = PCA(n_components=19)
reduced_dataset_x = pca.fit_transform(dataset_x)
inversed_dataset_x = pca.inverse_transform(reduced_dataset_x)
```

Biz şimdi orijinal veri tablosunu (dataset_x) önce 19 sütuna indirgeyip sonra yine 30 sütuna yükselttik. Bu işlemden inversed_dataset_x tablosunu elde ettik. Artık orijinal dataset_x tablosu ile inversed_dataset_x tablosunu yukarıda yazmış olduğumuz anomaly_scores fonksiyonuna sokarak anomali değerlerini elde edebiliriz:

```
scores = anomaly_scores(dataset_x, inversed_dataset_x)
```

Şimdi bu skorlar arasında en yüksek n tanesini (örneğin 1000 tanesini) alalım:

```
sorted_scores = np.argsort(scores)
anomaly_results = dataset_x[sorted_scores[-1000:]]

anomaly = np.sum(dataset_y[sorted_scores[-fraud_count:]])
print(anomaly)
```

Burada görüldüğü gibi önce indirgeme sonra yükseltme yapılip orijinal verilerle satır temelinde uzaklık hesaplanmıştır. Elde edilen scores vektörü tek boyutludur ve yalnızca satırların (yani kredi kartı işlemlerinin) anomali değerini barındırır. Biz en yüksek değere sahip olan işlemlerin anomali içeriği varsayımda bulunmuştur. Bu durumda buradaki scores vektörünü sıraya dizersek en kötü n tane değeri kolayca elde edebiliriz. Ancak bu vektörün sıraya dizilmesi orijinal verilerdeki satır ilişkisini bozacaktır. Bu nedenle örneğimizde bu vektör sort etmek yerine bu vektörün indislerini sort ettik. Sonuçta n_components = 19 değeri için en kötü 1000 anomali değeri arasında 314 tane anomalili işlem, en kötü 100 arasında ise 77 tane anomalili işlem yakalanmıştır. Ancak programın her çalıştırılmasında farklı değerler bulunabilmektedir. 1000 tane değer arasında 314 tane şüpheli işlemin yakalanması aslında oldukça başarılıdır. Çünkü zaten veri kümesindeki 284807 işlemin yalnızca 492 tanesi sahtecilik içermektedir.

Şimdi de bu kredi kart işlemleri K-Means yöntemiyle ele alalım:

```
import numpy as np

dataset_x = np.loadtxt('creditcard.csv', dtype=np.float32, delimiter=',', skiprows=1,
usecols=range(30))
dataset_y = np.loadtxt('creditcard.csv', dtype='str', delimiter=',', skiprows=1, usecols=[30])
dataset_y = np.array([int(s[1]) for s in dataset_y], dtype=np.float32)

from sklearn.preprocessing import StandardScaler

ss = StandardScaler()
dataset_x = ss.fit_transform(dataset_x)

from sklearn.cluster import KMeans

km = KMeans(n_clusters=1)
km.fit(dataset_x)

import numpy as np

distances = km.transform(dataset_x).ravel()
sorted_distances = np.argsort(distances)

outliers = sorted_distances[-1000:]
```

```
outliers_count = np.sum(dataset_y[outliers])
print(outliers_count)
```

K-Means yöntemini kullanarak yaptığımız anomalî tespitinde en kötü skora sahip 1000 işlem içerisinde bu yöntem 170 tane şüpheli işlemi tespit etmiştir.

İSTATİKSEL YÖNETMLERLE GERÇEKLEŞTİRİLEN REGRESYON ANALİZLERİ

Daha önce de belirttiğimiz gibi regresyon girdi ile çıktı arasında ilişki kurma sürecidir. Bağımsız değişken birden fazlaysa buna çoklu regresyon (multiple regression), bağımlı değişken birden fazlaysa buna da "çok değişkenli (multivariate) regresyon" denilmektedir. Örneğin x_1, x_2, x_3, x_4, x_5 değerlerinden hareketle bir y değerini bulmak isteyelim. Bu çoklu regresyona bir örnektir. Çünkü burada bulunmak istenen fonksiyon $y = f(x_1, x_2, x_3, x_4, x_5)$ gibi bir fonksiyondur. Ancak x_1, x_2, x_3, x_4, c_5 değerlerinden hareketle biz y_1 ve y_2 değerlerini bulmak istersek bu çok değişkenli regresyona örnektir. Çünkü burada biz $(y_1, y_2) = f(x_1, x_2, x_3, x_4, x_5)$ gibi bir fonksiyonu bulmaya çalışırız. Çok değişkenli regresyon ayrı ayrı yapılan çoklu regresyonla aynı şey değildir. Örneğin biz x_1, x_2, x_3, x_4, x_5 değerlerinden hareketle y_1 ve y_2 değerlerini bulmaya çalışalım. İlk bakışta bu y_1 ve y_2 değerlerinin iki ayrı çoklu regresyon ile mümkün gibi görülmektedir. Ancak ayrı ayrı yapılan çoklu regresyonla tek hamlede yapılan çok değişkenli regresyon aynı sonuçları vermez. Çünkü çok değişkenli regresyonda bağımlı değişkenler de birbirlerini etkileyebilmektedir. Bu durumda bağımlı değişkenler arasındaki ilişkiyi görmezden gelerek ayrı ayrı çoklu regresyon uygulamak farklı sonuçların elde edilmesine yol açmaktadır.

Regresyonlar aynı zamanda bulunacak fonksiyonun biçimine göre de "doğrusal", "polinomsal", "üstel" gibi sınıflara ayrılmaktadır. En yaygın kullanılan regresyon "çoklu doğrusal regresyondur (multiple linear regression)".

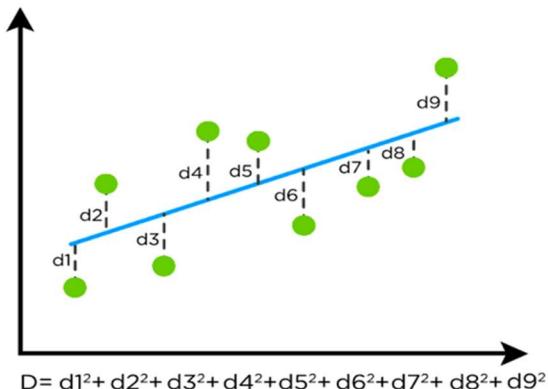
Regresyon ile korelasyon arasında önemli bir ilişki vardır. Aralarında korelasyonun sağlam olmadığı değişkenlerde regresyon analizi yapmak (özellikle doğrusal regresyon) genellikle uygun olmaz. Bu tür durumlarda regresyon sonucuna dayanılarak yapılan kestirimler başarısız olmaktadır. Bu nedenle nümerik regresyonun uygunluğunu anlamak için önce değerlerin korelasyon katsayısına bakmak gereklidir.

Daha önce biz regresyon problemlerini yapay sinir ağlarıyla çözmüştük. Korelasyonu zayıf olan olgular için yapay sinir ağları oldukça iyi bir yöntem haline gelmektedir. Maalesef karşılaşılan gerçek problemlerin büyük kısmında nümerik regresyon uygulanamayacak derecede düşük korelasyonlar bulunmaktadır. Bu durumda aralarında düşük korelasyon ilişkisi bulunan değişkenler için kestirim yöntemi olarak yapay sinir ağları tercih edilmelidir. Yani yapay sinir ağlarıyla regresyon uygulamalarında değişkenler arasında anlamlı bir korelasyonun bulunuyor olması gerekmektedir.

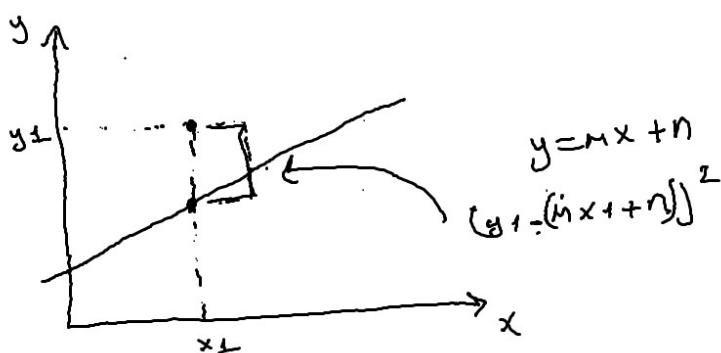
Doğrusal Regresyon

Yukarıda da belirtildiği gibi doğrusal regresyon birtakım noktaları temsil edebilecek bir doğru denklemin elde edilmesi sürecidir. Örneğin tek bağımsız değişkenli doğrusal regresyonda elde edilmek istenen doğru $y = a_0 + a_1x$ biçimindedir. Burada a_1 aynı zamanda doğrunun eğimidir., a_0 da doğrunun y ekseni kestiği noktasıdır. (Bu doğru denklemi genellikle $y = mx + n$ biçiminde de ifade edilmektedir.) Çoklu doğrusal regresyon ile tekli doğrusal regresyon arasında aslında hesaplamalar bakımından pek bir farklılık yoktur. Tekli doğrusal regresyon iki boyutlu düzlemdede bir doğru denkleminin elde edilmesi süreci çoklu doğrusal regresyon n boyutlu uzayda bir doğru denkleminin elde edilmesi sürecidir. Çok boyutlu uzaydaki bu genel doğru denklemi de $y = a_0 + a_1x_1 + a_2x_2 + a_3x_3 + \dots + a_nx_n$ biçimindedir. Bu durumda n bağımsız değişken için regresyon analizi sonucunda bizim $n + 1$ 'tanın değer elde etmemiz gerekmektedir. Mademki tek bağımsız değişkenli doğrusal regresyon hesabı ile çoklu doğrusal regresyon hesabı arasında bir farklılık yoktur, o halde biz doğrusal regresyonun genel ifadesini elde etmek için tek değişkenli modelden faydalananabiliriz.

Tek bağımsız değişkenli doğrusal regresyondaki katsayı değerlerinin elde edilmesi için kullanılan formülüne "en küçük kareler (least squares)" formülü denilmektedir. $y = mx + n$ doğrusundaki m ve n değerlerini veren en küçük karaler formülü aslında söz konusu noktaları en iyi biçimde ortalayan doğrunun m ve n değerlerini bulmaktadır. Burada noktaları en iyi biçimde ortalayan doğru demekle noktaların doğruya uzunlukları toplamını en küçükleşen doğru kastedilmektedir. Bir noktanın koordinatları (x_1, y_1) ise bu noktadan doğruya uzaklık $y_1 - mx_1 + n$ 'dır. $((x_1, y_1)$ noktasının doğru üzerindeki izdüşümündeki x değerinin de x_1 olduğuna dikkat ediniz. Yani ilgili noktadan doğruya dikme indirilmemektedir.)



Burada en küçüklemeye çalışılan şey noktaların doğruya uzaklıklarının toplamıdır. Ancak bu uzaklıklar negatif olabileceğinden dolayı negatiflikten onları kurtarmak için kare alma işlemi uygulanır. Belli bir noktanın buradaki doğruya uzunluğu basit biçimde şöyle hesaplanabilmektedir:



Amaç tüm noktaların doğruya toplam uzaklıklarının en küçükleneceği olduğuna göre X ve Y noktaları belirten vektörler olmak üzere en küçüklenecek ifade şöyleden oluşturulabilir:

$$\sum (Y - (mX + n))^2$$

Bizim buradaki asıl amacımız m ve n değerlerini bulmaktır. Bu işlem matematiksel olarak m'ye ve n'ye göre birinci kismi türevlerinin sıfırlanması ile gerçekleştirilir. O halde biz de yukarıdaki ifadenin m'ye ve n'ye göre kismi türevlerini alıp sıfırlayalım. Buradan da m'yi ve n'yi çekelim:

$$\frac{\partial f}{\partial m} = -2 \sum (Y - n - mX)X = 0$$

$$\frac{\partial f}{\partial n} = -2 \sum (Y - n - mX) = 0$$

$$m = \frac{\sum (X - \bar{X})(\bar{Y} - \bar{X})}{\sum (X - \bar{X})^2}$$

$$n = \frac{\sum X - m \sum X}{\sum (X - \bar{X})}$$

Aynı eşitlikler şöyle de ifade edilebilirdi:

$$m = \frac{N \sum(xy) - \sum x \sum y}{N \sum(x^2) - (\sum x)^2}$$

$$b = \frac{\sum y - m \sum x}{N}$$

Tabii aslında bu tek bağımsız değişken için elde edilen formül çok bağımsız değişken için de benzer biçimde uygulanmaktadır. Örneğin yukarıda bizim çıkarttığımız formülde x aslında tek bir değeri değil bir vektörü temsil ediyor olabilir. Bu durumda x ortalama (x üstü çizgi) da aslında her değişkene ilişkin sütunun kendi ortalamasıdır. O halde çok değişken için aslında m değeri de tek bir değer değil bir vektör olarak elde edilecektir. Bu vektördeki değerler $a_0 + a_1x_1 + a_2x_2 + a_3x_3 + \dots + a_nx_n$ doğru denklemindeki $a_1, a_2, a_3, \dots, a_n$ değerleri olacaktır. Başka bir deyişle Çoklu regresyon durumunda yukarıda çıkarttığımız formüldeki m eşitliğinde bulunan x ve x ortalama (x üzeri çizgi) aslında birer vektördür.

Şimdi doğrusal regresyon için m ve n değerlerini bulan bir fonksiyon yazalım:

```
import numpy as np

def linear_regression(x, y):
    a = np.sum((x - np.mean(x)) * (y - np.mean(y)))
    b = np.sum((x - np.mean(x)) ** 2)
    m = a / b
    n = (np.sum(y) - m * np.sum(x)) / len(x)
    return m, n
```

Şimdi de "test.csv" içersindeki noktalar doğrusal regresyon grafiğini çizelim. "test.csv" dosyasının içeriği şöyledir:

```
2,4
3,5
5,7
7,10
7,8
8,12
9.5,10.5
9,15
10,17
13,18
```

Program şöyle oluşturulabilir:

```

import numpy as np

def linear_regression(x, y):
    m = np.sum((x - np.mean(x)) * (y - np.mean(y))) / np.sum((x - np.mean(x)) ** 2)
    n = np.sum((np.sum(y) - m * np.sum(x))) / len(x)

    return m, n

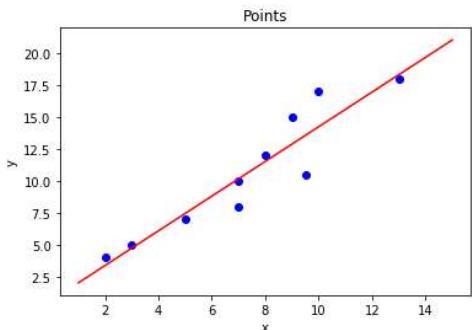
dataset = np.loadtxt('test.csv', dtype=np.float32, delimiter=',')
m, n = linear_regression(dataset[:, 0], dataset[:, 1])

x = np.linspace(1, 15, 100)
y = m * x + n

import matplotlib.pyplot as plt

plt.title('Points')
plt.xlabel('x')
plt.ylabel('y')
plt.scatter(dataset[:, 0], dataset[:, 1], color='blue')
plt.plot(x, y, color='red')

```



Şimdi de interpolasyonla kestirimde bulmaya çalışalım. Örneğin $x = 11$ için y 'nin değeri nedir?

```

xval = 11
yval = m * xval + n
print('{} --> {}'.format(xval, yval))

11 --> 15.611846947669983

```

Doğrusal regresyon scikit-learn'de `linear_model` isimli modülde `LinearRegression` isimli sınıfı gerçekleştirilmiştir. Bu sınıf hem tekli, hem çoklu hem de çok değişkenli regresyon için kullanılabilir. Sınıfın `__init__` metodu şöyledir:

```

class sklearn.linear_model.LinearRegression(fit_intercept=True, normalize=False, copy_X=True, n_jobs=None)

```

`LinearRegression` nesnesi yaratıldıkten sonra sınıfın `fit` metodu ile regresyon uygulanır. Sınıfın `predict` metodu ise değerleri doğru denkleminde yerine koymak bize sonucu vermektedir. Doğru denklemindeki katsayıları sınıfın `coef_` isimli özniteligiden, y ekseni kesim noktasını da `intercept_` isimli özniteligiden elde edebiliriz. Anımsanacağı gibi çoklu regresyonlarda doğru katsayıları bir tane değil n tanedir. Örneğin tekli regresyon için doğru denklemi şöyledir:

$$y = a_0 + a_1 x$$

Burada a_1 doğrunun eğimi (yani m), a_0 ise ekseni kesim noktasıdır. Dolayısıyla biz tekli regresyonda sınıfın `coef_` özniteligiden bir tane katsayı elde ederiz. Bu da a_1 katsayıdır. Sınıfın `coef_` özniteligi ise bize a_0 değerini vermektedir. Çoklu regresyonda ise doğru denklemi şu biçimde olacaktır:

$$y = a_0 + a_1 x_1 + a_2 x_2 + a_3 x_3 + \dots + a_n x_n$$

Burada ise sınıfın `coef_` özniteliği bize n elemanlı bir katsayı vektörü verecektir. Bu n elemanlı katsayı vektörü sırasıyla $a_1, a_2, a_3, \dots, a_n$ değerlerine karşılık gelmektedir. Sınıfın `intercept_` özniteliği ise yine bize 1 tane değer verecektir. Bu da genel gösterimdeki a_0 değeridir.

Şimdi yukarıda manuel olarak yaptığımız doğrusal regresyon örneğini `LinearRegression` sınıfı ile yapalım:

Örneğin:

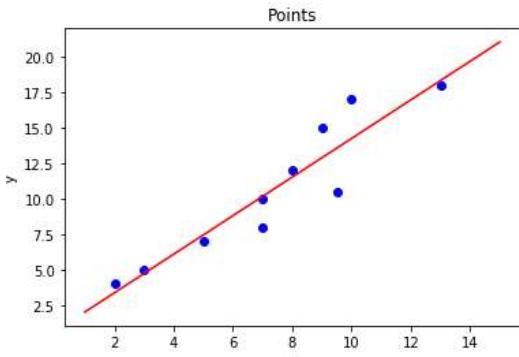
```
import numpy as np
dataset = np.loadtxt('test.csv', dtype=np.float32, delimiter=',')
from sklearn.linear_model import LinearRegression

lr = LinearRegression()
lr.fit(dataset[:, 0].reshape(-1, 1), dataset[:, 1])

x = np.linspace(1, 15, 100)
y = lr.coef_[0] * x + lr.intercept_

import matplotlib.pyplot as plt

plt.title('Points')
plt.xlabel('x')
plt.ylabel('y')
plt.scatter(dataset[:, 0], dataset[:, 1], color='blue')
plt.plot(x, y, color='red')
```



Burada `fit` metodunun iki boyutlu bir matris istediği dikkat ediniz. Çünkü `LinearRegression` sınıfı çoklu regresyonlarda da kullanılabilmektedir. Yukarıdaki örnekte `reshape` fonksiyonu bu amaçla kullanılmıştır. Yukarıdaki örneğimizde `x` değerleri `reshape` yapıldıktan sonra aşağıdaki duruma getirilmiştir:

```
array([[ 2. ],
       [ 3. ],
       [ 5. ],
       [ 7. ],
       [ 7. ],
       [ 8. ],
       [ 9.5],
       [ 9. ],
       [10. ],
       [13. ]], dtype=float32)
```

Doğrusal regresyon işleminde yapılan regresyonun noktaları ne kadar iyi temsil edebildiğine yönelik R^2 denilen bir ölçüt kullanılmaktadır. R^2 değeri ne kadar yüksekse noktaların doğruya uzunlukları o kadar kısalıdır. Yani regresyon doğrusu noktalara o kadar yakındır. R^2 değeri düştükçe regresyon doğrusunun noktaları temsil etme yeteneği azalmaktadır. Yani yüksek bir R^2 değeri söz konusu olduğunda bizim yapacağımız kestirim de o kadar iyi olacaktır. R^2 değeri noktaların doğruya olan uzaklıklarının karelerinin toplamının, noktaların kendi ortalamalarına uzaklıklarının karelerinin toplamına oranıdır. Biz burada R^2 değerinin nasıl hesaplanacağı üzerinde durmayacağız. `LinearRegression`

sınıfinın score isimli metodu bize R^2 değerini vermektedir. Örneğin yukarıdaki "test.csv" dosyasındaki noktalar için R^2 değeri şöyle elde edilebilir:

```
rsquare = lr.score(dataset[:, 0].reshape(-1, 1), dataset[:, 1])
print(rsquare)
```

```
0.871811303294605
```

Şimdi de LinearRegression sınıfını kullanarak çoklu regresyon örneği verelim. Biz daha önce Boston'daki ev fiyatlarını tahmin eden örneği yapay sinir ağlarıyla gerçekleştirmiştik. Şimdi Boston örneğini nümerik olarak çoklu regresyon ile gerçekleştirelim. Boston verileri keras kütüphanesinin yanı sıra scikit-learn içerisinde de datasets modülünde bulunmaktadır. Ancak bu Boston örneğindeki sütunların hepsini değil yalnızca 5'inci ve 12'inci sütunları biz regresyonda kullanacağız. Çünkü tüm sütunlar işleme sokulduğunda düşük korelasyon oluşturmaktadır. (Animşanacağı gibi bu tür düşük korelasyonlu durumlarda nümerik regresyon değil yapay sinir ağları ile regresyon çok daha uygun olmaktadır.)

```
from sklearn.datasets import load_boston

boston_dataset = load_boston()

dataset_x = boston_dataset.data[:, (5, 12)]
dataset_y = boston_dataset.target

from sklearn.linear_model import LinearRegression

lr = LinearRegression()
lr.fit(dataset_x, dataset_y)

print('Coefficients: {}'.format(lr.coef_))
print('Intercept: {}'.format(lr.intercept_))
```

Katsayılar ve eksen kesim noktası şöyle bulunmuştur:

```
Coefficients: [ 5.09478798 -0.64235833]
Intercept: -1.358272811874489
```

Şimdi x değerleri için bir y değerini bulalım. Bunu iki biçimde yapabiliriz. Birincisi doğru denkleminde yerine koyarak. İkincisi doğrudan predict metodunu kullanarak:

```
import numpy as np

xvals = np.array([10, 12], dtype=np.float32)
yval = np.dot(lr.coef_, xvals) + lr.intercept_
print('{} ---> {}'.format(xvals, yval))
```

Elde edilen değerler şöyledir:

```
[10. 12.] ---> 41.88130702056141
```

Şimdi de aynı işlemi predict metoduyla yapalım:

```
yval = lr.predict(xvals.reshape(1, -1))
print('{} ---> {}'.format(xvals, yval))
```

Elde edilen değerler şöyledir:

```
[10. 12.] ---> [41.88130702]
```

Biz bu örnekte iki değişkenli çoklu regresyon uyguladık. Regresyon sonucunda bulduğumuz doğru denkleminin genel biçimini şöyledir:

$$y = a_0 + a_1x_1 + a_2x_2$$

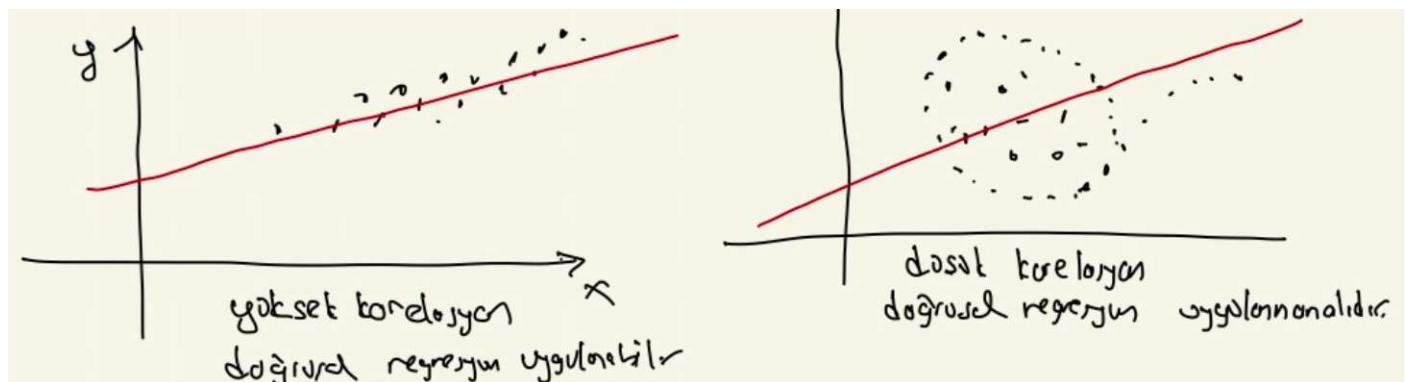
Burada a_0 değeri sınıfın intercept_ozniteliğinden elde edilmiştir. a_1 ve a_2 değerleri ise sınıfın coef_ozniteliğinden elde edilmiştir. Yani bizim bulduğumuz doğrusal denklemi şöyledir:

$$y = -1.358272811874489 + 5.09478798 \times x_1 - 0.64235833 \times x_2$$

Çok değişkenli doğrusal regresyonlarda da yine aynı LinearRegression sınıfı kullanılmaktadır. Burada fit işlemi uygulanırken y verilerinin bir matris biçiminde olması gereklidir.

İstatistiksel Doğrusal Regresyon Yönetimiyle Yapay Sinir Ağlarıyla Regresyon Yönteminin Karşılaştırılması

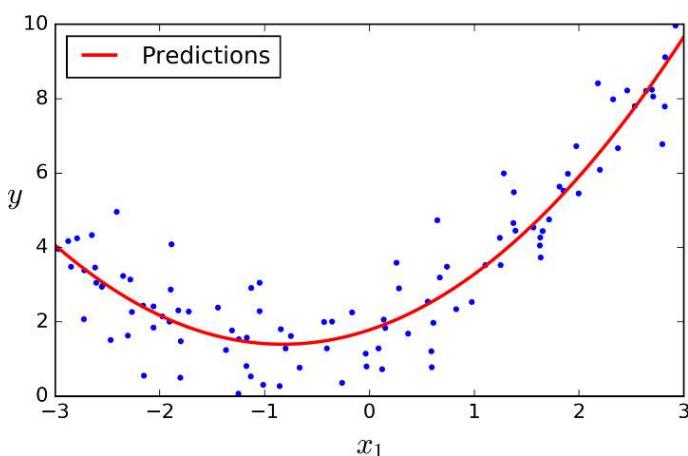
İstatistiksel doğrusal regresyon yukarıda da belirtildiği gibi korelasyonun yüksek olduğu verilere uygulanabilmektedir. x verileriyle y arasındaki korelasyon düşükse bu verileri bir doğru ile temsil etmek iyi bir fikir değildir. Örneğin:



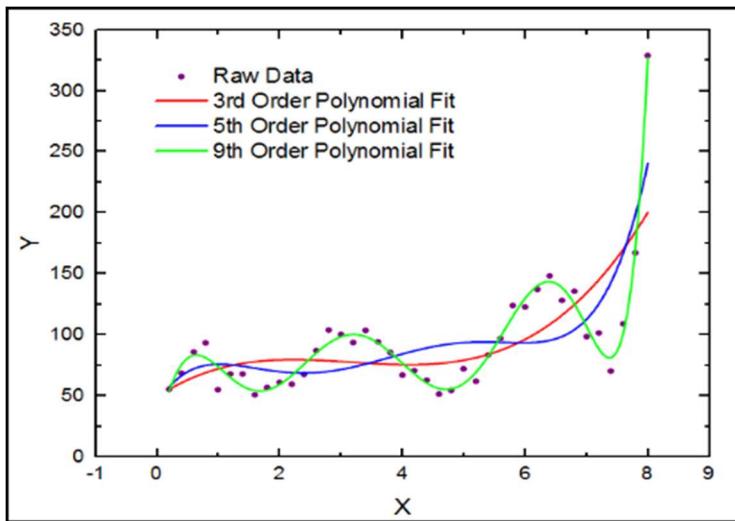
Yapay sinir ağları için böyle bir korelasyon koşulu yoktur. Yapay sinir ağları korelasyonu düşük verilerde de uygun kalıpları bulup bir tahminde bulunmamıza olanak sağlamaktadır. Öte yandan yapay sinir ağlarının eğitilmesi için nispeten yüksek sayıda veriye gereksinim vardır. Eğer eğitimde kullanılacak veri sayısı düşükse yapay sinir ağları çok zayıf bir performans göstermektedir. Halbuki doğrusal regresyon az sayıda verinin korele bir biçimde bulunduğu durumlarda çok daha pratik ve uygundur.

Polinomsal Regresyon

Polinomsal regresyon noktaları temsil edebilecek bir polinomsal eğrinin elde edilmesi sürecidir. Doğrusal olmayan bir regresyon çeşididir. Bazı durumlarda tercih edilebilmektedir. Örneğin:



Burada doğrusal bir regresyonun uygulanmasının uygun olmadığı grafikten görülmektedir. Pekiyi uygulanacak polinom kaçinci dereceden olmalıdır? Örneğin:



Polinomsal regresyon scikit-learn kütüphanesinde preprocessing modülündeki `PolynomialFeatures` sınıfının yardımıyla yapılmaktadır. Sınıfın `__init__` metodunun parametrik yapısı şöyledir:

```
sklearn.preprocessing.PolynomialFeatures(degree=2, interaction_only=False, include_bias=True, order='C')
```

Fonksiyonun `degree` parametresi kaçinci derece bir polinomla ilgilenildiğini belirtir. Nesne yaratıldıktan sonra önce fit sonra da transform işlemlerini yapmak gereklidir. İki işlem bir arada `fit_transform` fonksiyonuyla da yapılabilmektedir. `transform` ya da `fit_transform` bize dönüştürülmüş bir matris verir. Burada biz öncelikle bize verilen bu dönüştürülmüş matrisin anlamı üzerinde duracağız. Örneğin aşağıdaki gibi tek bağımsız değişken ve tek bağımlı değişkenden oluşan bir verimiz olsun:

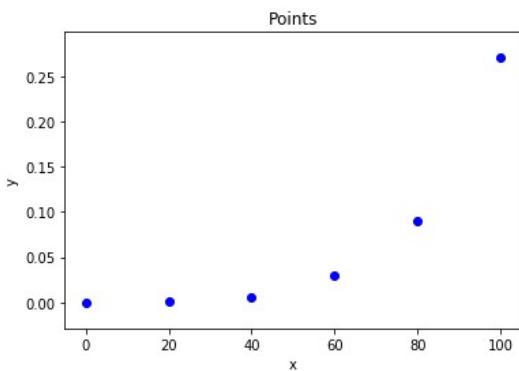
`test.csv`

```
0,0.0002
20,0.0012
40,0.0060
60,0.0300
80,0.0900
100,0.27
```

Bu noktaların serpilme grafiğini çizelim:

```
import numpy as np

dataset = np.loadtxt('test.csv', dtype=np.float32, delimiter=',')
import matplotlib.pyplot as plt
plt.title('Points')
plt.xlabel('x')
plt.ylabel('y')
plt.scatter(dataset[:, 0], dataset[:, 1], color='blue')
```



Grafikten de görüldüğü gibi x ve y arasında ikinci derece polinomsal bir ilişkinin varlığı anlaşılmaktadır. O halde biz tahminleme yapmak için ikinci derece polinomsal bir regresyon analizi uygulayabiliriz. Pekiyi biz noktalara doğrusal regresyon uygulamak istesek elde edeceğimiz doğru ne kadar uygun olabilir? Deneyelim:

```
import numpy as np

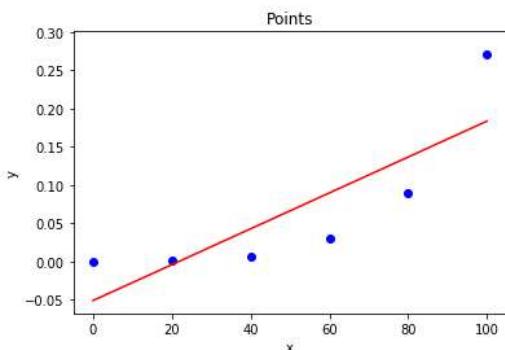
dataset = np.loadtxt('test.csv', dtype=np.float32, delimiter=',')
from sklearn.linear_model import LinearRegression

lr = LinearRegression()
lr.fit(dataset[:, 0].reshape(-1, 1), dataset[:, 1])

x = np.linspace(0, 100, 100)
y = lr.coef_[0] * x + lr.intercept_

import matplotlib.pyplot as plt
plt.title('Points')
plt.xlabel('x')
plt.ylabel('y')
plt.scatter(dataset[:, 0], dataset[:, 1], color='blue')
plt.plot(x, y, color='red')
```

Elde edilen grafik şöyledir:



Burada elde edilen doğrunun noktaları iyi temsil etmediği gözle görülmektedir. Şimdi de R^2 değerine bakalım:

```
rsquared = lr.score(dataset[:, 0].reshape(-1, 1), dataset[:, 1])
print(rsquared)

0.6903499390522619
```

R^2 değerinin düşük olduğunu görüyorsunuz. Yukarıda da belirtildiği gibi bu noktalar için bir doğru geçirmeye çalışmak yerine ikinci dereceden bir polinom eğrisi geçirmeye çalışmak daha uygun olacaktır.

Scikit-learn kütüphanesinde polinomsal regresyon iki aşamada uygulanmaktadır. Birinci aşamada `sklearn.preprocessing` modülündeki `PolynomialFeatures` sınıfı kullanılarak noktalar transpoze edilir. İkinci aşamada

transpoze edilmiş noktalar yoluyla doğrusal regresyon uygulanır. İzleyen paragraflarda da açıklandığı gibi polinomsal regresyon aslında çoklu doğrusal regresyonla aynı anlamdadır. Şimdi biz PolynomialFeatures sınıfı ile noktalarımızı transpoze edelim:

```
import numpy as np

dataset = np.loadtxt('test.csv', dtype=np.float32, delimiter=',')
dataset_x = dataset[:, 0].reshape(-1, 1)
dataset_y = dataset[:, 1]

from sklearn.preprocessing import PolynomialFeatures

pf = PolynomialFeatures(degree=2)
transformed_dataset_x = pf.fit_transform(dataset_x)
print(transformed_dataset_x)
```

Burada fit_transform işleminden elde edilen matrisin şecline bakınız:

```
[[1.0e+00 0.0e+00 0.0e+00]
 [1.0e+00 2.0e+01 4.0e+02]
 [1.0e+00 4.0e+01 1.6e+03]
 [1.0e+00 6.0e+01 3.6e+03]
 [1.0e+00 8.0e+01 6.4e+03]
 [1.0e+00 1.0e+02 1.0e+04]]
```

Buradaki matrisin 6x3'lük olduğuna dikkat ediniz. Buradaki 6 değer, 6 tane noktanın olmasından gelmektedir. Pekiyi 3 değeri nereden gelmektedir? Buradaki 3 sutun tek değişkenli (örneğimiz x) ikinci derece polinomun tüm terimlerinin sayısı ilgilidir. Tek değişkenli ikinci derece polinomun genel ifadesi şöyledir:

$$y = a_0 + a_1x + a_2x^2$$

Göründüğü gibi burada toplam 3 tane tahmin edilmesi istenilen a_i katsayıları vardır. Transform edilen matrisin birinci sütunu x^0 değerini, ikinci sütunu x^1 değerini, üçüncü sütunu ise x^2 değerini belirtmektedir.

Aslında bizim tahmin etmeye çalıştığımız değerler bu a_i değerleridir. Pekiyi bağımsız değişken sayısı 2 tane olsaydı ikinci derece polinomun genel biçimini nasıl olurdu?

$$y = a_0 + a_1x_1 + a_2x_2 + a_3x_1x_2 + a_4x_1^2 + a_5x_2^2$$

Göründüğü gibi burada tahmin edilmeye çalışılacak 6 tane a_i değeri bulunmaktadır. Buradaki sütunlar da sırasıyla $x_1^0x_2^0$, x_1 , x_2 , x_1x_2 , x_1^2 , x_2^2 biçimindedir. Yani bizim "test.csv" dosyasımızdaki verilerde iki tane x değeri olsaydı bizim transform ettiğimiz matrisin 6 tane sütunu olacaktı:

$$\left[\begin{array}{cccccc} x_1^0 & x_2^0 & x_1 & x_2 & x_1x_2 & x_1^2 \\ - & - & - & - & - & - \\ - & - & - & - & - & - \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots \end{array} \right]$$

PolynomialFeatures sınıfının fit_transform metodundan elde ettiğimiz matris aslında yukarıda açıklanan sütun değerlerine ilişkindir. Pekiyi bu durumun polinomsal regresyonla ne ilgisi vardır?

İşte aslında polinomsal regresyonla çoklu (multiple) doğrusal regresyon aynı anlamdadır. Örneğin aslında tek değişkenli ikinci derece polinomsal regresyon üç değişkenli çoklu doğrusal regresyonla aynı anlama gelmektedir. İki

değişkenli ikinci derece polinomsal regresyon da 6 değişkenli çoklu doğrusal regresyon aynı anlamdadır. Örneğin tek değişkenli ikinci derece bir polinom söz konusu olsun. Bu poliomun genel biçimini şöyledir:

$$y = a_0 + a_1x + a_2x^2$$

Burada aslında doğrusal regresyon bağlamında x ve x^2 'ler değişken değil katsayı olarak ele alınabilir. Yani biz polinomu şöyle doğrusal bir biçimde dönüştürebiliriz:

$$y = a_0 + x_1a_1 + x_2a_2$$

Burada aslında x değerleri bizim gözlediğimiz değerlerdir. Biz burada a_i değerlerini bulmaya çalışmaktaiz. Şimdi değişken sayısı iki olduğunda da benzer biçimde bu da 6 terimli bir çoklu doğrusal regresyon modeline dönüştürülebilir.

$$y = a_0 + x_1a_1 + x_2a_2 + x_3a_3 + x_4a_4 + x_5a_5$$

Başka bir deyişle aslında:

$$y = AX$$

biçiminde matrisel formda gösterilmiş bir polinomsal deneklem,

$$y = XA$$

biçiminde de ifade edilebilir. Buradan hareketle bizim yapacağımız şey aslında transform edilmiş çok sütunlu değerleri doğrusal regresyona sokmak olacaktır. Örneğin:

```
import numpy as np

dataset = np.loadtxt('test.csv', dtype=np.float32, delimiter=',')
dataset_x = dataset[:, 0].reshape(-1, 1)
dataset_y = dataset[:, 1]

from sklearn.preprocessing import PolynomialFeatures

pf = PolynomialFeatures(degree=2)
transformed_dataset_x = pf.fit_transform(dataset_x)
print(transformed_dataset_x)

from sklearn.linear_model import LinearRegression

lr = LinearRegression()
lr.fit(transformed_dataset_x, dataset_y)

a0 = lr.coef_[0]
a1 = lr.coef_[1]
a2 = lr.coef_[2]

print('a0 = {}, a1 = {}, a2 = {}'.format(a0, a1, a2))

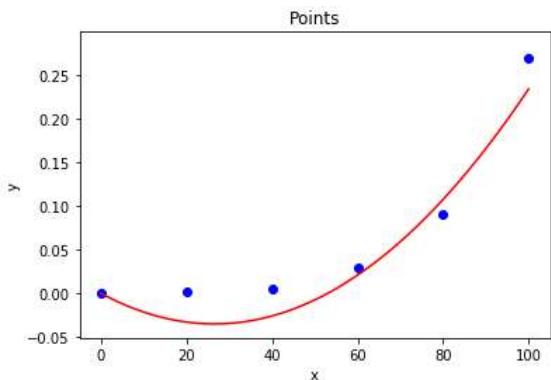
x = np.linspace(0, 100, 100)
y = a0 + a1 * x + a2 * x ** 2

import matplotlib.pyplot as plt
plt.title('Points')
plt.xlabel('x')
plt.ylabel('y')
plt.scatter(dataset[:, 0], dataset[:, 1], color='blue')
plt.plot(x, y, color='red')
```

Buradan elde ettiğimiz a_0 , a_1 , ve a_2 katsayıları şöyledir:

$$a_0 = 0.0, a_1 = -0.0026392594445496798, a_2 = 4.9812791985459626e-05$$

Elde ettiğimiz polinomun noktalarla birlikte grafiği de şöyledir:



Şimdi burada yapılanların anlamını bir daha değerlendirelim. Başlangıçta bizim elimizde tek değişkenli ikinci derece bir polinomun tahmin edilmesi problemi vardı:

$$y = a_0 + a_1 x + a_2 x^2$$

Burada bizim tahmin etmeye çalıştığımız değerler a_0 , a_1 ve a_2 değerleridir. Biz bu polinomu aşağıdaki gibi doğrusal hale getirdik:

$$y = x_0 + x_1 a_0 + x_2 a_1 + x_3 a_2$$

Aslında biz LinearRegression sınıfının predict metodu ile de elde etmeye çalıştığımız polinoma ilişkin y değerlerini bulabiliriz. Yani grafiğimizi şöyle de çizebilirdik:

```
x = np.linspace(0, 100, 100).reshape(-1, 1)
transformed_x = pf.fit_transform(x)
y = lr.predict(transformed_x)

import matplotlib.pyplot as plt
plt.title('Points')
plt.xlabel('x')
plt.ylabel('y')
plt.scatter(dataset[:, 0], dataset[:, 1], color='blue')
plt.plot(x, y, color='red')
```

Doğrusal regresyondan elde ettiğimiz katsayıları inceleyiniz:

$$a_0 = 0.0, a_1 = -0.0026392594445496798, a_2 = 4.9812791985459626e-05$$

Yani aslında bizim elde ettiğimiz polinom şöyledir:

$$y = -0.0026 x + 4.98 \times 10^{-5} x^2$$

Şimdi de polinomsal regresyonu doğrusal regresyona dönüştürdükten sonra elde ettiğimiz R^2 değerine bakalım:

```
rsquared = lr.score(transformed_dataset_x, dataset_y)
print(rsquared)
```

Elde edilen sonuç şöyledir:

Anımsanacağı gibi biz daha önce bu noktalardan doğru geçirdiğimizde 0.6903499390522619 değeri elde etmiştık. Görüldüğü gibi bu noktalardan ikinci bir derece bir polinomun geçirilmesi modeli çok daha iyileştirmiştir.

Şimdi de n sütunlu bir veri kümесinin k'inci dereceden bir polioma nasıl uydurulacağını özet olarak verelim:

1) Önce PolynomialFeatures sınıfı türünden bir nesne yaratılır. Sonra sınıfın fit_transform metodu çağrılarak transform edilmiş katsayı matrisi elde edilir:

```
pf = PolynomialFeatures(degree=k)
transformed_dataset_x = pf.fit_transform(dataset_x)
```

2) Sonra transform edilmiş değerleri LinearRegression sınıfında kullanıp fit işlemi yapmalıyız.

```
lr = LinearRegression()
lr.fit(transformed_dataset_x, dataset_y)
```

3) Artık her şey hazırır. LinearRegression sınıfının predict metoduna x değerleri verilirse y değeri elde edilir. Örneğin:

```
x_transformed = pf.fit_transform(x)
y = lr.predict(x_transformed)
```

4) k'inci derece polinomun katsayıları da istenirse lr.coef_ özniteliğinden alınır. lr.intercept özniteliğinin orijinal polinomun katsayılarıyla bir ilgisi yoktur. Yani bir deyişle aslında elde edilen polinomun katsayılı şöyledir:

```
y = lr.coef_[0] + lr.coef_[1] * x + lr.coef_[2] * x ** 2 + .... + lr.coef_[k] * x ** k
```

Pekiyi biz elimizdeki noktalardan bir doğru mu, yoksa bir polinom mu geçireceğimizi nasıl belirleyeceğiz? Ya da eğer polinom geçireceksek bunun kaçinci dereceden polinom olacağını nasıl belirleyeceğiz? Aslında bu kararı otomatik bir biçimde veren scikit-learn içerisinde bir fonksiyon bulunmaktadır. Uygulamacı noktaların grafiine bakarak sezgisel biçimde bunu belirleyebilir. Gerçekten de biz yukarıdaki örnekte noktalardan ikinci derece bir polinomun geçirilebileceğini grafiğe bakarak sezgisel biçimde belirledik. Tabii noktaların grafiğine gözle bakabilmek için değişken sayısının 1 tane olması gereklidir. Halbuki gerçek uygulamalarda değişken sayısı 1'den fazla olmaktadır. Kaçinci dereceden bir polinomun uygun olacağını belirlemek aslında deneme yanılma yoluyla yapılabilir. Yani uygulamacı ikinci derece, üçüncü derece, dördüncü derece gibi farklı derecelerde polinom geçirerek R² değerine bakabilir. Bu R² değerinin en iyi olduğu dereceyi belirleyebilir. Örneğin yukarıda üzerinde alıştığımız noktalardan üçüncü derece bir polinom geçirmeye çalışalım:

```
import numpy as np

dataset = np.loadtxt('test.csv', dtype=np.float32, delimiter=',')
dataset_x = dataset[:, 0].reshape(-1, 1)
dataset_y = dataset[:, 1]

from sklearn.preprocessing import PolynomialFeatures

pf = PolynomialFeatures(degree=3)
transformed_dataset_x = pf.fit_transform(dataset_x)
print(transformed_dataset_x)

from sklearn.linear_model import LinearRegression

lr = LinearRegression()
lr.fit(transformed_dataset_x, dataset_y)

x = np.linspace(0, 100, 100).reshape(-1, 1)
transformed_x = pf.fit_transform(x)
```

```

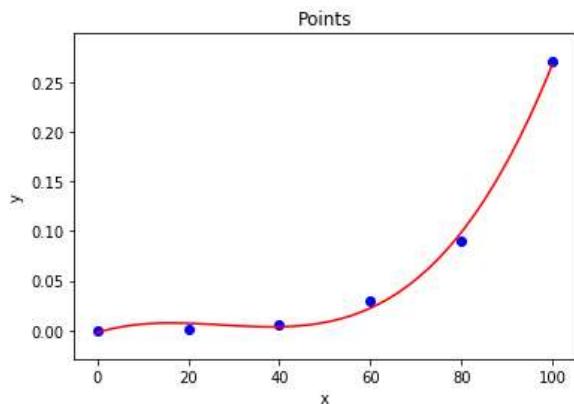
y = lr.predict(transformed_x)

import matplotlib.pyplot as plt
plt.title('Points')
plt.xlabel('x')
plt.ylabel('y')
plt.scatter(dataset[:, 0], dataset[:, 1], color='blue')
plt.plot(x, y, color='red')

rsquared = lr.score(transformed_dataset_x, dataset_y)
print(rsquared)

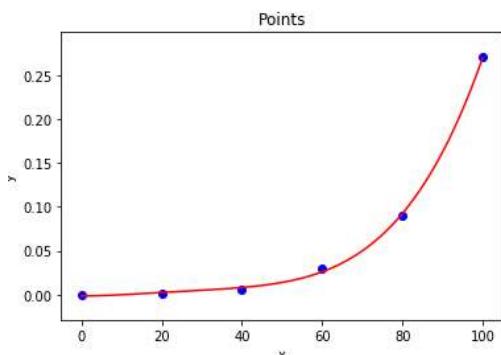
```

Buradan elde ettiğimiz grafik ve R^2 değeri şöyledir:



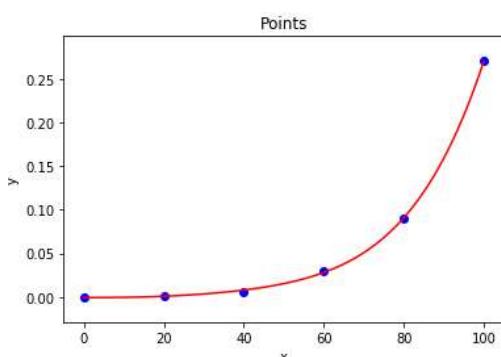
0.9966690525829125

O halde bu noktalar için üçüncü derece bir polinom aslında daha uygundur. Dördüncü derece polinom için elde edilen grafik ve R^2 değerleri de şöyledir:



0.9994477004504332

Göründüğü gibi R^2 değeri çok az daha iyileşmiştir. Ancak yüksek dereceli polinomlardan genel olarak kaçınmak gereklidir. Şimdi 5'inci derece polinom için grafik ve R^2 değerine bakalım:



0.9998423761893278

Yukarıda da belirttiğimiz gibi az bir iyileşme için derece yükseltmesi iyi bir teknik değildir. Burada üçüncü derece bir polinom uygun olabilir.

GRADIENT ASCENT ve GRADIENT DESCENT ALGORİTMALARININ ANLAMI

Gradient ascent ve gradient descent algoritmaları makine öğrenmesinde çok sık kullanılmaktadır. Anımsanacağı gibi biz de yapay sinir ağlarında optimizasyon algoritması olarak "stochastic gradient descent (sgd)" algoritmasını kullanmıştık. Stochastic gradient descent algoritması gradient descent algoritmasının bir türüdür. Pekiyi bu algoritmaların çalışma biçimini nasıldır? Aslında bu algoritmalar bir noktadan başlayarak uygun bir doğrultuda yavaş yavaş ilerleme ile karakterize olmaktadır. Yani gradient ascent ve gradient descent algoritmaları iteratifdir, en yüksek ya da en düşük optimal noktaya yavaş yavaş erişmeyi hedeflemektedir. Algoritmada doğrultu hedefe varmayı sağlayacak biçimde hesaplanmaktadır. Matematiksel anlamda eğer bir maksimizasyon problemi söz konusu ise "gradient ascent", bir minimizasyon söz konusu ise "gradient descent" yöntemi kullanılır.

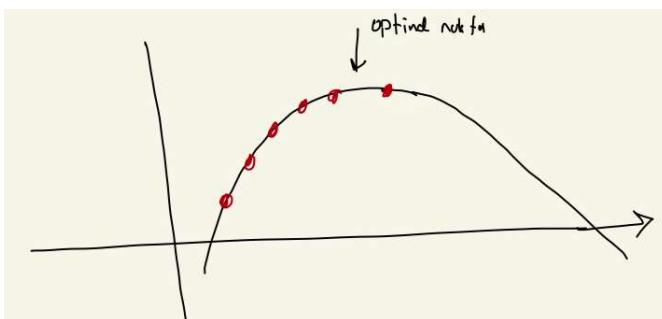
Bu yöntemlerde önce bir amaç fonksiyonu tespit edilmelidir. Yani ilerlendikçe ne hedeflenmektedir? Neyin maksimize ya da minimize edilmesi istenmektedir? Bunun bir biçimde matematiksel ifadesi ortaya konmalıdır. İkinci olarak hedefe varılıp varılmadığının tespit edilmesi gereklidir. Yani amaç fonksiyonu bir doğrultu belirtir. Fakat tatmin edici bir noktaya gelinip gelinmediği "malliyet fonksiyonu (cost function)" ya da "kayıp fonksiyonu (loss function)" denilen bir fonksiyonla belirlenir.

Yukarıda da belirtildiği gibi gradient ascent ve gradient descent algoritmalarında bir amaç, bu amaca uygun bir doğrultunun tespit edilmesi gerekmektedir. Sonra bu doğrultuda ilerlenir ve tatmin edici bir noktaya gelinip gelinmediği kontrol edilir. Matematiksel olarak doğrultu belirleme işlemi amaç fonksiyonun birinci türevi alınarak yapılmaktadır. Eğer amaç fonksiyonu birden fazla değişkenden oluşuyorsa bu durumda her değişken için parçalı türevler alınır. Bu parçalı türevlerin oluşturduğu vektöre de gradient vektör denilmektedir. Gradient vektör ters üçgenle temsil edilmektedir. Örneğin:

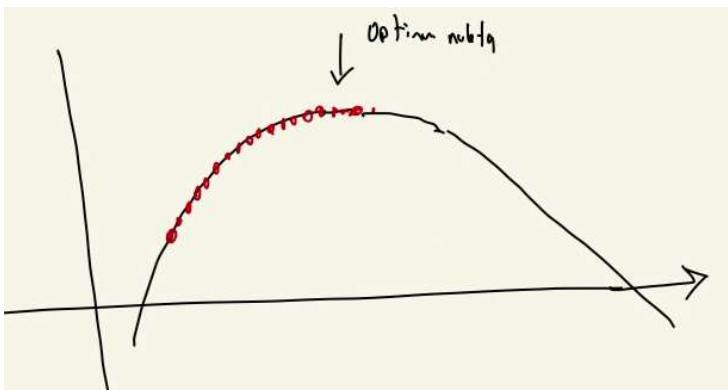
$$\nabla f = \left[\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \dots, \frac{\partial f}{\partial x_n} \right]$$

Pekiyi amaç fonksiyonu nasıl tespit edilecektir? Amaç fonksiyonunun spesifik probleme dayalı olarak tespit edilmesi gereklidir. Bu nedenle her problemin amaç fonksiyonu farklı olabilir. Benzer biçimde maaliyet (ya da kayıp) fonksiyonu da probleme dayalı biçimde tespit edilebilir. Pekiyi neden bu tür problemler denklem yoluyla değil de iteratif doğrultuda ilerleme yoluyla çözülmek istenmektedir? Bunun nedeni bazı tür problemlerde denklem çözümünün mümkün olmamasıdır.

Pek çok iteratif yöntemde ilerleme belli bir çarpansal değere orantılı olarak yapılmaktadır. Buna "öğrenme hızı" (learning rate) denilmektedir. Öğrenme hızı değer olarak yükseltilirse noktalar arasındaki sıçramalar daha yüksek olur. Hedefe daha hızlı yaklaşılır. Ancak öğrenme hızı yüksek olduğunda hedefe yaklaşım hızlı olsa da hedefin hassas bir biçimde elde edilmesi zorlaşmaktadır. Benzer biçimde öğrenme hızı düşürüldüğünde hedefin daha hassas bir biçimde bulunması sağlanır. Ancak hedefin yanına yaklaşma uzun zaman alabilemektedir. Örneğin:



Burada öğrenme hızı yüksek tutulduğu için hedefe (maksimum noktaya) hızlı yaklaşılmış ancak hedef hassas bir biçimde belirlenmemiştir. Fakat örneğin:



Burada hedefe (maksimum nokta) daha uzun sürede yaklaşılmıştır. Ancak hedef daha hassas bir biçimde belirlenebilmektedir. Biz makine öğrenmesinde "öğrenme hızı (learning rate)" biçiminde bir parametreyle karşılaşlığımızda şu çıkarımıza bulunmalıyız: "Bu parametrenin değeri yükseltilirse ben daha kısa zamanda hedefin yakınılarına gelebilirim. Böylece hedefe yaklaşmam daha az bilgisayar zamanıyla gerçekleşir. Ama tam hedefi daha az bir hassasiyetle belirleyebilirim. Eğer ben bu parametrenin değerini düşürürsem hedefe yaklaşmam daha uzun bir zaman alır ama ben hedefi daha hassas belirleyebilirim." Öğrenme hızı algoritmayı kuranın belirlediği görelî bir değerdir. Mutlak değer değildir.

Şimdi biz gradient descent yöntemiyle bir parabolün en küçük y değerini bulmak isteyelim. Bunun için bizim bir x değerinden başlayıp belli bir doğrultuda giderek en düşük değeri bulmamız gereklidir. Parabolün genel fonksiyonu şöyledir:

$$y = ax^2 + bx + c$$

Burada amacımız y değerinin en küçük yapılmasıdır. Amaç fonksiyonumuz da $ax^2 + bx + c$ fonksiyonudur. Buradaki maliyet fonksiyonu amaç fonksiyonundan elde edilen değer ile varmak istenilen değer (örneğimizde 0 olabilir) arasındaki fark olarak tanımlanabilir. Biz bu farkı belirlediğimiz bir epsilon değerinden küçük hale getirmeye çalışabiliyoruz. Gideceğimiz doğrultuyu belirlemek için gradient vektör kullanılır. Zaten bu örneğimizde tek bir değişken olduğundan gradient vektörümüzde amaç fonksiyonun türevinden oluşan tek elemanlı bir vektördür. Bu durumda gradient vektör de tek bir elemandan oluşacaktır. Örneğin:

```
def gradient_descend(a, b, c, y, learning_rate):
    x = 10
    count = 0

    while True:
        yguess = a * x ** 2 + b * x + c
        if abs(yguess - y) < 0.000001: # Loss function
            break
        x += (-2 * a * x - b) * learning_rate
        count += 1

    return x, count

x, count = gradient_descend(1, 0, -4, -4, 0.001)
print(x, count)
y = x ** 2 - 4
print(y)
```

Şimdi de doğrusal regresyon problemini gradient descent algoritmasıyla çözelim. Anımsanacağı gibi doğrusal regresyonda amaç $y = mx + n$ doğrusundaki m ve n değerlerini uygun biçimde belirlemekti. Buradaki amaç fonksiyonu şöyledi:

$$E = \frac{1}{N} \sum_{i=0}^N (y_i - (mx_i + n))^2$$

Şimdi gradient descent algoritması için bu amaç fonksiyonundaki iki değişken olan m ve n için türev alarak gradient vektörü bulmaya çalışalım:

$$\frac{\partial E}{\partial n} = \frac{-2}{N} \sum_{i=0}^N x_i (y_i - mx_i - n)$$

$$\frac{\partial E}{\partial m} = \frac{-2}{N} \sum_{i=0}^N (y_i - mx_i - n)$$

Şimdi artık maliyet fonksiyonu olmadan gradient vektördeki doğrultularda belirli bir miktarda iteratif olarak ilerleyecek biçimde kodu yazabiliriz. Aşağıdaki fonksiyonda x ve y değerleri noktaların değerleridir. epoch uygulanacak iterasyon sayısını belirtmektedir. learning_rate ise ilerleme adımlarının büyüklüğünü belirtir. Biz buarada gradient vektörde belirtilen doğrultuda küçük adımlarla doğruyu oluşturan m ve n değerlerini güncelleyeceğiz. Şüphesiz doğrusal regresyonun bu biçimde iteratif olarak çözülmesine gerek yoktur. Zaten doğrusal regresyon için "en küçük karaler" denilen kesin bir formül bulunmaktadır. Ancak biz burada bu örneği yalnızca gradient algoritmaların çalışma biçimini anlatabilmek için veriyoruz:

```
import numpy as np

dataset = np.loadtxt('test.csv', dtype=np.float32, delimiter=',')
dataset_x = dataset[:, 0]
dataset_y = dataset[:, 1]

def linear_regressin_gradient(x, y, epoch, learning_rate):
    N = len(x)
    m = 0
    n = 0

    for i in range(epoch):
        ypred = m * x + n
        dfm = (-2 / N) * np.sum(x * (y - ypred))
        dfn = (-2 / N) * np.sum(y - ypred)
        m = m - learning_rate * dfm
        n = n - learning_rate * dfn

    return m, n

m, n = linear_regressin_gradient(dataset_x, dataset_y, 10000, 0.001)

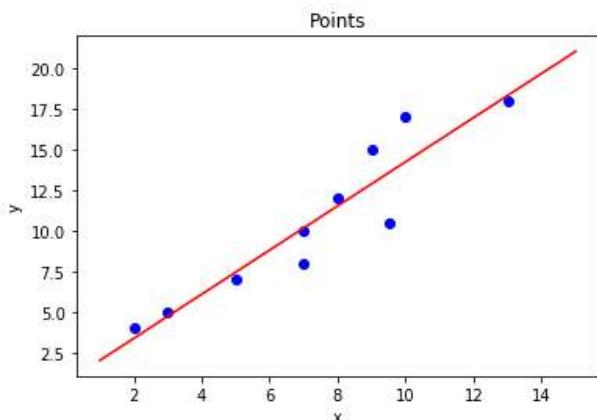
x = np.linspace(1, 15, 100)
y = m * x + n

import matplotlib.pyplot as plt

plt.title('Points')
plt.xlabel('x')
plt.ylabel('y')
plt.scatter(dataset[:, 0], dataset[:, 1], color='blue')
plt.plot(x, y, color='red')
```

```
plt.pause(1)
print(m, n)
```

Elde edilen grafik ve doğrunun m ve n değerleri şöyledir:



```
1.3620186817169206 0.6356675564765912
```

Şimdi de problemi iteratif olmayan biçimde en küçük kareler yöntemiyle LinearRegression sınıfıyla çözelim:

```
import numpy as np

dataset = np.loadtxt('test.csv', dtype=np.float32, delimiter=',')
from sklearn.linear_model import LinearRegression

lr = LinearRegression()
lr.fit(dataset[:, 0].reshape(-1, 1), dataset[:, 1])

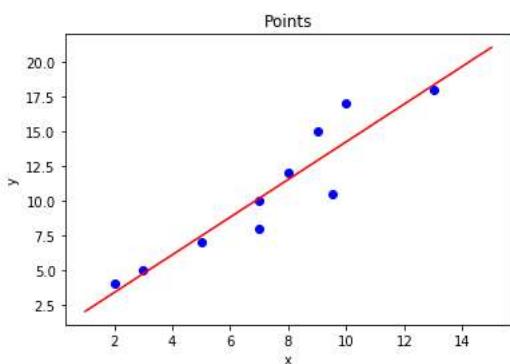
x = np.linspace(1, 15, 100)
y = lr.coef_[0] * x + lr.intercept_

import matplotlib.pyplot as plt

plt.title('Points')
plt.xlabel('x')
plt.ylabel('y')
plt.scatter(dataset[:, 0], dataset[:, 1], color='blue')
plt.plot(x, y, color='red')

plt.pause(1)
print(lr.coef_[0], lr.intercept_)
```

Elde edilen grafik ve doğrunun m ve n değerleri şöyledir:



```
1.3594104 0.6583328
```

Stokastik Gradient ve Mini Batch Yöntemleri

Gradient Ascent ve Gradient Descent yöntemlerinde doğrultuda ilerleme veri kümesindeki tüm vektör işleme sokularak yapılmaktadır. Bu ilerleme de belli miktarda bir döngüyle (epoch) devam ettirilir. Halbuki stokastik gradient ascent ve stokastik gradient descent yöntemlerinde doğrultuda ilerleme veri kümesindeki her eleman için tek tek ya da rastgele elemanlar için (stokastik sözcüğü bu nedenden kullanılmıştır) tek tek yapılır. Tüm veri setinin bu biçimde tek tek işleme sokulması bir kez yapılabileceği gibi n kez de yapılmaktadır. Böylece toplam iş miktarı stokastik versiyonda azaltılmış olmaktadır.

Veri kümesindeki tüm vektörün işleme sokulmasına "batch" işlem denilmektedir. Mini batch yöntemi bu bakımdan Gradient yöntemleriyle stokastik gradient yöntemlerinin bir ortalaması gibidir. Mini batch yönteminde tüm veriler içerisindeki belli miktarda kümeler oluşturularak bunlar işleme sokulmaktadır. Örneğin tüm veriler 1000 tane olsun. Biz bu yöntemde 1000 taneyi tek hamlede değil, tek tek de değil belirlediğimiz bir miktarda (örneğin 10'arlık) grup grubu işleme sokarız. Yine bu işlem toplamda baştan sona bir kez ya da n kez yapılmaktadır.

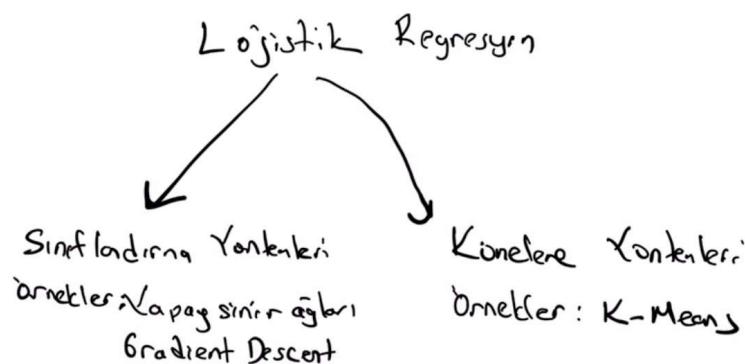
Örneğin veri kümesinde 100000 eleman bulunuyor olsun. Gradient yönteminde her iterasyonda bu 100000 eleman işleme sokulup bundan sonra ağırlıklar güncellenir. Bu işlemler de belli bir sayıda (örneğin 10000 kere) devam ettirilir. Stokastik gradient yönteminde ise bu 100000 eleman tek tek (ya da rastgele elemanlar) işleme sokulur. Bu işlem toplamda bir kez ya da n kez (örneğin 10 kere) tekrar ettirilir. Mini Batch yönteminde ise bu elemanlar küçük gruplar (örneğin 100'erli) halinde işleme sokulmaktadır. Duruma göre bu de bir kez ya da n kez tekrarlanmaktadır.

Genel olarak stokastik gradient descent yöntemi maksimum ya da minimum değere çok daha hızlı yakınsamaktadır. Ancak gradient yöntem stokastik yönteme göre daha iyi sonuç verir. Veri miktarının az olduğu durumlarda stokastik gradient yerine normal gradient yöntem daha uygun kaçmaktadır. (Örneğin 5 noktadan oluşan doğrusal regresyonda stokastik gradient normal gradient yönteme göre çok daha kötü sonuç vermektedir.) Ayrıca veri miktarı az ise stokastik gradient yönteminde learning_rate değerini yükseltmek daha iyi sonuç verebilmektedir.

Gradient Descent Sınıflandırma Yöntemiyle Lojistik Regresyon

Anımsanacağı gibi çıktısı iki değerden oluşan ve sınıflandırma amacıyla kullanılan regresyon modeline lojistik regresyon modeli denilmektedir. (Sonraları çıktı değerinin ikiden fazla olduğu modeller de lojistik regresyon olarak adlandırılabilir.)

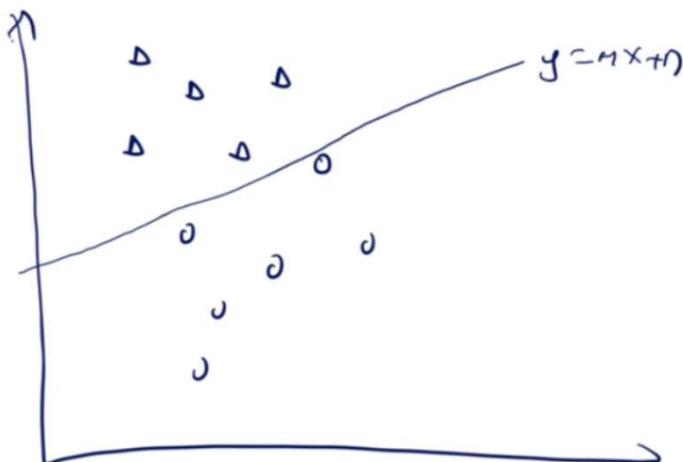
Lojistik regresyon problemlerinin çözümü için genel olarak iki ana yöntem grubundan faydalankmaktadır: Sınıflandırma (classification) ve Kümeleme (clustering). Daha önce görmüş olduğumuz sinir ağları sınıflandırma yöntemine bir örnek oluştururken K-Means ya da DBSCAN yöntemleri kümeleme yöntemlerine örnek oluşturmaktadır.



Sınıflandırma tarzı lojistik regresyon yöntemleri genel olarak denetimli (supervised) yöntemlerdir. Halbuki kümeleme tarzı lojistik regresyon yöntemleri denetimsiz (unsupervised) yöntemlerdir.

Lojistik regresyon problemleri yapay sinir ağları ve kümeleme yöntemi kullanılmadan denetimli biçimde gradient descent yöntemiyle de çözülebilmektedir.

Aslında lojistik regresyon iki kümeyi ayıran bir doğru denkleminin belirlenmesi ile de yapılabilir. Örneğin:



İki kümeyi en iyi biçimde ayıran doğru denklemi elde edilirse hem bir sınıflandırma yapılmış olur hem de yeni değerlerin hangi sınıf içerisinde gireceği yönünde bir kestirim aracı elde edilmiş olur. Kestirim sınıfı belirlenecek noktanın doğrunun yukarısında mı yoksa aşağısında mı kaldığına göre yapılmaktadır. Aslında buradaki yaklaşım yukarıda yaptığımız doğrusal regresyon örneğine benzemektedir.

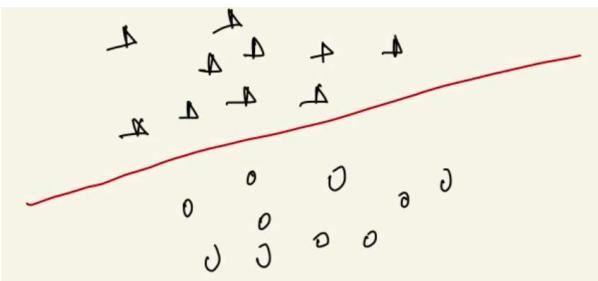
Bu durumda biz birtakım değerleri benzerliklerine göre grupperlendirmek bu kursta gördüğümüz yöntemler sunlar olacaktır:

- Yapay Sinir Ağları (Eğitimli – Sınıflandırma Tarzı)
- K-Means (Eğitsiz – Kümeleme Tarzı)
- DBScan (Eğitsiz – Kümeleme Tarzı)
- Gradient Descent (Eğitimli – Sınıflandırma Tarzı)

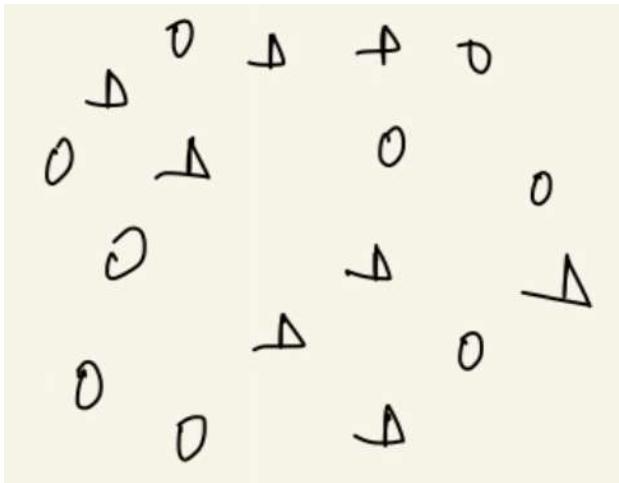
Pekiyi grupperlendirme amacıyla biz bu yöntemlerden hangisini ne zaman tercih etmeliyiz?

- Eğer elimizde yalnızca dataset_x varsa ama bunların hangi gruppala girdiğine yönelik dataset_y yoksa biz mecburen K-Means ya da DBSCAN gibi kümeleme yöntemlerini kullanırız.
- Eğer elimizde dataset_x verilerinin yanı sıra bunların zaten hangi grupparda olduğunu belirten bir dataset_y verileri de varsa biz yapay sinir ağları ve gradient descent sınıflandırma yöntemlerini kullanabiliriz. (Tabii aslında eğitim için gereken dataset_y verileri olsa da biz bunlardan faydalananmamış yine K-Means ve DBSCAN gibi kümeleme yöntemlerini kullanabiliriz. Ancak genel olarak eğitimli sınıflandırma yöntemleri bu durumda kümeleme yöntemlerine göre tercih edilmektedir.)

Pekiyi elimizde eğitimli öğrenme için kullanabileceğimiz dataset_x ve dataset_y verileri bulunuyor olsun. Bu durumda yapay sinir ağları mı yoksa izleyen bölümde göreceğimiz gradient descent yöntemi mi tercih edilmelidir? İşte eğer veriler "doğrusal olarak ayrılabilen (linearly separable)" biçimdeyse yani bir doğru ile iki sınıf birbirlerinden ayrılabiliriyorsa gradient descent yöntemi daha hızlı ve uygun olmaktadır. Ancak veriler bir doğru ile iki kümeye ayrılamıyorsa bu durumda gradient descent yönteminin performansı düşmektedir. Yapay sinir ağları bu tür durumlarda daha uygun olmaktadır. Örneğin aşağıdaki veriler doğrusal olarak ayrılabilen biçimdedir:

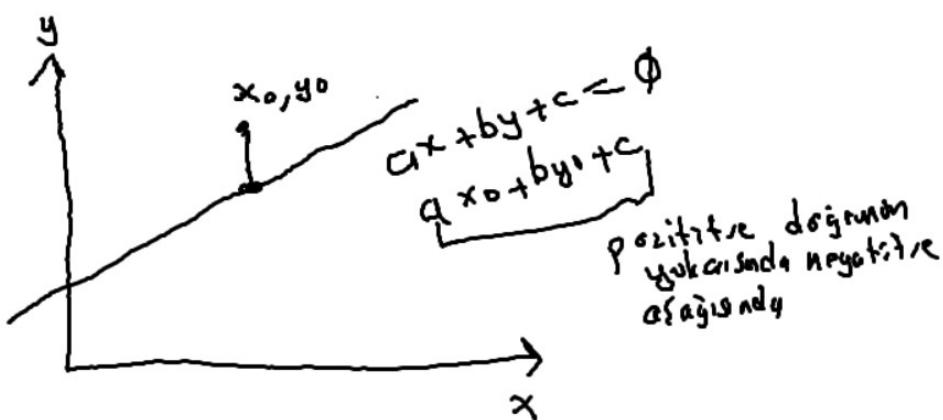


Burada biz yapay sinir ağı yerine gradient descent yöntemini tercih edebiliriz. Aşağıdaki veriler ise doğrusal olarak ayrılabilir olmayan biçimdedir:



Burada biz bir doğru denklemi elde ederek verileri iyi bir biçimde sınıflandıramayız. İşte bu tür durumlarda yapay sinir ağları tercih edilmelidir. Maalesef sınıflandırma problemlerinin önemli bir bölümü "doğrusal olarak ayrılabilir" biçimde değildir.

Pekiyi iki sınıfı birbirinden ayırmak için elde edeceğimiz doğru denkleminde amaç fonksiyonu ne olmalıdır? Yani biz iterasyonlar sırasında neyi küçütmeye çalışmalıyız? İşte burada iki kümeyi ayıran doğru denklemının genel biçimi $ax + by + c = 0$ olarak ifade edilebilir. Bir noktayı bu doğru denkleminde x ve y olarak yerleştirdiğimizde çıkan değer bu noktanın doğruya uzunluğunu verir.



Biz bu yöntemde denetimli öğrenme uygulayacağımıza göre gerçek değerlerle buradaki uzunluk arasında bir ilişki kurmamız gereklidir. Bunun da en basit yolu sigmoid fonksiyonunu kullanmaktır. Yani biz noktanın doğruya uzunluğunu sigmoid fonksiyonuna sokarsak bu fonksiyon bize 0 ile 1 arasında bir değer verecektir. Ve aynı zamanda söz konusu olan nokta doğrunun ne kadar yukarıındaysa 1'e o kadar yaklaşacak, ne kadar aşağıındaysa 0'a o kadar yaklaşacaktır. O halde amaç fonksiyonu gerçek değerlerden bu sigmoid değerinin çıkartılması biçiminde

oluşturulabilir. Biz de bu amaç fonksiyonunu minimize etmeye çalışabiliriz. Tabii bu işlemi yavaş yavaş bir doğrultu biçiminde gradient descent yöntemiyle yaparız. Çözüme bu biçimde varan bir program şöyle yazılabilir:

```
import numpy as np

dataset = np.loadtxt('test.csv', skiprows=1, delimiter=',', dtype=np.float32)
dataset_x = dataset[:, :2]
dataset_y = dataset[:, 2].reshape(-1, 1)

dataset_x = np.append(dataset_x, np.ones((dataset_x.shape[0], 1)), axis=1)

def sigmoid(x):
    return 1.0 / (1 + np.exp(-x))

def gradeint_descent_logistic(x, y, learning_rate=0.001, epoch=50000):
    weights = np.ones((x.shape[1], 1))

    for k in range(epoch):
        h = sigmoid(np.matmul(x, weights))
        error = y - h
        weights = weights + learning_rate * np.matmul(x.transpose(), error)

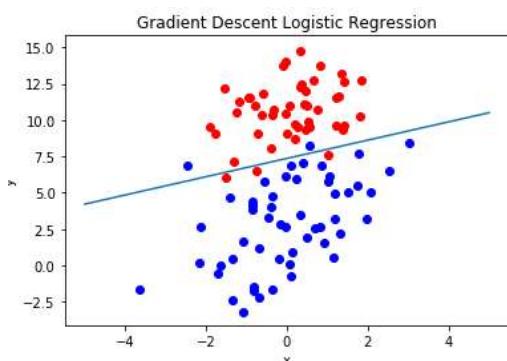
    return weights

weights = gradeint_descent_logistic(dataset_x, dataset_y)

import matplotlib.pyplot as plt

plt.title("Gradient Descent Logistic Regression")
plt.scatter(dataset[dataset[:, 2] == 1, 0], dataset[dataset[:, 2] == 1, 1], color='b')
plt.scatter(dataset[dataset[:, 2] == 0, 0], dataset[dataset[:, 2] == 0, 1], color='r')
plt.xlabel("x")
plt.ylabel("y")

x = np.linspace(-5, 5, 100)
y = (-weights[2] - weights[0] * x) / weights[1]
plt.plot(x, y)
```



Buradaki "test.csv" dosyasının içeriği şöyledir:

```
x,y,class
-0.017612,14.053064,0
-1.395634,4.662541,1
-0.752157,6.538620,0
-1.322371,7.152853,0
0.423363,11.054677,0
0.406704,7.067335,1
0.667394,12.741452,0
-2.460150,6.866805,1
0.569411,9.548755,0
-0.026632,10.427743,0
0.850433,6.920334,1
```

1.347183,13.175500,0
1.176813,3.167020,1
-1.781871,9.097953,0
-0.566606,5.749003,1
0.931635,1.589505,1
-0.024205,6.151823,1
-0.036453,2.690988,1
-0.196949,0.444165,1
1.014459,5.754399,1
1.985298,3.230619,1
-1.693453,-0.557540,1
-0.576525,11.778922,0
-0.346811,-1.678730,1
-2.124484,2.672471,1
1.217916,9.597015,0
-0.733928,9.098687,0
-3.642001,-1.618087,1
0.315985,3.523953,1
1.416614,9.619232,0
-0.386323,3.989286,1
0.556921,8.294984,1
1.224863,11.587360,0
-1.347803,-2.406051,1
1.196604,4.951851,1
0.275221,9.543647,0
0.470575,9.332488,0
-1.889567,9.542662,0
-1.527893,12.150579,0
-1.185247,11.309318,0
-0.445678,3.297303,1
1.042222,6.105155,1
-0.618787,10.320986,0
1.152083,0.548467,1
0.828534,2.676045,1
-1.237728,10.549033,0
-0.683565,-2.166125,1
0.229456,5.921938,1
-0.959885,11.555336,0
0.492911,10.993324,0
0.184992,8.721488,0
-0.355715,10.325976,0
-0.397822,8.058397,0
0.824839,13.730343,0
1.507278,5.027866,1
0.099671,6.835839,1
-0.344008,10.717485,0
1.785928,7.718645,1
-0.918801,11.560217,0
-0.364009,4.747300,1
-0.841722,4.119083,1
0.490426,1.960539,1
-0.007194,9.075792,0
0.356107,12.447863,0
0.342578,12.281162,0
-0.810823,-1.466018,1
2.530777,6.476801,1
1.296683,11.607559,0
0.475487,12.040035,0
-0.783277,11.009725,0
0.074798,11.023650,0
-1.337472,0.468339,1
-0.102781,13.763651,0
-0.147324,2.874846,1
0.518389,9.887035,0
1.015399,7.571882,0
-1.658086,-0.027255,1
1.319944,2.171228,1
2.056216,5.019981,1
-0.851633,4.375691,1
-1.510047,6.061992,0
-1.076637,-3.181888,1
1.821096,10.283990,0
3.010150,8.401766,1
-1.099458,1.688274,1
-0.834872,-1.733869,1
-0.846637,3.849075,1
1.400102,12.628781,0
1.752842,5.468166,1
0.078557,0.059736,1

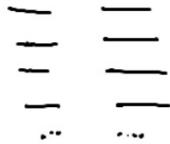
```

0.089392, -0.715300, 1
1.825662, 12.693808, 0
0.197445, 9.744638, 0
0.126117, 0.922311, 1
-0.679797, 1.220530, 1
0.677983, 2.556666, 1
0.761349, 10.693862, 0
-2.168791, 0.143632, 1
1.388610, 9.341997, 0
0.317029, 14.739025, 0

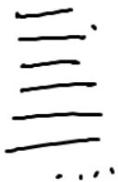
```

Bu programda yapılanlar şunlardır:

- Önce "test.csv" dosyasından değerler okunmuştur. Bu işlemin sonucunda dataset_x ve dataset_y biçiminde iki ndarray elde edilmiştir. dataset_x iki sütunlu aşağıdaki biçimde bir matriztir:



dataset_y ise tek sütunlu aşağıdaki biçimsel görünümü sahiptir:



dataset_y 0 ve 1 değerlerinden oluşmaktadır. Bu değerler sınıfları belirtmektedir.

- Bizim eğitimden amacımız aslında toplamda $ax + by + c$ doğrusundaki a, b, c değerlerini bulmaktır. Bunun için dataset_x'ye bir sütun daha ekleyip (c için) onu şu görünümü getirdik:



- Bundan sonra programda a, b, c değerleri için weights isminde 3 elemanlı bir sütun vektörü oluşturduk.



- Daha sonra programda noktaların weights katsayılarıyla belirtilen doğruya uzunlukları hesaplanmış, bunlar sigmoid fonksiyonuna sokulup gerçek y değerleri ile farkları hesaplanmıştır:

```

for k in range(epoch):
    h = sigmoid(np.matmul(x, weights))
    error = y - h
    weights = weights + learning_rate * np.matmul(x.transpose(), error)

```

- Buradaki amaç fonksiyonu (yani minimize edilecek fonksiyon) $y - h$ yani error fonksiyonudur. Görüldüğü gibi her yinelemede x matrisi ile error değeri çarpılarak weights güncellenmiştir.

Pekiyi yukarıdaki örnekte sistemin başarısını nasıl test edebiliriz. Bunun için ilk akla gelen yöntem başarılı olanların oranını tespit etmektir. Bu tespit şöyle yapılabilir:

```
result = np.matmul(dataset_x, weights)
success_ratio = np.sum((result > 0) == dataset_y) / len(dataset_y)
print(success_ratio)
```

Buradan 0.95 değeri elde edilmiştir.

Yukarıdaki yöntem stochastic gradient descent uygulanarak aşağıdaki biçimde dönüştürülebilir:

```
import numpy as np

dataset = np.loadtxt('test.csv', skiprows=1, delimiter=',', dtype=np.float32)
dataset_x = dataset[:, :2]
dataset_y = dataset[:, 2].reshape(-1, 1)

dataset_x = np.append(dataset_x, np.ones((dataset_x.shape[0], 1)), axis=1)

def sigmoid(x):
    return 1.0 / (1 + np.exp(-x))

def stochastic_gradient_descent_logistic(x, y, learning_rate=0.001, epoch=150):
    weights = np.ones(x.shape[1])

    for i in range(epoch):
        for k in range(x.shape[0]):
            learning_rate = 4 / (1 + i + k) + 0.01
            index = int(np.random.randint(0, x.shape[0]))
            h = sigmoid(np.matmul(x[index], weights))
            error = y[index] - h
            weights = weights + learning_rate * error * x[index]

    return weights

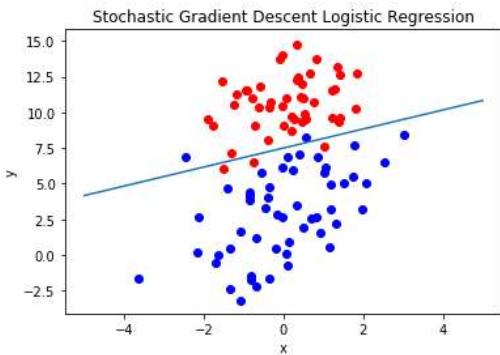
weights = stochastic_gradient_descent_logistic(dataset_x, dataset_y)

import matplotlib.pyplot as plt

plt.title("Stochastic Gradient Descent Logistic Regression")
plt.scatter(dataset[dataset[:, 2] == 1, 0], dataset[dataset[:, 2] == 1, 1], color='b')
plt.scatter(dataset[dataset[:, 2] == 0, 0], dataset[dataset[:, 2] == 0, 1], color='r')
plt.xlabel("x")
plt.ylabel("y")

x = np.linspace(-5, 5, 100)
y = (-weights[2] - weights[0] * x) / weights[1]
plt.plot(x, y)

result = np.matmul(dataset_x, weights)
success_ratio = np.sum((result > 0) == dataset_y) / len(dataset_y)
print(success_ratio)
```



Buradan elde edilen başarı oranı %50.4 bulunmuştur. Şimdi bu yöntemde learning_rate'in de belli bir aralıkta değiştirilmesiyle iyileştirilmiş biçimiyle oluşturulabilir:

```

import numpy as np

dataset = np.loadtxt('test.txt')
dataset_x = dataset[:, :2]
dataset_y = dataset[:, 2].reshape(-1, 1)

dataset_x = np.append(dataset_x, np.ones((dataset_x.shape[0], 1)), axis=1)

def sigmoid(x):
    return 1.0 / (1 + np.exp(-x))

def gradeint_descent_logistic(x, y, epoch=50):
    weights = np.ones(x.shape[1])

    for i in range(epoch):
        for k in range(x.shape[0]):
            learning_rate = 4 / (1 + i + k) + 0.01
            index = int(np.random.randint(0, x.shape[0]))
            h = sigmoid(np.matmul(x[index], weights))
            error = y[index] - h
            weights = weights + learning_rate * error * x[index]

    return weights

weights = gradeint_descent_logistic(dataset_x, dataset_y)

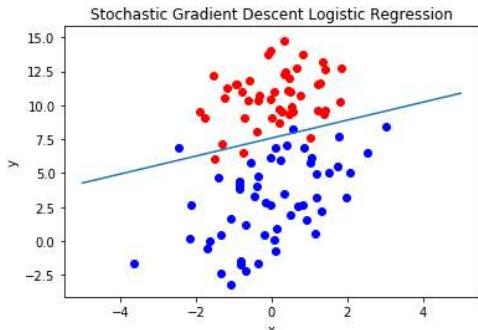
import matplotlib.pyplot as plt

plt.title("Stochastic Gradient Descent Logistic Regression")
plt.scatter(dataset[dataset[:, 2] == 1, 0], dataset[dataset[:, 2] == 1, 1], color='b')
plt.scatter(dataset[dataset[:, 2] == 0, 0], dataset[dataset[:, 2] == 0, 1], color='r')
plt.xlabel("x")
plt.ylabel("y")

x = np.linspace(-5, 5, 100)
y = (-weights[2] - weights[0] * x) / weights[1]
plt.plot(x, y)

result = np.matmul(dataset_x, weights)
success_ratio = np.sum((result > 0) == dataset_y) / len(dataset_y)
print(success_ratio)

```



Buradaki başarı değeri 9.95 bulunmuştur.

Scikit-Learn'deki Hazır LogisticRegression Sınıfının Kullanılması

Aslında Scikit-Learn içerisinde zaten gradient descent yöntemiyle lojistik regresyon işlemini yapan LogisticRegression isimli yetenekli bir sınıf vardır. Bu sınıfın kullanılması oldukça kolaydır. Tek yapacağımız şey LogisticRegression sınıfı türünden bir nesne yaratmak ve bu nesnelerle fit işlemini uygulamaktır. Bundan sonra artık predict metoduyla tahminleme yapabiliriz. Sınıfın score isimli metodu yapılan regresyonun başarı yüzdesini bize vermektedir. Yine burada elde edilen doğru denkleminin katsayılarını biz sınıfın coef_ isimli örnek özniteligidenden, eksen kesim noktasını da sınıfın intercept_ örnek özniteligidenden elde edebiliriz. Yukarıdaki örneğin LogisticRegression sınıfıyla gerçekleştirmi şöyleden olabilir:

```
import numpy as np

dataset = np.loadtxt('test.csv', skiprows=1, delimiter=',', dtype=np.float32)
dataset_x = dataset[:, :2]
dataset_y = dataset[:, 2].reshape(-1, 1)

from sklearn.linear_model import LogisticRegression

lr = LogisticRegression()
lr.fit(dataset_x, dataset_y)

import matplotlib.pyplot as plt

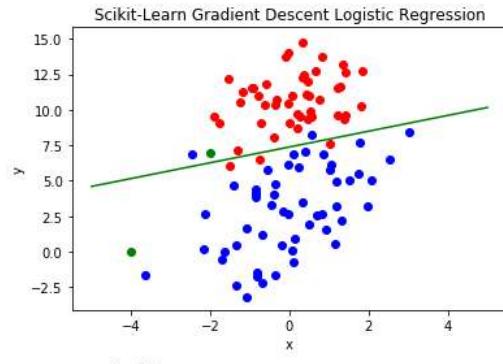
plt.title("Scikit-Learn Gradient Descent Logistic Regression")
plt.xlabel("x")
plt.ylabel("y")
plt.scatter(dataset[dataset[:, 2] == 1, 0], dataset[dataset[:, 2] == 1, 1], color='b')
plt.scatter(dataset[dataset[:, 2] == 0, 0], dataset[dataset[:, 2] == 0, 1], color='r')
x = np.linspace(-5, 5, 100)
y = (-lr.coef_[0, 0] * x - lr.intercept_[0]) / lr.coef_[0, 1]
plt.plot(x, y, color='green')

predict_points = np.array([-2, 7], [-4, 0])

plt.scatter(predict_points[:, 0], predict_points[:, 1], color='g')

plt.pause(1)
score = lr.score(dataset_x, dataset_y)
print('score = {}'.format(score))

result = lr.predict(predict_points)
print('{} point prediction = {}'.format(predict_points, result))
```



```
score = 0.95
[[ -2  7]
 [ -4  0]] point prediction = [0. 1.]
```

LogisticRegression nesnesini yaratırken aslında `__init__` metodunda pek çok argüman da girebiliriz. Bu argümanların hepsi default değer almış durumdadır. Aşağıda fonksiyonun parametre listesi görülmektedir:

```
class sklearn.linear_model.LogisticRegression(penalty='L2', dual=False, tol=0.0001, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None, solver='Lbfgs', max_iter=100, multi_class='auto', verbose=0, warm_start=False, n_jobs=None, l1_ratio=None)
```

Buradaki `max_iter` parametresi gradient descent uygulanırken kaç yineleme yapılacağını (yani bizim örneğimizdeki epoch) belirtmektedir. Bu parametrenin default değerinin 100 olduğunu dikkat ediniz. Fonksiyonun `solver` parametresi çözüm algoritmasını belirlemek için kullanılmaktadır. Küçük veri kümesi için bu parametrenin 'liblinear' girilmesi uygun olabilmektedir.

Lojistik regresyon uygulamakarında veri bilimcisi gerçek değerlerle tahmin edilen değerler arasındaki farkı görmek isteyebilir. Örneğin gerçekte 1 olması gereken değerlerin kaç tanesi 1 kaç tanesi 0 olmuştur gibi. İşte bunu gösteren matrise "confusion matrix" denilmektedir. Confusion matrix şüphesiz manuel biçimde oluşturulabilir. Ancak Scikit-Learn içerisinde `sklearn.metrics` modülünde bu matrisi oluşturan hazır bir `confusion_matrix` isimli fonksiyon vardır. Bu fonksiyon bizden iki parametre ister. Birinci parametre gerçek y değerlerini ikinci parametre ise tahmin edilmiş olan y değerlerini belirtmektedir. Yukarıdaki örneğimiz için confusion matrix şöyle oluşturulmuştur:

```
from sklearn.metrics import confusion_matrix

result = confusion_matrix(dataset_y, lr.predict(dataset_x))
print(result)

[[44  3]
 [ 2 51]]
```

Burada sınıflandırmada kullanılacak sınıf sayısı iki olduğu için confusion matrix 2x2 boyutundadır. Genel olarak sınıf sayısı n olmak üzere bu matris nxn boyutunda olur. Confusion matrix'te (i, j) hücresi gerçek durumun i olması halinde bunun j biçiminde tespit edildiği nokta sayısını vermektedir. Örneğin yukarıdaki matriste gerçek sonuç 0 iken 0 sonucu 44 kez elde edilmiştir. Öte yandan gerçek sonuç 0 iken 1 sonucu 3 kez elde edilmiştir. Benzer biçimde gerçek sonuç 1 iken 0 sonucu 2 kez, gerçek sonuç 1 iken 1 sonucu 51 kez elde edilmiştir. Gerçekten de grafik incelendiğinde kırmızıların 3 tanesinin doğrunun altında olduğu, mavilerin de 2 tanesinin doğrunun üzerinde olduğu görülmektedir.

Şüphesiz biz bu confusion matrix'i nokta sayısına bölersek oransal bir matris elde edebiliriz:

```
result_ratio = result / len(dataset_y)
print(result_ratio)

[[0.44 0.03]
 [0.02 0.51]]
```

Scikit-Learn kütüphanesindeki hazır LogisticRegression sınıfı aslında çok daha genel gerçekleştirilmiştir. Biz bu sınıfla sonraki konularda da ele alınacağı gibi çok sütundan oluşan ve birden fazla sınıftan ve etiketten oluşan lojistik regresyon problemlerini de çözebiliriz. Önceki konularda da belirtildiği gibi lojistik regresyon denildiğinde ikili sınıflandırma anlamaktadır. Fakat LogisticRegression sınıfı aslında çok çok sınıflı lojistik regresyon modellerinde de kullanılabilmektedir. İzleyen bölümlerde böyle çok sınıflı lojistik regresyon örnekleri verilemektedir. Ayrıca Scikit-Learn kütüphanesindeki LogisticRegression sınıfı çok etiketli (çok değişkenli) sınıflandırmalar için de kullanılabilmektedir.

Çok Sınıflı Gradient Descent Lojistik Regresyon İçin IRIS Örneği

Anımsanacağı gibi Iris (Zambak) veri kümesi 4 özellikten hareketle zambakların türünü 3 seçenekten biri biçiminde belirleyebilmek için oluşturulmuş bir örnek veri kümesidir. Biz daha önce bu veri kümesini doğrudan dosyadan yükleyerek kullanmıştık. Aslında bu veri kümesi hazır biçimde Scikit-Learn içerisinde de bulunmaktadır. Aşağıda Iris veri kümesi için lojistik regresyon örneği verilmiştir:

```
from sklearn.datasets import load_iris

iris = load_iris()
dataset_x = iris.data
dataset_y = iris.target

from sklearn.model_selection import train_test_split

training_dataset_x, test_dataset_x, training_dataset_y, test_dataset_y =
train_test_split(dataset_x, dataset_y, test_size=0.25)

from sklearn.linear_model import LogisticRegression

lr = LogisticRegression()
lr.fit(training_dataset_x, training_dataset_y)

result = lr.score(test_dataset_x, test_dataset_y)
print('score = {}'.format(result))

from sklearn.metrics import confusion_matrix

predicted_test_y = lr.predict(test_dataset_x)
cm = confusion_matrix(test_dataset_y, predicted_test_y)
print(cm)
```

Buradan elde edilen skor ve confusion_matrix şöyledir:

```
score = 0.9736842105263158
[[13  0  0]
 [ 0 12  1]
 [ 0  0 12]]
```

Göründüğü gibi test verileri ile yapılan denemede yalnızca 1 numaralı tür 1 kere 2 olarak tespit edilmiştir.

Iris örneğindeki %97'lik bu başarı Iris verilerinin "doğrusal olarak ayırtılabilir (linearly seperable)" olduğunu göstermektedir. Eğer Iris verileri doğrusal olarak ayırtılabilir olmasaydı bizim başarımız düşük kalırdı. Böylece biz burada gradient descent lojistik regresyon kullanmamız gerektiğini anladık ve yapay sinir ağları ya da ileride göreceğimiz "destek vektör makineleri (support vector machines)" gibi alternatif yöntemleri denememiz uygun olurdu.

Buradaki Iris örneğinde coef_ ve intercept_ örnek özniteliklerini yazdırduğumuzda şunları görmekteyiz:

```

print(lr.coef_)
print()
print(lr.intercept_)

[[-0.44695305  0.7897291 -2.34012927 -0.95547692]
 [ 0.60398165 -0.30810544 -0.19988568 -0.93823562]
 [-0.15702859 -0.48162367  2.54001496  1.89371254]]

[ 9.86891014  1.52164148 -11.39055162]

```

Burada katsayı matrisinin 3×4 'lük olduğunu görürsünüz. Matrisin 3 satırdan oluşması sonraki konuda açıklandığı gibi 3 sınıfın 3 doğru ile ayrıştırılmasından kaynaklanmaktadır. Yani 3 sınıfı bir lojistik regresyonda biz bu 3 sınıfı 3 ayrı doğruya ayrırtırabilmekteyiz. Katsayı matrisinin 4 sütundan oluşması ise regresyonda 4 tane değişkenin (yani sütunun) bulunuyor olmasındandır. Eksen kesim matrisinin 1×3 'lük olduğunu da dikkat ediniz. Her doğrunun bir tane eksen kesim noktası bulunmaktadır.

Çok Sınıflı Lojistik Regresyon İçin Mnist Örneği

Anımsanacağı gibi Mnist veri kümesi her biri 28×28 pixelden oluşan gray-scale resimlerden oluşuyordu. Bu resimlerde 0-9 arasındaki sayılar bulunmaktadır. Bizim amacımız da yeni bir resmi verdığımızda bunun üzerindeki sayının belirlenmesiydi. Biz Mnist örneğini daha önce yapay sinir ağları konusunda yapmıştık. Şimdi aynı örneği LogisticRegression sınıfıyla yapalım:

```

from tensorflow.keras.datasets import mnist

(training_dataset_x, training_dataset_y), (test_dataset_x, test_dataset_y) = mnist.load_data()

training_dataset_x = training_dataset_x.reshape(-1, 28 * 28)[:2000, :]
training_dataset_y = training_dataset_y[:2000]
test_dataset_x = test_dataset_x.reshape(-1, 28 * 28)

from sklearn.linear_model import LogisticRegression

lr = LogisticRegression(verbose=2)
lr.fit(training_dataset_x, training_dataset_y)

result = lr.score(test_dataset_x, test_dataset_y)
print('score = {}'.format(result))

```

Elde ettiğimiz başarı skoru şöyledir:

```
score = 0.9255
```

Bu yüksek başarı değerinden yine Mnist verilerinin "doğrusal olarak ayırtırılabilir (linearly separable)" olduğunu görüyorsunuz. Confusion matrix ise şöyledir:

```

from sklearn.metrics import confusion_matrix

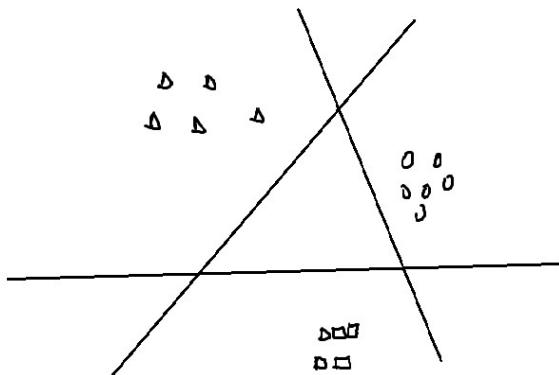
predicted_test_y = lr.predict(test_dataset_x)
cm = confusion_matrix(test_dataset_y, predicted_test_y)
print(cm)

```

```
[[ 963    0    0    3    1    3    4    4    2    0]
 [  0 1112    4    2    0    1    3    2   11    0]
 [  3   10  926   15    6    4   15    8   42    3]
 [  4   1   21  916    1   26    3    9   22    7]
 [  1   1    7    3  910    0    9    7   10   34]
 [ 11   2    1   33   11  776   11    6   35    6]
 [  9   3    7    3   7   16  910    2    1    0]
 [  1   6   24    5   7   1    0  951    3   30]
 [  8   7    6   23    6   26   10   10  869    9]
 [  9   7    0   11   25    6    0   22    7  922]]
```

Çok Sınıflı Lojistik Regresyonlarda Regresyon Doğruları

Daha önce çok sınıflı (yani sınıf sayısı 2'den fazla olan) lojistik regresyonlar için sınıf sayısı kadar doğru oluşturulduğunu belirtmiştim. Örneğin Iris verilerinde zambaklar 3 sınıfından birine sokulmak istenmektedir. Yani Iris örneğindeki sınıf sayısı 3'tür. Bu durumda LogisticRegression sınıfı bize 3 tane doğru vermelidir. Gerçekten de anımsanacağı gibi bu problemde `coef_` matrisi 3X4'lük boyuttaydı. Tabii her doğru eksenleri kestiğine göre her doğru için bir `intercept_` olması gereklidir. Bu durumda Iris örneğinde bize verilen `intercept_` vektörü de 3 elemanlıdır. Peki neden n sınıf için n tane doğru gerekmektedir? Aslında bu n doğrunun her biri bir sınıfı diğer tümünden ayırmak için kullanılmaktadır.



Çok sınıflı lojistik regresyon problemlerinin "doğrusal olarak ayrılabilir (linearly separable)" olması aslında her bir sınıfın yalnızca bir doğru ile diğerlerinden ayrılabilir olması anlamına gelmektedir. Daha önce belirttiğim gibi doğrusal olarak ayırtılamayan sınıflandırma problemlerinde gradient descent yöntemiyle lojistik regresyonun başarısı düşük olacaktır.

Şimdi biz bu doğruları elde edebilmek için 2 sütunlu 3 sınıfı bir lojistik regresyon örneği yapalım:

```
import numpy as np

from sklearn.datasets.samples_generator import make_blobs

dataset_x, dataset_y = make_blobs(n_samples=100, n_features=2, centers=3, cluster_std=1)

from sklearn.linear_model import LogisticRegression

lr = LogisticRegression()
lr.fit(dataset_x, dataset_y)
from sklearn.datasets.samples_generator import make_blobs

dataset_x, dataset_y = make_blobs(n_samples=100, n_features=2, centers=3,
cluster_std=0.5)

from sklearn.linear_model import LogisticRegression

lr = LogisticRegression()
```

```

lr.fit(dataset_x, dataset_y)

score = lr.score(dataset_x, dataset_y)
print(score)

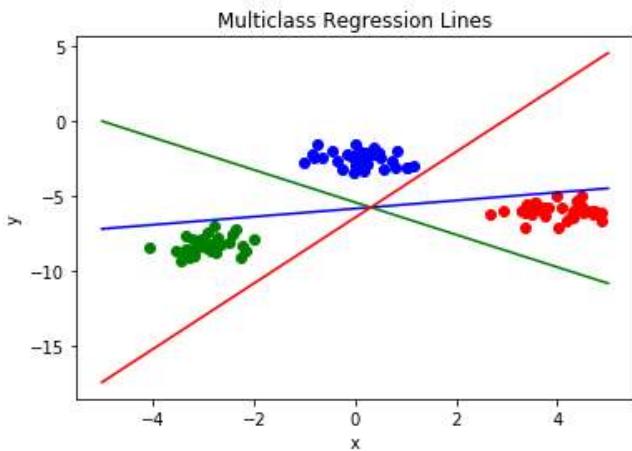
import numpy as np
import matplotlib.pyplot as plt

plt.title('Multiclass Regression Lines')
plt.xlabel('x')
plt.ylabel('y')

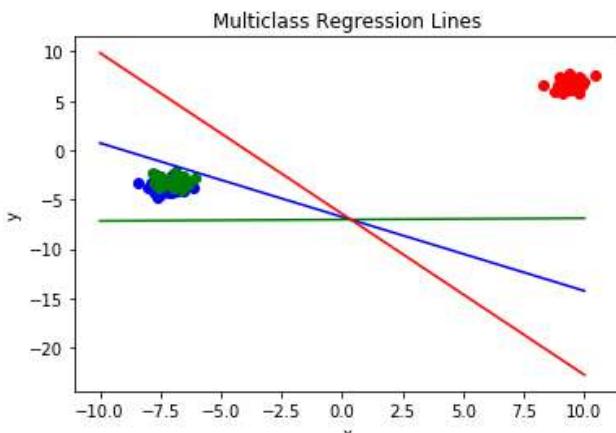
plt.scatter(dataset_x[dataset_y == 0, 0], dataset_x[dataset_y == 0, 1], color='blue')
plt.scatter(dataset_x[dataset_y == 1, 0], dataset_x[dataset_y == 1, 1], color='green')
plt.scatter(dataset_x[dataset_y == 2, 0], dataset_x[dataset_y == 2, 1], color='red')

x = np.linspace(-5, 5, 100)
for i in range(3):
    y = (-lr.coef_[i, 0] * x - lr.intercept_[i]) / lr.coef_[i, 1]
    plt.plot(x, y, color=['blue', 'green', 'red'][i])

```



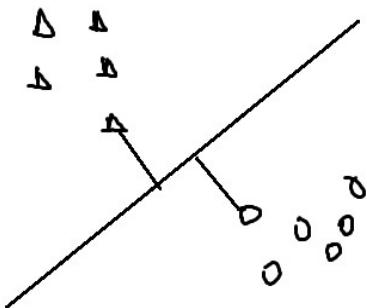
Buradan elde edilen score 1'dir. Bu mükemmel bir ayrıştırma anlamına gelmektedir. Şimdi doğrusal olarak ayrılabilir olmayan bir örnek verelim:



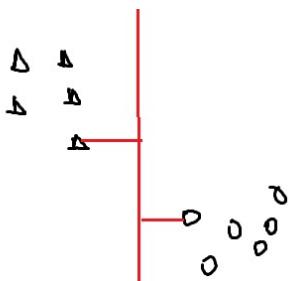
Buradaki skor 0.78 olarak elde edilmiştir.

DESTEK VEKTÖR MAKİNELERİ (SUPPORT VECTOR MACHINES)

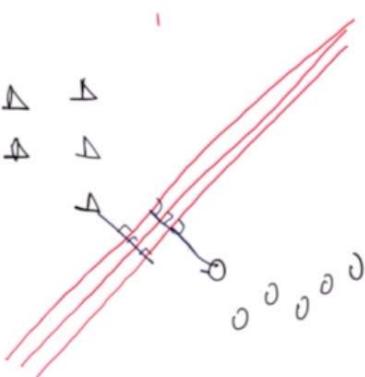
Destek vektör makineleri de eğitimli (supervised) sınıflandırma (classification) yöntemlerinden biridir. Destek vektör makinelerinde de aslında sınıfları ayırmak için doğrular kullanılmaktadır. Ancak bu doğrular en yakın noktaları ayırmaya amacıyla oluşturulmaktadır. Bu en yakın noktalara destek vektörleri (support vectors) denilmektedir. Örneğin:



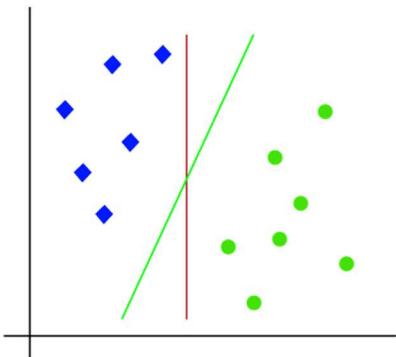
Burada iki sınıfı ayırmak için öyle bir doğru oluşturulur ki bu doğru şu özelliğe sahiptir: "Bu doğruya en yakın iki sınıfındaki noktalar ele alındığında bu noktalar ile doğru arasındaki toplam uzaklık diğer alternatif doğrulara göre daha fazla olmalıdır". Buradaki noktanın doğruya uzaklığında dikme uzaklığı kullanılmaktadır. Örneğin aşağıdaki diğer alternatif doğruya göz önüne alalım:



Burada alternatif bir bir doğru görülmektedir. Peki yukarıdaki hangi iki doğru yukarıda tanımladığımız tanıma daha yakındır? İşte hedefimiz doğruya iki sınıfındaki en yakın noktaların uzakları toplamını maksimum yapan doğruya elde etmektir. Doğruya en yakın noktalarla doğru arasındaki toplam uzaklığı marjin denir. Peki marjini aynı olan sonsuz sayıda paralel doğru elde edilebileceğine göre bunlardan hangisi seçilecektir. İşte en yakın iki noktaya uzaklıklar eşit olan marjini en yüksek doğru seçilmektedir.



. Aşağıda iki sınıflı bir model için alternatif iki destek doğrusu görüyorsunuz:



Burada gözle de görüldüğü gibi yeşil doğru kırmızı doğrudan daha iyidir.

Destek vektör makinelerinde uygun doğrunun elde edilmesi süreci teorik bakımdan biraz karışıktır. Biz burada bu doğrunun nasıl elde edileceği üzerinde durmayacağız. Bunun için başka kaynaklara başvurulabilir.

Destek vektör makineleri Scikit-Learn içerisindeki SVC sınıfıyla temsil edilmektedir. Sınıfın `__init__` metodunun parametrik yapısı şöyledir:

```
class sklearn.svm.SVC(C=1.0, kernel='rbf', degree=3, gamma='scale', coef0=0.0, shrinking=True,
probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1,
decision_function_shape='ovr', break_ties=False, random_state=None)
```

Aslında destek vektör makineleri yalnızca sınıflandırma amacıyla değil aynı zamanda regresyon amacıyla kullanılmaktadır. sklearn.svm modülündeki bu SVC (Support Vector Classification) sınıflandırma amacıyla SVR (Support Vector Regression) sınıfı da regresyon amacıyla kullanılmaktadır.

Şimdi destek vektör makineleri ile sınıflandırmaya bir örnek verelim. Bu sınıflandırma örneğinde scikit-learn içerisinde bulunan kanser tarama verilerinin gerçekten kanser olup olmadığını belirlemeye çalışacağız. Scikit-Learn'deki bu veri kümesinde kişilerden meme kanseri riski için önemli olabilecek bazı biyomedikal veriler elde edilmiştir. Amaç bu verilere dayanarak kişideki lezyonun iyi huylu mu (benign) yoksa kötü huylu mu (malign) olduğuna karar vermektr. Bu sınıflandırma probleminin destek vektör makineleri ile çözümü şöyledir:

```
from sklearn.datasets import load_breast_cancer

bc = load_breast_cancer()
print('Sütun isimleri: {}'.format(bc.feature_names))
print('Sınıf Isimleri: {}'.format(bc.target_names))

dataset_x = bc.data
dataset_y = bc.target

from sklearn.model_selection import train_test_split

training_dataset_x, test_dataset_x, training_dataset_y, test_dataset_y =
train_test_split(dataset_x, dataset_y, test_size=0.3)

from sklearn.svm import SVC

svc = SVC(kernel='linear')
svc.fit(dataset_x, dataset_y)

score = svc.score(test_dataset_x, test_dataset_y)
print(score)

result = svc.predict(test_dataset_x)
ratio = sum(result == test_dataset_y) / len(test_dataset_y)
print(ratio)
```

Burada SVC nesnesi yaratılırken kernel olarak 'linear' verildiğine dikkat ediniz. Bu kernel doğrusal olarak ayrılabilir veri kümesi için en iyi yöntemdir. Anımsanacağı gibi doğrusal olarak ayrılabilir veriler için lojistik regresyon da kullanılabilmektedir. Fakat eğer veriler doğrusal olarak ayırtılabilir değilse destek vektör makinelerinde ismine "kernel trick" denilen yöntemle boyut yükseltilmesi yapılarak sorun çözülmektedir. Halbuki lojistik regresyonda böyle bir "kernel trick" durumu yoktur. (Kernel trick aslında doğrusal olarak ayırtılamayan veri kümelerini boyut yükselterek doğrusal olarak ayırtılabilir hale getiren bir yöntemdir.)

SVC sınıfının predict metodu ile kestirim yapılabilir. Yukarıdaki örnekte test_dataset_x verileri üzerinde predict ile kestirimde bulunulmuştur. score fonksiyonu yine kestirim yapıp başarı yüzdesini bize verir. Tabii biz bu başarı yüzdesini yukarıdaki örnekte olduğu gibi manuel bir biçimde elde edebiliriz. Söz konusu kanser verileri için başarı oranı %96.4 bulunmuştur. Şimdi aynı örneği Lojistik regresyon ile yapalım:

```
from sklearn.datasets import load_breast_cancer

bc = load_breast_cancer()
print('Sütun isimleri: {}'.format(bc.feature_names))
print('Sınıf İsimleri: {}'.format(bc.target_names))

dataset_x = bc.data
dataset_y = bc.target

from sklearn.model_selection import train_test_split

training_dataset_x, test_dataset_x, training_dataset_y, test_dataset_y =
train_test_split(dataset_x, dataset_y, test_size=0.3)

from sklearn.linear_model import LogisticRegression

lr = LogisticRegression()
lr.fit(dataset_x, dataset_y)

score = lr.score(test_dataset_x, test_dataset_y)
print(score)

result = lr.predict(test_dataset_x)
ratio = sum(result == test_dataset_y) / len(test_dataset_y)
print(ratio)
```

Elde edilen sonuçlar benzerdir.

Şimdi de kendimiz make_blobs fonksiyonuyla rastgele veri üreterek destek vektör makineleriyle elde edilen doğrula lojistik regresyondan elde edilen doğrulu grafiksel biçimde göstermeye çalışalım. Aşağıda böyle bir kod görüyorsunuz:

```
import numpy as np

from sklearn.datasets.samples_generator import make_blobs

dataset_x, dataset_y = make_blobs(n_samples=100, n_features=2, centers=2, cluster_std=0.8)

from sklearn.svm import SVC

svc = SVC(kernel='linear')
svc.fit(dataset_x, dataset_y)

from sklearn.linear_model import LogisticRegression

lr = LogisticRegression()
lr.fit(dataset_x, dataset_y)

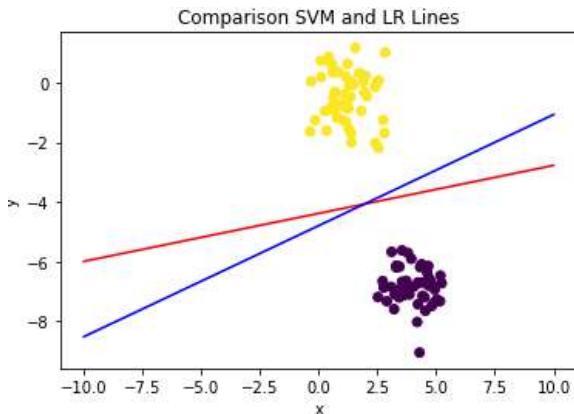
import matplotlib.pyplot as plt
```

```

plt.title('Comparison SVM and LR Lines')
plt.xlabel('x')
plt.ylabel('y')

plt.scatter(dataset_x[:, 0], dataset_x[:, 1], c=dataset_y)
x = np.linspace(-10, 10, 100)
y = (-svc.coef_[0][0] * x - svc.intercept_[0]) / svc.coef_[0][1]
plt.plot(x, y, c='red')
y = (-lr.coef_[0][0] * x - lr.intercept_[0]) / lr.coef_[0][1]
plt.plot(x, y, c='blue')

```



Burada kırmızı destek vektör makineleri ile mavi ise lojistik regresyon ile elde edilmiş olan doğrulardır.

Pekiyi eğer veri kümesi doğrusal olarak ayırtılabilir bir küme ise lojistik regresyon mu yoksa destek vektör makineleri mi daha iyi sonuç bermektedir? Bu karşılaştırma sonraki başlık altında değerlendirilecektir.

Pekiyi veri kümesi doğrusal olarak ayırtılabilir değilse ne yapmalıyız? İşte lojistik regresyonda yapılacak bir şey yoktur. Bu durumda yapay sinir ağları ya da destek vektör makineleri kullanılabilir. Yukarıda da belirtildiği gibi destek vektör makinelerinde kernel değiştirilerek boyut yükselmesi ile doğrusal olarak ayırtılabilir olmayan sınıflandırma problemleri de çözülebilmiştir. Bunun için destek vektör makineelerinde birkaç hazır kernel bulunmaktadır. Şimdi doğrusal olarak ayırtılabilir olmayan dairesel bir veri kümesi için "radial kernel" kullanımını örnek olarak verelim. Bunun için önce dairesel iki sınıfı rasgele noktalar oluşturalım. Bu işlem `sklearn.datasets` modülü içerisindeki `make_circles` fonksiyonuyla yapılabilmektedir. (Animasyonu gibi `make_blobs` fonksiyonu doğrusal olarak ayırtılabilir tarzda rastgele veriler üretiyor.)

```

from sklearn.datasets import make_circles

dataset_x, dataset_y = make_circles(n_samples=100)

from sklearn.model_selection import train_test_split

training_dataset_x, test_dataset_x, training_dataset_y, test_dataset_y =
train_test_split(dataset_x, dataset_y, test_size=0.3)

import matplotlib.pyplot as plt

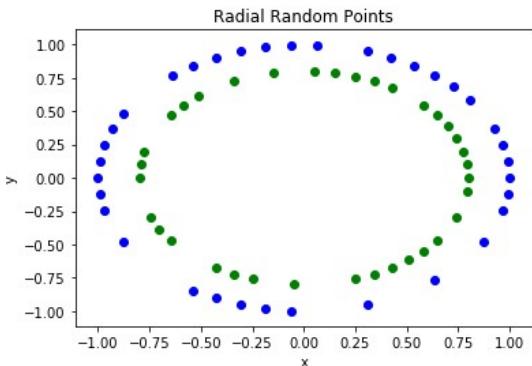
plt.scatter(training_dataset_x[training_dataset_y == 0, 0],
            training_dataset_x[training_dataset_y == 0, 1], color='blue')

plt.title('Radial Random Points')
plt.xlabel('x')
plt.ylabel('y')
plt.scatter(training_dataset_x[training_dataset_y == 1, 0],
            training_dataset_x[training_dataset_y == 1, 1], color='green')

plt.pause(1)

```

Bu işlemler sonucunda aşağıdaki gibi rastgele 2 sınıfı dairesel noktalar elde ettik:



Göründüğü gibi bu iki sınıfı dairesel noktalar bir doğru ile ayırtılabilirler. Olsa olsa bu noktalar dairesel bir eğri ile ayırtılabilirler. İşte SVC sınıfındaki "radiak kernel (rbf)" bunu sağlamaktadır:

```
from sklearn.svm import SVC

svc = SVC(kernel='rbf')
svc.fit(training_dataset_x, training_dataset_y)

result = svc.predict(test_dataset_x)
ratio = sum(result == test_dataset_y) / len(test_dataset_y)
print('Başarı oranı: {}'.format(ratio))
print('Score: {}'.format(svc.score(training_dataset_x, training_dataset_y)))
```

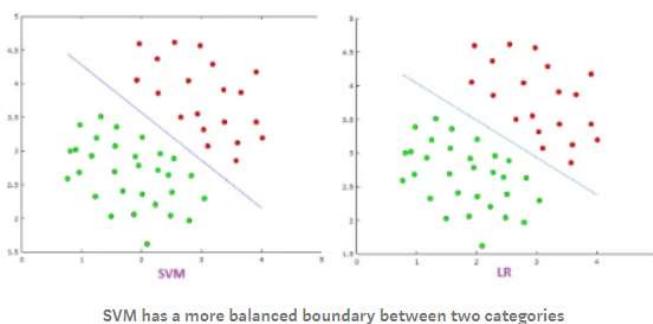
Buradan şu sonuç elde edilmiştir:

```
Başarı oranı: 1.0
Score: 1.0
```

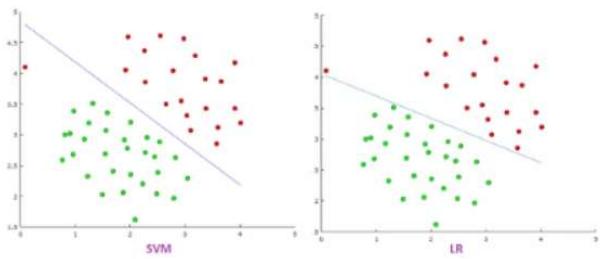
SVC sınıfındaki "poly" kernel ise sınıfları bir doğru ile ya da dairesel olarak değil polinomsal bir fonksiyonla ayırtırmaya çalışmaktadır.

Lojistik Regresyon, Destek Vektör Makineleri, Kümeleme ve Yapay Sinir Ağları Yöntemlerinin Karşılaştırması

Eğer veri kümesi doğrusal olarak ayırtılabilir değilse zaten lojistik regresyon kullanılamaz. Bu durumda destek vektör makineleri (kernel değiştirerek), kümeleme yöntemleri ya da yapay sinir ağları kullanılmalıdır. Eğer veri kümesi doğrusal olarak ayırtılabilir ise lojistik regresyon, destek vektör makineleri ve yapay sinir ağları yöntemlerinin her biri kullanılabilir. Genel olarak destek vektör makinelerinin daha adil (ortalayacak biçimde) bir sınır tespiti yaptığı söylenebilir.



Öte yandan lojistik regresyon üç değerler olması durumunda bu üç değerleri dikkate alacak biçimde daha iyi bir sınıflandırma yapabilmektedir:



LR's boundary change by including one new example. In reality this may not be a good idea so being not sensitive like SVM tends to be better.

Büyük veriler için lojistik regresyon destek vektör makinelerinden daha fazla işlem gerektirmektedir. Dolayısıyla daha yavaş olma eğilimindedir.

Yapay sinir ağları her türlü verilerde çalışabilen genel bir yöntem olmasına karşın eğitim konusunda sorunları olabilmektedir. Yapay sinir ağlarından performans elde edebilmek için belli miktarda veri ile eğitmek gerekmektedir. Halbuki lojistik regresyon ve destek vektör makinelerinde makul çözüm az sayıda veri ile elde edilebilmektedir.

Kümeleme yöntemlerinin denetimli (supervised) değil denetimsiz (unsupervised) yöntemler olduğunu anımsayınız. Dolayısıyla elimizde yalnızca dataset_x varsa fakat dataset_y yoksa mecburen kümeleme yöntemlerini kullanız.

DOĞRUSAL KARAR MODELLERİNİN ÇÖZÜMÜ

Doğrusal programlama "yöneylem araştırması (operational research)" denilen alanın en önemli konularından biridir. Belli doğrusal kısıtlar altında maksimizasyon ve minimizasyon işlemlerine yönelikdir. Doğrusal programlama 2. Dünya Savaşı sürecinde geliştirilmiştir. Makine öğrenmesi işlemlerinde belli adımlarda doğrusal programlama ile optimizasyonların yapılması gerekmektedir. Yani bu konu doğrudan makine öğrenmesi içeresine girmese de bir biçimde makine öğrenmesi konusuyla ilişkilidir. Doğrusal programlamadaki doğrusallık kullanılan değişkenlerin birinci derece olması ile ilgilidir. Eğer kısıtları ve amaç fonksiyonunu oluşturan değişkenler birinci dereceden değilse buna da "doğrusal olmayan programlama (nonlinear programming)" denilmektedir.

Doğrusal karar modellerinde kısıtlar (constraints) ve amaç fonksiyonu (objective function) vardır. Doğrusal programlama faaliyetinde kısıtları sağlayan amaç fonksiyonunun en büyük ya da en küçük değeri elde edilmeye çalışılmaktadır. Kısıtlar bir çözüm alanı oluşturur. Bu çözüm alanı içerisinde (kısıtlar sağlanacak biçimde) amaç fonksiyonu en büyütlenmeye ya da en küçütlenmeye çalışılmaktadır. Doğrusal karar modellerinin çözüm alanı konveks bir kümedir. Bu konveks kümede uç noktalar (extreme points) bulunur. Eniyi çözüm de bu uç noktalardan birindedir. Doğrusal karar modellerin çözümü için yalnızca uç noktaları dolaşan algoritmalar önerilmiştir. En etkin algoritmaların biri "Simplex" denilen algoritmadır. Simplex algoritmasının revize edilmiş biçimine de "Revised Simplex" denilmektedir. Tabii biz burada Simplex algoritmasının kendisini görmeyeceğiz. Yalnızca doğrusal karar modelinin oluşturulmasını ve çözümünde kullanılan hazır sınıfları ve fonksiyonları göreceğiz.

Doğrusal karar modelleri matematiksel olarak iki biçimde ifade edilebilmektedir. Bunlardan birine kanonik biçim diğerine standart biçim denilmektedir. Kanonik biçimde tüm kısıtlar \leq biçimine ya da \geq biçimine getirilir. Standart biçimde ise kısıtlar $=$ biçiminde bulunmaktadır. Kanonik biçimin genel hali şöyledir:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \dots + a_{1n}x_n &\leq b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \dots + a_{2n}x_n &\leq b_2 \\ \vdots & \\ a_{m1}x_1 + a_{m2}x_2 + a_{m3}x_3 + \dots + a_{mn}x_n &\leq b_m \end{aligned}$$

$$Z_{\text{Max}} = c_1x_1 + c_2x_2 + \dots + c_nx_n$$

Aslında bu modeli matrisel biçimde aşağıdaki gibi de ifade edebiliriz:

$$AX \leq b$$

$$x \geq 0$$

$$Z_{\max} = CX$$

Standart form ise matrisel biçimde şöyle gösterilebilir:

$$AX = b$$

$$x \geq 0$$

$$Z_{\max} = CX$$

Doğrusal karar problemleri genellikle bize metinsel olarak gelir. Biz bu problemin matematiksel modelini yukarıda belirtildiği gibi kanonik ya da standart biçimde ifade ederiz. Sonra da bunu Python'ın hazır modülleriyle çözeriz. Örnek bir doğrusal karar modeli problemi şöyle olabilir:

Bir çiftçinin buğday, mısır ve arpa ürünlerinin ekimi için 300 hektarlık arazisi vardır. Çiftçi hektar başına buğdaydan 150 TL, mısırda 220 TL, arpadan da 180 TL kar beklemektedir. İşgücü yüzünden çiftçi buğday için 150 hektardan ve arpa için 120 hektardan daha fazla yer ayırmamalıdır. Verimlilik yönünden ise en az 80 hektar buğday için yer ayırmalı ve mısır ekimi için de toplam arazinin %30'undan daha fazla yer ayırmamalıdır. Çiftçi karını en büyüklemek istemektedir. Çiftçinin hangi ürün için ne kadar alan ayırması gereklidir?

Bu problemde x_1 buğday için ayrılan alan, x_2 mısır için ayrılan alan ve x_3 de arpa için ayrılan alan olmak üzere kısıtlar ve amaç fonksiyonu şöyle modellenebilir:

$$x_1 + x_2 + x_3 = 300$$

$$x_1 \leq 150$$

$$x_3 \leq 120$$

$$x_1 \geq 80$$

$$x_2 \leq 90$$

$$x_1, x_2, x_3 \geq 0$$

$$Z_{\max} = 150 x_1 + 220 x_2 + 180 x_3$$

Göründüğü gibi burada problem matematiksel terimlerle ifade edilmiştir. Artık biz bu modelden hareketle Python'daki hazır modülleri kullanarak problemi çözebilir. Diğer bir problem de şöyle olabilir:

Bir boyacı fabrikası hem iç hem dış boyacı üretiyor. Boyacı üretiminde A ve B olmak üzere iki tip hammadde kullanılıyor. Bir günde A hammaddesinden en çok 6 ton, B hammaddesinden en çok 8 ton kullanılabilir. Günlük hammadde ihtiyacı iç ve dış boyacı için ton olarak aşağıdaki tabloda verilmiştir.

Hammadde	Gereken Hammadde		Mevcut Miktar (ton)
	Miktarı (ton)		
	Dış Boya	İç Boya	
A	1	2	6
B	2	1	8

Bir pazar araştırması iç boyanın günlük isteminin, dış boyacı istemini 1 tondan fazla aşmadığını ve iç boyanın en büyük isteminin 2 ton olduğunu gösteriyor. İç boyanın toplam satış fiyatı bir tonda 2 br, dış boyanın toplam satış fiyatı bir tonda 3 br dir. Şirket toplam geliri en büyükleyecek biçimde kaç ton iç boyacı kaç ton dış boyacı üretimi yapması gerektiğini belirlemek istemektedir.

Bu problemde iki değişken vardır: İç boyacı ve dış boyacı miktarı. Bunları x_1 ve x_2 biçiminde temsil edebiliriz. Amaç fonksiyonu bir maksimizasyon biçimindedir. Kısıtlar şu biçimde ifade edilebilir:

$$\begin{aligned}
 X_1 + 2X_2 &\leq 6 \quad (\text{A hammadde kısıtı}) \\
 2X_1 + X_2 &\leq 8 \quad (\text{B hammadde kısıtı}) \\
 X_2 - X_1 &\leq 1 \quad (\text{İç boyanın günlük istemi dış boyayı 1 tondan fazla aşmayacak}) \\
 X_2 &\leq 2 \quad (\text{İç boyanın en büyük istemi 2 ton}) \\
 X_1, X_2 &\geq 0
 \end{aligned}$$

$$\max f(\mathbf{X}) = \max Z = 3X_1 + 2X_2$$

Diger bir örnek de şöyle olabilir:

Bir kişi sadece et, süt ve yumurta yiyecek diyet yapmaktadır. Bu kişinin günde en az 15 mg A vitamini, 30 mg C vitamini ve 10 mg D vitamini alması gerekmektedir. Buna karşılık besinlerle aldığı kolesterol 80 br/gün'ü geçmemelidir. 1 lt sütte 1 mg A, 100 mg C 10 mg D ve 70 br kolesterol vardır ve sütün litresi 800 TL dir. 1 kg ette 1 mg A, 10 mg C, 100 mg D vitamini ve 50 br kolesterol vardır. Etin kilosu 3700 TL dir. Yumurtanın düznesinde 10 mg A, 10 mg C ve 10 mg D vitamini ile 120 br kolesterol bulunmakta olup, yumurtanın düznesi 275 TL dir. Kişiin istediği, bu diyeti en ucuz yolla gerçekleştirmektir. Buna göre problemin DP modelini oluşturunuz.

Problemin matematiksel modeli şöyle oluşturulabilir:

- x1: Bir günde tüketilecek süt miktarı (litre)
- x2: Bir günde tüketilecek et miktarı (kg)
- x3: Bir günde tüketilecek yumurta miktarı (tane)

$$\min Z = 800x_1 + 3700x_2 + 275x_3$$

Kısıtlar şunlardır:

$$\begin{aligned}
 x_1 + x_2 + 10x_3 &\geq 15 && (\text{A vitamini kısıtı}) \\
 100x_1 + 10x_2 + 10x_3 &\geq 30 && (\text{C vitamini, kısıtı}) \\
 10x_1 + 100x_2 + 10x_3 &\geq 10 && (\text{D vitamini kısıtı}) \\
 70x_1 + 50x_2 + 120x_3 &\leq 80 \\
 x_1, x_2, x_3 &\geq 0
 \end{aligned}$$

Diger örnek söyle olabilir:

Bir oyuncak imalatçısı model otomobil ve uçak üretimi yapmayı planlamaktadır. Şirket bu iki imalatını iki ayrı işlemin yapıldığı I ve II nolu atölyelerinde gerçekleştirmektedir. Çizelgede bir adet model otomobil ile model uçak imali için atölye işlem süreleri ve atölye kapasiteleri verilmiştir. Bir model otomobil satışından 45 TL, bir model uçak satışından ise 55 TL kar elde edilecektir. Maksimum kar için her bir üründen ne kadar imal edilmeli?

Atölyeler	Mallar		Kapasite (saat)
	Otomobil	Uçak	
	İşlem zamanı (saat/ad.)		
I	6	4	120
II	3	10	180

Örnek https://kemaisonmez.weebly.com/uploads/2/4/7/1/24712761/sm_ders_3_dogrusal_programlama_genel.pdf URL'sinden alınmıştır.

Problemin değişkenleri söyle ifade edilebilir:

- x1: Model otomobil miktarı (adet)
- x2: Model uçak miktarı (adet olarak)

Problemin amaç fonksiyonu söyledir:

$$\text{Max } Z = 45x_1 + 55x_2$$

Problemin kısıtları şöyledir:

$$\begin{aligned} 6x_1 + 4x_2 &\leq 120 \\ 3x_1 + 10x_2 &\leq 180 \\ x_1, x_2 &\geq 0 \end{aligned}$$

Doğrusal Programlama Modelinin SciPy linprog İle Çözülmesi

Doğrusal karar modellerinin çözümü için `scipy.optimize` modülü içerisindeki `linprog` sınıfı kullanılabilir. Bu sınıfın `__init__` metodunun parametrik yapısı şöyledir:

```
scipy.optimize.linprog(c, A_ub=None, b_ub=None, A_eq=None, b_eq=None, bounds=None, method='simplex', callback=None, options=None)
```

Bu fonksiyonun kullanılabilmesi için amaç fonksiyonun minimizasyon biçiminde olması gereklidir. Eğer amaç fonksiyon makismizasyonsa bizim onu negatif değerlerle çarpıp minimizasyon haline getirmemiz gereklidir. Çünkü amaç fonksiyonunun en büyütlenmesi onun negatifinin en küçütlenmesi ile aynı anlamladır. Fonksiyondaki `A_ub` (upper bound) parametresi modeldeki " \leq " kısıtlarının A katsayı matrisini belirtir. Eğer kısıtlar içerisinde " \geq " olanlar varsa eşitsizliğin her iki tarafı negatif ile çarpılıp bu eşitsizlik " \leq " haline getirilmelidir. `A_eq` parametresi " $=$ " olan kısıtların A katsayı matrisini belirtmektedir. Fonksiyondaki `bounds` parametresi kısıtlardaki değişkenlerin alabileceği değer aralığını belirlemek için kullanılır. Bu parametre için argüman girilmezse default durum " ≥ 0 " biçimindedir. Eğer değişkenler özel bir kısıt içeriysorsa burada tüm değişkenler sırasıyla bir dolaşılabilir nesne içerisinde demetler biçiminde oluşturulmalıdır. Örneğin x_1 koşulunun " ≥ 10 " olduğunu, x_2 koşulunun da " ≥ -5 " ve " ≤ 20 " olduğunu varsayıyalım. Burada `bounds` parametresi şöyle düzenlenmelidir.

```
bounds = [(10, None), (-5, 20)]
```

Buradaki `None` değeri minimal için $-\infty$, Maksimum için $+\infty$ anlamına gelmektedir.

`linprog` fonksiyonunun geri dönüş değeri bir sınıf türündendir. Bu sınıfın `__str__` ve `__repr__` metotları karar modelini sonucunu bize yazı olarak verir. Fakat bu sonuçları biz de şu örnek özniteliklerden ayrı ayrı nümerik biçiminde alabiliriz:

fun: Amaç fonksiyonun değeri (Eğer amaç fonksiyonu maksimizasyonsa bu değerin negatif dikkate alınmalıdır.)
x: Karar değişkenlerinin optimum değerleri
success: Optimal çözüm var mı, yok mu? bool bir değer
nit: Optimal çözüm için gereken iterasyon sayısı

Şimdi bir örnek yapalım:

$$Z_{\max} = 3x + 2y$$

Kıstlar:

$$\begin{aligned} 2x + y &\leq 18 \\ 2x + 3y &\leq 42 \\ 3x + y &\leq 24 \\ x, y &\geq 0 \end{aligned}$$

Çözüm şöyle yapılabilir:

```
import numpy as np
from scipy.optimize import linprog

c = np.array([-3, -2], dtype=np.float32)
```

```

aub = np.array([[2, 1], [2, 3], [3, 1]], dtype=np.float32)
bub = np.array([18, 42, 24])
lp = linprog(c, A_ub=aub, b_ub=bub)

print(lp)
print('-----')
print('Variables: {}'.format(lp.x))
print('Max Value: {}'.format(-lp.fun))

```

Sonuç şöyle lede edilmiştir:

```

con: array([], dtype=float64)
fun: -32.9999980043574
message: 'Optimization terminated successfully.'
nit: 4
slack: array([9.60302629e-08, 3.18105720e-07, 3.0000009e+00])
status: 0
success: True
x: array([ 3.0000001, 11.9999989])
-----
Variables: [ 3.0000001 11.9999989]
Max Value: 32.9999980043574

```

Şimdi daha önce verdiğimiz atölye problemini çözelim:

```

import numpy as np
from scipy.optimize import linprog

c = np.array([-3, -2], dtype=np.float32)
aub = np.array([[2, 1], [2, 3], [3, 1]], dtype=np.float32)
bub = np.array([18, 42, 24])
lp = linprog(c, A_ub=aub, b_ub=bub)

print('variables: {}'.format(lp.x))
print('Max Value: {}'.format(-lp.fun))

con: array([], dtype=float64)
fun: -1274.999989391895
message: 'Optimization terminated successfully.'
nit: 4
slack: array([9.93566687e-08, 1.50681728e-07])
status: 0
success: True
x: array([ 9.9999999, 14.9999999])
-----
Variables: [ 9.9999999 14.9999999]
Max Value: 1274.999989391895

```

Doğrusal Programlama Modeli İçin Pulp Kütüphanesinin Kullanımı

Pulp kütüphanesi linprog fonksiyonundan daha detaylıdır. Ancak gösterim daha doğal bir formattadır. Ancak bu kütüphane default olarak Anaconda içerisinde yoktur. Kütüphane aşağıdakilerden biriyle kurulabilir:

```

pip install pulp
python -m pip install pulp

```

Anaconda için conda programıyla kurulum da şöyle yapılabilir:

```

conda install -c conda-forge pulp

```

Kurulum sırasında hangi python sürümünün pip ya da python programının çalıştığını dikkat ediniz.

Pulp kütüphanesini dokümantasyonuna aşağıdaki adresten erişebilirsiniz:

<https://coin-or.github.io/pulp/>

Pulp kütüphanesinin kullanımı şöyledir:

1) Önce pulp modülündeki LpProblem sınıfı türünden bir nesne yaratılır. Bu nesne yaratılırken problemin ismi ve maksimizasyon mu, minimizasyon mu olduğu belirtilir.

```
pulp.LpProblem(name='NoName', sense=1)
```

Örneğin:

```
import pulp  
lp = pulp.LpProblem('My Problem', pulp.LpMaximize)
```

2) Bundan sonra değişkenleri oluşturmak gereklidir. Değişkenler pulp.LpVariable isimli bir sınıfta temsil edilmişlerdir. Değişkenler oluşturulurken onlara isimler ve sınır değerler verilebilir. Burada lowBound ve upBound parametreleri default olarak None değerini almıştır. Bu None değeri lowBound için $-\infty$, upBound için $+\infty$ biçimindedir.

```
pulp.LpVariable(name, lowBound=None, upBound=None, cat='Continuous', e=None)
```

Örneğin:

```
import pulp  
lp = pulp.LpProblem('My Problem', pulp.LpMaximize)  
  
x = pulp.LpVariable("x", lowBound=0)  
y = pulp.LpVariable("y", lowBound=0)
```

3) Şimdi sıra amaç fonksiyonunu ve kısıtları oluşturmaya gelmiştir. Amaç fonksiyonu LpProblem sınıfının `+=` operatör metoduyla oluşturulmaktadır. Örneğin:

```
lp += 3 * x + 2 * y
```

Kısıtlar da tamamen aynı yöntemle oluşturulmaktadır. Ancak kısıtlarda "`<=`", "`>=`" ya da "`==`" operatörleri de kullanılmalıdır.

```
lp += 3 * x + 2 * y  
lp += 2 * x + y <= 18  
lp += 2 * x + 3 * y <= 42  
lp += 3 * x + y <= 24
```

Sınıfın `__str__` ve `__repr__` metodları modelin matematiksel temsilini yazı olarak bize vermektedir.

4) Bundan sonra sıra modeli çözmeye gelmiştir. Çözme işlemi LpProblem sınıfının `solve` isimli metoduyla yapılmaktadır.

```
solve(solver=None, **kwargs)
```

Programcı isterse çözüm algoritmasını ya da çözüm yazılımını değiştirebilmek için solver parametresini kullanabilir.

LpProblem sınıfının `objective`, `constraints`, ve `status` örnek öznitelikleri bize modele ilişkin bilgileri vermektedir.

5) Çözüm sonucunda değişkenlerin optimal değerlerini alabilmek için `pulp.value` fonksiyonu kullanılmaktadır. Örneğin:

```

print('x = {}'.format(pulp.value(x)))
print('y = {}'.format(pulp.value(y)))

```

Amaç fonksiyonunun değeri de yine pulp.value metoduyla constraints özniteliği verilerek elde edilebilir. Örneğin:

```
print('Objective: {}'.format(pulp.value(lp.objective)))
```

Çözümün tüm kodları şöyledir:

```

import pulp

lp = pulp.LpProblem('MyProblem', pulp.LpMaximize)
x = pulp.LpVariable('x', lowBound=0)
y = pulp.LpVariable('y', lowBound=0)

lp += 3 * x + 2 * y

lp += 2 * x + y <= 18
lp += 2 * x + 3 * y <= 42
lp += 3 * x + y <= 24

print(lp)

lp.solve()
print('x = {}'.format(pulp.value(x)))
print('y = {}'.format(pulp.value(y)))
print('Objective: {}'.format(pulp.value(lp.objective)))

```

LpProblem sınıfının writeLP metodu modeli bir text dosyanın içerisinde yazar. Ancak maalesef pulp modülünde modülü okuyan bir fonksiyon yoktur.

Şimdi son olarak daha çok değişkenli bir doğrusal programlama modelini metinden hareketle çözmeye çalışalım:

Bir şehirde her biri 1200 kişilik öğrenci kapasitesi olan üç ayrı lise bulunmaktadır. Şehir yönetimi şehri Kuzey, Güney, Doğu, Batı ve Merkez olarak beş ayrı bölgeye ayırmıştır. Bazı öğrenciler bölgelerinin dışındaki okullara gitmek durumundadır. Bu durumda okul yönetimleri öğrencilerin alacağı toplam mesafeyi en azı indirerek istemektedir. Gerekli olan doğrusal modeli kurunuz. Okullar bve mesafeleri aşağıdaki tabloda verilmiştir:

Bölgeler	Merkez Lisesi	Batı Lisesi	Güney Lisesi	Öğrenci Sayısı
Kuzey	8	11	14	700
Güney	12	9	0	300
Doğu	9	16	10	900
Batı	8	0	9	600
Merkez	0	8	12	500

Bu problemde bölgelerden liselere kaç kişinin gideceği karar değişkenlerini oluşturmaktadır. X_{ij} karar değişkeni j 'inci bölgeden i 'inci okula kaç kişinin gideceğini belirtmektedir. Örneğin X_{13} karar değişkeni Doğu bölgesinden Merkez liseye kaç kişinin gideceğini belirtmektedir. Model şöyle kurulabilir:

$$\text{Min } Z = 8X_{11} + 12X_{12} + 9X_{13} + 8X_{14} + 0 * X_{15} + 11X_{21} + 9X_{22} + 16X_{23} + 0X_{24} + 8X_{25} + 14X_{31} + 0X_{32} + 10X_{33} + 9X_{34} + 12X_{35}$$

Kısıtlar:

$$X_{11} + X_{21} + X_{31} = 700 \quad (\text{Kuzey bölgesindeki toplam öğrenci kısıtı})$$

$$X_{12} + X_{22} + X_{32} = 300 \quad (\text{Güney bölgesindeki toplam öğrenci kısıtı})$$

$$X_{13} + X_{23} + X_{33} = 900 \quad (\text{Doğu bölgesindeki toplam öğrenci kısıtı})$$

$X_{14} + X_{24} + X_{34} = 600$ (Batı bölgesindeki toplam öğrenci kısıtı)
 $X_{15} + X_{25} + X_{35} = 500$ (Merkez bölgesindeki toplam öğrenci kısıtı)

$X_{11} + X_{12} + X_{13} + X_{14} + X_{15} \leq 1200$ (Merkez lisesinin kapasite kısıtı)
 $X_{21} + X_{22} + X_{23} + X_{24} + X_{25} \leq 1200$ (Batı lisesinin kapasite kısıtı)
 $X_{31} + X_{32} + X_{33} + X_{34} + X_{35} \leq 1200$ (Güney lisesinin kapasite kısıtı)

$X_{11}, X_{12}, X_{13}, X_{14}, X_{15}, X_{21}, X_{22}, X_{23}, X_{24}, X_{25}, X_{31}, X_{32}, X_{33}, X_{34}, X_{35} \geq 0$

$x_{ik} \rightarrow i^{\text{nci}} \text{ okuldan } k^{\text{inci}} \text{ bölgeye gidecek öğrenci sayısı}$

Problemi önce linprog ile çözelim:

```
import numpy as np
from scipy.optimize import linprog

c = np.array([8, 12, 9, 8, 0, 11, 9, 16, 0, 8, 14, 0, 10, 9, 12])
A_ub = np.array([[1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], 
                 [0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0], 
                 [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1]])
b_ub = np.array([1200, 1200, 1200])

A_eq = np.array([[1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0], 
                 [0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0], 
                 [0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0], 
                 [0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1], 
                 [0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0]])
b_eq = np.array([700, 300, 900, 600, 500])

result = linprog(c=c, A_ub=A_ub, b_ub=b_ub, A_eq=A_eq, b_eq=b_eq)
print(result)
```

Elde edilen sonuç şöyledir:

```
fun: 14600.0
message: 'Optimization terminated successfully.'
    nit: 14
    slack: array([ 0., 600., 0.])
status: 0
success: True
    x: array([700., 0., 0., 0., 500., 0., 0., 0., 600., 0., 0.,
       300., 900., 0., 0.])
```

Şimdi aynı problemi pulp ile çözelim:

```
import pulp

lp = pulp.LpProblem('District_School_Problem', pulp.LpMinimize)

x11 = pulp.LpVariable("x11", lowBound=0)
x12 = pulp.LpVariable("x12", lowBound=0)
x13 = pulp.LpVariable("x13", lowBound=0)
x14 = pulp.LpVariable("x14", lowBound=0)
x15 = pulp.LpVariable("x15", lowBound=0)

x21 = pulp.LpVariable("x21", lowBound=0)
x22 = pulp.LpVariable("x22", lowBound=0)
x23 = pulp.LpVariable("x23", lowBound=0)
x24 = pulp.LpVariable("x24", lowBound=0)
x25 = pulp.LpVariable("x25", lowBound=0)
```

```

x31 = pulp.LpVariable("x31", lowBound=0)
x32 = pulp.LpVariable("x32", lowBound=0)
x33 = pulp.LpVariable("x33", lowBound=0)
x34 = pulp.LpVariable("x34", lowBound=0)
x35 = pulp.LpVariable("x35", lowBound=0)

variables = [x11, x12, x13, x14, x15, x21, x22, x23, x24, x25, x31, x32, x33, x34, x35]

lp += 8 * x11 + 12 * x12 + 9 * x13 + 8 * x14 + 0 * x15 + 11 * x21 + 9 * x22 + 16 * x23 + 0 *
x24 + 8 * x25 + 14 * x31 + 0 * x32 + 10 * x33 + 9 * x34 + 12 * x35

lp += x11 + x12 + x13 + x14 + x15 <= 1200
lp += x21 + x22 + x23 + x24 + x25 <= 1200
lp += x31 + x32 + x33 + x34 + x35 <= 1200

lp += x11 + x21 + x31 == 700
lp += x12 + x22 + x32 == 300
lp += x13 + x23 + x33 == 900
lp += x14 + x24 + x34 == 600
lp += x15 + x25 + x35 == 500

lp.solve()

for variable in variables:
    print('{} = {}'.format(variable, pulp.value(variable)))

```

Elde edilen sonuçlar şöyledir:

```

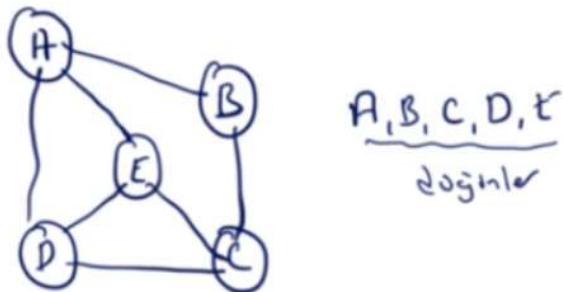
x11 = 700.0
x12 = 0.0
x13 = 0.0
x14 = 0.0
x15 = 500.0
x21 = 0.0
x22 = 0.0
x23 = 0.0
x24 = 600.0
x25 = 0.0
x31 = 0.0
x32 = 300.0
x33 = 900.0
x34 = 0.0
x35 = 0.0

```

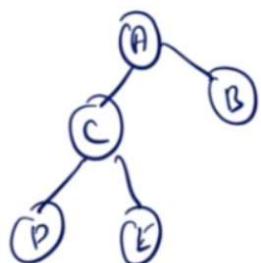
GRAFLAR ÜZERİNDE İŞLEMLER

Graflar optimizasyon uygulamalarında en çok karşılaşılan veri yapılarından biridir. Biz bu bölümde graf veri yapısının Python'da nasıl kullanılacağı ve bu veri yapısıyla temel graf optimizasyon problemlerinin nasıl çözüleceği üzerinde duracağız.

Graflar düğümlerden(nodes) ve kenarlardan (edges) oluşmaktadır. Genellikle şekilsel olarak düğümler yuvarlaklarla, kenarlar da çizgilerle gösterilirler. Örneğin:

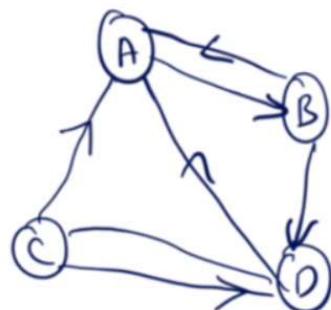


Veri yapılarında bir düğüme birden fazla yolla ulaşılabilirse buna graf ancak bir düğüme tek bir yolla ulaşılıyorsa buna da ağaç (tree) denilmektedir. Ağaçlar grafların özel bir durumudur. Aşağıda bir ağaç veri yapısı görülmektedir:



Graflarda düğümlere ve kenarlara bilgiler ilişirilebilir. Örneğin Mecidiyeköy'den Beşiktaş'a gidilebilecek en kısa yolu bulmak isteyelim. Bu bir graf problemidir. Yani problemin öncelikle bir graf olarak modellenmesi gereklidir. Bu modelleme yapılırken kavşak noktaları düğüm olarak alınır. Düğümler arasındaki tüm yollar kenarlarla grafta belirtilir. Sonra da graf üzerinde en kısa yol problemi uygulanır.

Graflar yönsüz (undirected) ve yönlü (directed) olmak üzere ikiye ayrılmaktadır. Yönsüz graflar aslında çift yönlü graflar biçiminde değerlendirilebilir. Yönsüz graflarda A ve B düğümü arasında bir kenar varsa bu durum hem A'dan B'ye hem de B'den A'ya gidiş olduğu anlamına gelir. Yönlü graflarda ise A'dan B'ye gidiş olması B'den A'ya gidiş olacağının anlamına gelmemektedir. Yönlü graflar genellikle kenarlarda oklarla belirtilirler. Örneğin:



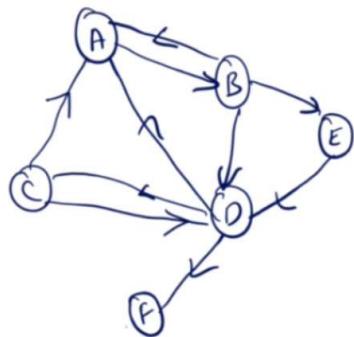
Örneğin karayollarını temsil eden graflar yönlüdür. Fakat sosyal ağlardaki arkadaşlık grafları yönüzdür.

Yapay zeka uygulamaları bazen graf veri yapısı ilgili olabilmektedir. Örneğin sosyal ağlar graf veri yapısı ile temsil edilip bunun üzerinde yapay zeka algoritmaları uygulanabilmektedir.

Graf veri yapıları tipik olarak iki yöntemle gerçekleştirilmektedir: Komşuluk Matrisi (Adjacency Matrix) ve Komşuluk Listeleri (Adjacency Lists). Komşul matrisi yöntemi daha seyrek kullanılmaktadır. Bu yöntemde hangi düğümden hangi düğüme yol olduğu bir matrisle belirlenir. Örneğin yukarıdaki graf için komşuluk matrisi şöyle oluşturulabilir:

	A	B	C	D
A	1	0	0	0
B	1	1	0	1
C	1	0	1	1
D	1	0	1	1

Komşuluk listesi yöntemi en sık kullanılan yöntemdir. Bu yöntemde her düğümün hangi düğümlerle bağlı olduğu bir listede tutulur. Tabii bağlı düğümlerin sayısı fazlaysa liste yerine aramayı hızlandırmak için sözlük tarzı bir veri yapısı tercih edilmektedir. Komşuluk listesi yöntemi Python'da sözlük nesneleriyle basit biçimde oluşturulabilir. Örneğin aşağıdaki yönlü grafi komşuluk listesi yöntemiyle Python'da oluşturmak isteyelim:



```
graph = {A: {B}, B: {A, D, E}, C: {A, D}, D: {A, F, C}, E: {D}, F: {}}
```

Fakat graflarda düğümlere ve kenarlara da bilgiler ilişirilebilmektedir. Peki bu durumda nasıl bir genelleştirme yapılabilir? Örneğin Networkx kütüphanesi grafi aşağıdaki gibi temsil etmiştir:

```

graph TD
    nodes = [{"A": {"name": "Aster", "size": 1200}, "B": {"name": "Lunar", "size": 2100}, "C": {"length": 1200}}]
    edges = [{"A": {"B": {"length": 1200}}, "C": {"length": 1200}}]

```

Göründüğü gibi graf düğümlerden ve kenarlardan oluşmaktadır. Düğümler bir sözlük içerisinde yer almaktadır. Ama sözlüğün anahtarına karşılık gelen değer de sözlüktür. Çünkü bir düğüme atanacak birden fazla özellik olabilir. Benzer biçimde kenarlar da birer sözlük biçimindedir. Her kenara ilişirilecek bilgi de bir sözlükle temsil edilmiştir. Bu bilgiler de birden fazla olabilirler.

Graf veri yapısı için Python'ın standart kütüphanelerinde bir modül yoktur. Ancak graf işlemleri için üçüncü parti çeşitli kütüphaneler bulunmaktadır. Bunların arasından en çok tercih edilenlerden ikisi Networkx ve igraph kütüphaneleridir. Biz buarada Networkx kütüphanesini ele alacağız.

Anaconda sürümünde Networkx default olarak yüklenmiş biçimde gelmektedir. Diğer sürümlerde kütüphane manuel biçimde pip yoluyla yüklenebilir. Biz kütüphaneyi nx ismiyle import ederek kullanacağız. Örneğin:

```
import networkx as nx
```

Kütüphanenin dokümantasyonuna şuradan erişilebilir:

<https://networkx.github.io/documentation/stable/>

Networkx Kütüphanesinin Kullanımı

Kütüphanenin tipik kullanımı şöyledir:

1) Kütüphanede toplam dört farklı graph veri yapısı bulunmaktadır. Graph isimli sınıf yönsüz (undirected) graflar için, DiGraph isimli sınıf yönlü graflar için, MultiGraph yönsüz paralel kenarlara izin veren graflar için, MultiDiGraph yönlü paralel kenarlara izin veren graflar için kullanılmaktadır. Tipik olarak önce bu sınıflar türünden bir graph nesnesi yaratılır. Graph ve DiGraph sınıfının `__init__` metodlarının parametrik yapıları şöyledir:

```
Graph(incoming_graph_data=None, **attr)  
Graph(incoming_graph_data=None, **attr)
```

Bu fonksiyonla `**` parametresini kullanarak grafa istediğimiz bilgileri iliştirebiliriz. Bu bilgiler grafi temsil eden herhangi şeyler olabilir. `incoming_graph_data` grafi yaratırken hemen düğüm ve kenar girmek için kullanılmaktadır. Bu girişin bazı biçimleri vardır. Bunun için dokümanlara başvurabilirsiniz. Örneğin:

```
import networkx as nx
```

```
graph = nx.Graph()
```

Burada biz boş bir graph yattık. Tabii istersek graph nesnesine kendi istediğimiz birtakım bilgileri `**` parametresine karşı gelecek biçimde isimli olarak girebiliriz. Örneğin:

```
import networkx as nx
```

```
graph = nx.Graph(name='MyGraph', date='19/07/2020')
```

Graf kenarlar verilerek dolu bir biçimde de yaratılabilir:

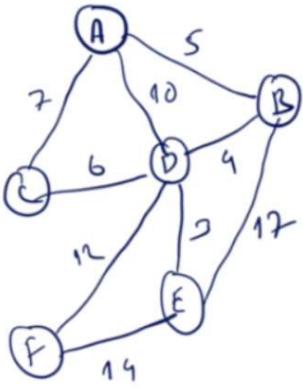
```
import networkx as nx
```

```
edges = [('A', 'B'), ('A', 'C'), ('C', 'B')]  
graph = nx.Graph(edges)
```

2) Yaratılmış grafa biz `add_edge` metoduna kenar, `add_node` metoduyla düğüm ekleyebiliriz. Önce düğüm ekleyip sonra kenar eklemek zorunlu değildir. Bir kenar eklenirken kenara konu olan düğümler yoksa zaten bu düğümler de garafa eklenmektektir.

```
add_edge(u_of_edge, v_of_edge, **attr)
```

Fonksiyonun ilk iki parametresi başlangıç ve bitiş düğümlerini belirtir. Düğümler hash'lenebilir (hashable) herhangi bir türden oabilir. (Örneğin, int, float, str gibi.) `**` parametresi bizim ilgili kenara istediğimiz gibi atayabileceğimiz özellikleri belirtmektedir. Örneğin aşağıdaki grafın yollarını `add_edge` ile oluşturmaya çalışalım:



Buradaki sayılar ilgili yolların uzunluğu olsun. Networkx graf veri yapısı şöyle oluşturulabilir:

```
import networkx as nx

g = nx.Graph(title='Road Graph')

g.add_edge('A', 'B', length=5)
g.add_edge('A', 'C', lenght=7)
g.add_edge('A', 'D', lenght=10)
g.add_edge('D', 'B', lenght=4)
g.add_edge('D', 'C', lenght=6)
g.add_edge('D', 'F', lenght=12)
g.add_edge('D', 'E', lenght=3)
g.add_edge('B', 'E', lenght=17)
g.add_edge('F', 'E', lenght=10)
```

Göründüğü gibi düğüm eklenmesine hiç gerek duyulmamıştır. `add_edge` metodunda belirtilen düğümler zaten varsa olan düğümler kullanılır. Eğer `add_edge` metodunda belirtilen düğümler henüz yoksa zaten `add_edge` tarafından yaratılmaktadır. Tabii bazen yalnızca düğüm de yaratmak isteyebiliriz. Yani hiç kenarı olmayan bir düğüm de söz konusu olabilir. Ya da önce düğümleri yaratıp onlara birtakım bilgiler iliştirip ondan sonra kenarları yaratabiliz. Düğüm yaratma işlemi `add_node` metoduyla yapılmaktadır:

```
add_node(node_for_adding, **attr)
```

Fonksiyonun birinci parametresi eklenecek düğümü belirtir. Yine bu düğüm hash'lenebilir herhangi bir nesne olabilir. Buradaki `**` parametresi düğüme iliştirilecek bilgileri belirtmektedir. Grafi `add_edge` ile oluştururken düğüme bilgi iliştiremediğimize dikkat ediniz. Eğer düğümlere bilgiler iliştirilmek isteniyorsa önce düğümler `add_node` ile eklenebilir. Tabii aslında bir düğüme yaratıldıktan sonra da bilgiler iliştirebiliriz.

Graf sınıflarının `number_of_nodes` ve `number_of_edges` isimli metodları graftaki düğüm ve kenar sayılarını bize verir. Örneğin:

```
print(g.number_of_edges())
print(g.number_of_nodes())
```

Graftaki tüm düğümleri `nodes` örnek özniteligi ile, tüm kenarları da `edges` örnek özniteligi ile alabiliriz. Bu sınıflar bize dolaşılabilir bir nesne verirler. Bu nesneler dolaşıldığında bizim düğümler için verdigimiz nesneler (yukarıdaki örnekte 'A', 'B' biçiminde harfler) kenarlar dolaşıldığında da iki düğümden oluşan demetler elde edilmektedir. Örneğin:

```
for node in g.nodes:
    print(node, '--->', type(node))

for edge in g.edges:
    print(edge, '--->', type(edge))
```

Kodun ekran çıktısı şöyle olacaktır:

```
A ---> <class 'str'>
B ---> <class 'str'>
C ---> <class 'str'>
D ---> <class 'str'>
F ---> <class 'str'>
E ---> <class 'str'>
('A', 'B') ---> <class 'tuple'>
('A', 'C') ---> <class 'tuple'>
('A', 'D') ---> <class 'tuple'>
('B', 'E') ---> <class 'tuple'>
('B', 'D') ---> <class 'tuple'>
('C', 'D') ---> <class 'tuple'>
('D', 'F') ---> <class 'tuple'>
('D', 'E') ---> <class 'tuple'>
('F', 'E') ---> <class 'tuple'>
```

Dolaşılabilir sınıfın `__str__` metodu bize düğümleri ve kenarları bir demet yazısı biçiminde vermektedir. O halde biz doğrudan nodes ve edges örnek özniteliklerini print fonksiyonuyla da yazdırabiliriz.

```
print(g.nodes)
print(g.edges)
```

Kodun çıktısı şöyle olacaktır:

```
['A', 'B', 'C', 'D', 'F', 'E']
[('A', 'B'), ('A', 'C'), ('A', 'D'), ('B', 'E'), ('B', 'D'), ('C', 'D'), ('D', 'F'), ('D', 'E'), ('F', 'E')]
```

Graftan herhangi bir düğüm `remove_node` metodu ile silinebilir. Bir düğüm silindiğinde o düğüme bağlı olan kenarların hepsi de silinmektedir. Benzer biçimde bir kenar da `remove_edge` metoduyla silinebilmektedir. Ancak kenar silindiğinde kenara ilişkin düğümler silinmemektedir.

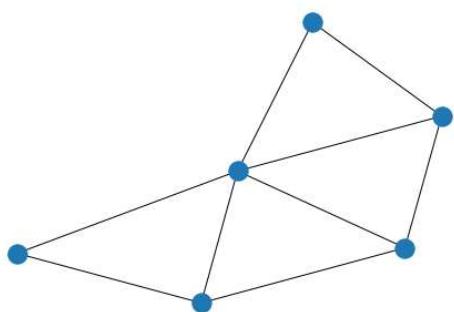
Graftaki belli bir düğümün komşularını (yani o düğüme bağlı kenarları) elde etmek için [...] operatörü kullanılabilir. Bu işlem bize sözlük tarzı (AtlasView isimli bir sınıf) dolaşılabilir bir nesne vermektedir. Örneğin:

```
for s in g['A']:
    print(s)
```

Burada A'nın komşuları yazı olarak elde edilmektedir. Eğer biz A ile B arasındaki kenar özelliklerini elde etmek istiyorsak `g['A']['B']` ifadesini kullanmalıyız.

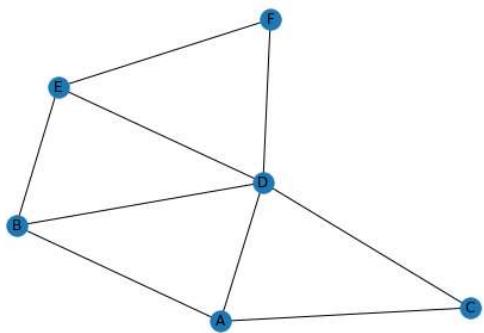
Oluşturulan grafın çizdirilmesi oldukça kolaydır. Tek yapılacak şey networkx modülündeki `draw` fonksiyonunu kullanmaktır. Örneğin:

```
nx.draw(g)
```



Düğümlerin üzerinde isimlerin yazılması isteniyorsa `with_labels` parametresi `True` geçilmelidir. Örneğin:

```
nx.draw(g, with_labels=True)
```

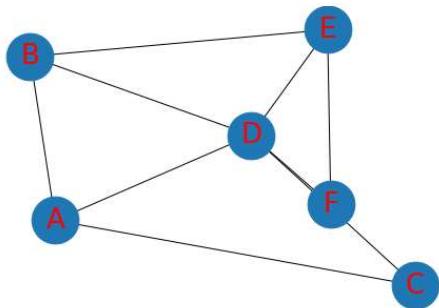


Fonksiyonun pek çok isimli parametresi vardır. Bu parametreler aşağıdaki link'ten incelenebilir:

https://networkx.github.io/documentation/stable/reference/generated/networkx.drawing.nx_pylab.draw_networkx.html?highlight=draw_networkx#networkx.drawing.nx_pylab.draw_networkx

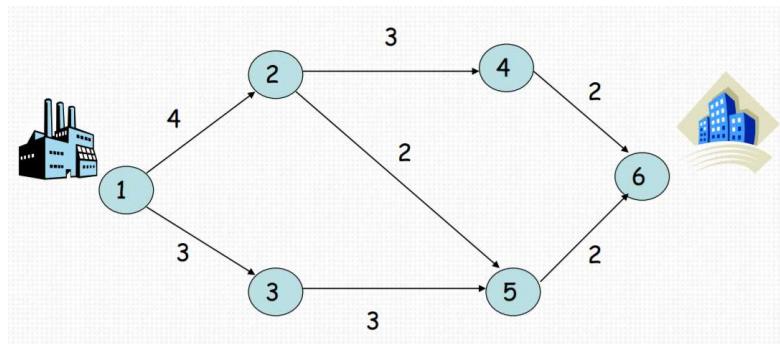
Örneğin:

```
nx.draw(g, with_labels=True, font_size=26, node_size=2000, font_color='red')
```



Tabii aslında graflar birtakım optimizasyon tarzı problemleri çözmek için kullanılmaktadır. Yani graflar aslında yalnızca veri yapısıdır. Önemli olan bu veri yapısı üzerinde yararlı işlemler yapabilen algoritmaların çalıştırılmasıdır. İşte networkx kütüphanesinde de graflar üzerinde işlemler yapabilen pek çok hazır fonksiyon vardır.

Yukarıda da belirtildiği gibi `Graph` sınıfı yönsüz (yani çift yönlü) graf oluşturmaktadır. Halbuki `DiGraph` sınıfı yönlü graf oluşturur. Örneğin aşağıdaki yönlü grafi oluşturmak isteyelim:



```
import networkx as nx

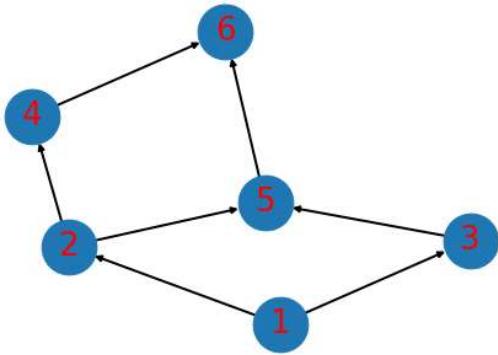
g = nx.DiGraph(title='Road Graph')
```

```

g.add_edge(1, 2, length=4)
g.add_edge(1, 3, length=3)
g.add_edge(3, 5, length=3)
g.add_edge(5, 6, length=2)
g.add_edge(2, 4, length=3)
g.add_edge(2, 5, length=2)
g.add_edge(4, 6, length=2)

nx.draw(g, with_labels=True, font_size=24, font_color='white', node_size=1000)

```



Aslında graf bilgileri bazı uygulamalarda komşuluk matrisi biçiminde de ifade edilebilmektedir. Örneğin yukarıdaki grafın komşuluk matrisi şöyledir:

```

0 4 3 0 0 0
0 0 0 3 2 0
0 0 0 0 3 0
0 0 0 0 0 2
0 0 0 0 0 2
0 0 0 0 0 0

```

Bu komşuluk matrisinden yönlü graf elde eden fonksiyon basit biçimde şöyle yazılabilir:

```

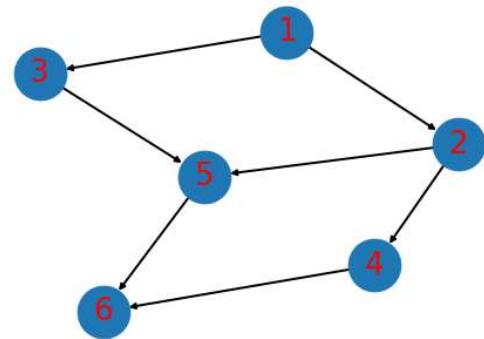
import numpy as np
import networkx as nx

def adjacency_matrix_to_graph(path):
    g = nx.DiGraph()
    data = np.loadtxt(path, dtype=np.float32)
    for i in range(len(data)):
        for k in range(len(data)):
            if data[i, k] != 0:
                g.add_edge(i + 1, k + 1, length=data[i, k])

    return g

g = adjacency_matrix_to_graph('graph.txt')
nx.draw(g, with_labels=True, font_size=26, node_size=2000, font_color='red', width=2)

```



Aslında bu işi yapan networkx kütüphanesinde `from_numpy_matrix` isimli bir fonksiyon da vardır. Örneğin:

```
data = np.loadtxt('graph.txt', dtype=np.float32)
g = nx.from_numpy_matrix(data)
nx.draw(g, with_labels=True, font_size=26, node_size=2000, font_color='red', width=2)
```

Tabii bunun tersini yapan `adjacency_matrix` isimli bir fonksiyon da vardır:

```
data2 = nx.adjacency_matrix(g)
print(data2)
print(data2.toarray())
```

Graflar üzerinde çeşitli optimizasyon işlemleri için `networkx.algorithms` modülündeki fonksiyonlar kullanılmaktadır. Networkx kütüphanesi bu bakımdan çok zengindir. Burada bazı graf problemleri örnek olarak verilecektir.

Graflar üzerindeki en klasik problem "en kısa yol (shortest path)" problemidir. Bu problemde bir düğümden bir düğüme gidebilmek için gereken en kısa yol bulunmaya çalışılır. En kısa yol problemi için `algorithms.shortest_paths.generic.shortest_path` fonksiyonu kullanılmaktadır. Bu fonksiyona biz başlangıç düğümünü, bitiş düğümünü ve neye göre uzunluk belirleneceğine belirten kenar özelliğini veriririz. O da bize en kısa yolu düğümlerde oluşan bir liste olarak verir. Örneğin:

```
import networkx as nx

g = nx.DiGraph(title='Road Graph')

g.add_edge(1, 2, length=4)
g.add_edge(1, 3, length=3)
g.add_edge(1, 2, length=4)
g.add_edge(2, 4, length=3)
g.add_edge(2, 5, length=2)
g.add_edge(3, 5, length=3)
g.add_edge(4, 6, length=2)
g.add_edge(5, 6, length=4)

print(g.number_of_edges())
print(g.number_of_nodes())

nx.draw(g, with_labels=True, font_size=26, node_size=2000, font_color='red', width=2)

from networkx.algorithms.shortest_paths.generic import shortest_path
path = shortest_path(g, 1, 6, weight='length')
print(path)
```

Programın çıktısı şöyle olacaktır:

[1, 3, 5, 6]

Graflar üzerinde turlama (cycling) işlemleri en çok kullanılan işlemleridir. Belli bir düğümden başlanarak tüm düğümlerin ya da kenarların elde edilmesi gerekebilir. Örneğin simple_cycles isimli fonksiyon bize graf içerisindeki kısa ya da uzun bütün turları vermektedir:

```
import networkx as nx

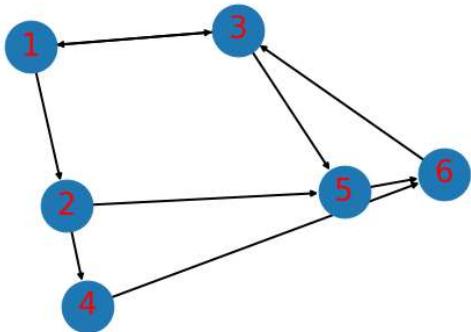
g = nx.DiGraph(title='Road Graph')

g.add_edge(1, 2, length=4)
g.add_edge(1, 3, length=3)
g.add_edge(1, 2, length=4)
g.add_edge(2, 4, length=3)
g.add_edge(2, 5, length=2)
g.add_edge(3, 5, length=3)
g.add_edge(4, 6, length=2)
g.add_edge(5, 6, length=4)
g.add_edge(6, 3, length=7)
g.add_edge(3, 1, length=3)

print(g.number_of_edges())
print(g.number_of_nodes())

nx.draw(g, with_labels=True, font_size=26, node_size=2000, font_color='red', width=2)

cycles = nx.simple_cycles(g)
print(list(cycles))
```



```
[[1, 3], [1, 2, 5, 6, 3], [1, 2, 4, 6, 3], [3, 5, 6]]
```

Yukarıda da belirtildiği gibi graflarda pek çok algoritmik problem vardır. networkx kütüphanesi bu problemlerin çoğunu çözecek fonksiyonlara sahiptir. Örneğin "en küçük örten ağaç (minimum spanning tree)" yönsüz bir graftaki bütün düğümleri içeren en kısa ağacın oluşturulma problemidir. Su boruları ya da elektrik telleri gibi alt yapı hizmetlerinde önemli kullanımı vardır. (Örneğin tüm evlere su götürecek en kısa uzunluklu boru hattının oluşturulması örneği gibi.) Bu algoritma networkx.algorithms.tree.mst modülündeki minimum_spanning_edges fonksiyonuyla çözülmektedir. Örneğin:

```
import networkx as nx

g = nx.Graph(title='Road Graph')

g.add_edge(1, 2, length=4)
g.add_edge(1, 3, length=3)
g.add_edge(1, 2, length=4)
g.add_edge(2, 4, length=3)
g.add_edge(2, 5, length=2)
g.add_edge(3, 5, length=3)
g.add_edge(4, 6, length=2)
g.add_edge(5, 6, length=4)
g.add_edge(6, 3, length=7)
```

```

print(g.number_of_edges())
print(g.number_of_nodes())

nx.draw(g, with_labels=True, font_size=26, node_size=2000, font_color='red', width=2)

from networkx.algorithms.tree.mst import minimum_spanning_edges
se = minimum_spanning_edges(g)
print(list(se))

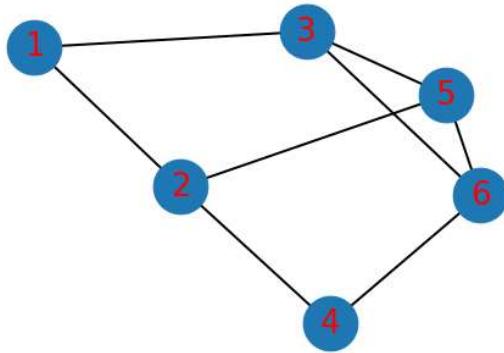
```

Programın çıktısı şöyle olacaktır:

```

8
6
[((1, 2, {'length': 4}), (1, 3, {'length': 3}), (2, 4, {'length': 3}), (2, 5,
{'length': 2}), (3, 6, {'length': 7})]

```



Diğer klasik problemlerden biri de "maksimum akış (maximum flow)" problemidir. Bu problemde belli bir düğümden belli bir düğüme maksimum kapasite aktarımı yapmak istenmektedir. Tipik durumda bu kapasite aktarımı borulardan suların aktarılması biçiminde düşünülebilir. Problemden amaç çeşitli yollar kullanarak maksimum akışın elde edilmesidir. Şüphesiz iki düğüm arasındaki akış o düğümlerdeki yollarda bulunan boruların en küçük kapasitelisi ile ilgilidir. Örnek bir problem şöyle verilebilir:

```

import networkx as nx

g = nx.DiGraph()
g.add_edge('x','a', capacity=3.0)
g.add_edge('x','b', capacity=1.0)
g.add_edge('a','c', capacity=3.0)
g.add_edge('b','c', capacity=5.0)
g.add_edge('b','d', capacity=4.0)
g.add_edge('d','e', capacity=2.0)
g.add_edge('c','y', capacity=2.0)
g.add_edge('e','y', capacity=3.0)

nx.draw(g, with_labels=True, font_size=26, node_size=2000, font_color='red', width=2)

from networkx.algorithms.flow import maximum_flow

flow_value, flow_dict = maximum_flow(g, 'x', 'y')
print(flow_value)
print(flow_dict)

```

Graf Çizme İçin Graphviz Kütüphanesinin Kullanımı

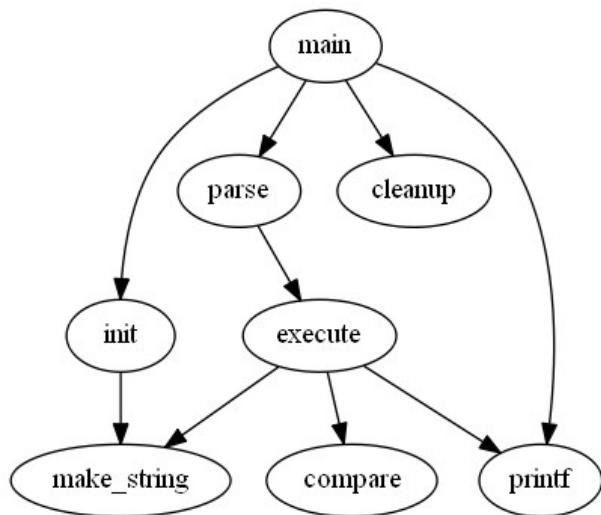
Graphviz aslında genel amaçlı bir graf çizim kütüphanesidir. Bu kütüphane komut satırından ya da pek çok dilden kullanılabilir. Tipik kullanımda kullanıcı uzantısı .dot ya da .gv olacak biçimde bir text dosyasına Graphviz dilini kullanarak grafını yazışal biçimde oluşturur. Sonra dot isimli programı çalıştırır. Bu program Graphviz kaynak

dosyasını okuyarak buradan hareketle bir çizim dosyası oluşturmaktadır. Örneğin aşağıdaki gibi bir sample.gv dosyası oluşturmuş olalım:

```
digraph G {  
    main -> parse -> execute;  
    main -> init;  
    main -> cleanup;  
    execute -> make_string;  
    execute -> printf  
    init -> make_string;  
    main -> printf;  
    execute -> compare;  
}
```

Aşağıdaki gibi bir komutla bir png dosyası elde edebiliriz:

```
dot -Tpng sample.gv > test.jpg
```



Graphviz kütüphanesinin dokümantasyonuna aşağıdaki bağlantıdan erişilebilir:

<https://graphviz.gitlab.io/documentation/>

Graphviz Kütüphanesi aslında pek çok dilden kullanılabilimektedir. Biz burada Python'daki kullanımı üzerinde duracağız. Kütüphanenin Python için genel dokümantasyonuna aşağıdaki bağlantıdan erişilebilir:

<https://graphviz.readthedocs.io/en/stable/manual.html>

Graphviz kütüphanesi Python'dan kullanımı tipik olarak şöyle kullanılmaktadır:

1) Önce graph ilgili sınıf kullanılarak (yönsüz için Graph, yönlü için DiGraph) yaratılır. Örneğin:

```
import graphviz as gv
```

```
dot = gv.Digraph('My Graph')
```

2) Graf için düğümler yaratılır. Bu işlem Graph sınıflarının node isimli metodlarıyla yapılmaktadır. Bu metodlar bizden düğümün ismini ve etiketini parametre olarak alırlar. Etiket verilmek zorunda değildir. Örneğin:

```
import graphviz as gv
```

```
dot = gv.Digraph('My Graph')
```

```
dot.node('A', 'Adana')
dot.node('E', 'Eskişehir')
dot.node('K', 'Kastamonu')
dot.node('O', 'Ordu')
dot.node('İ', 'İstanbul')
```

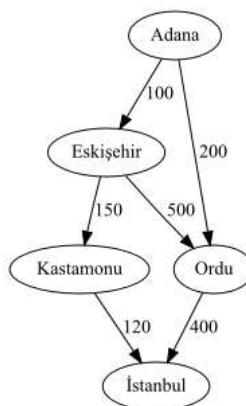
3) Bundan sonra kenarlar edge metoduyla eklenir. Örneğin:

```
dot.edge('A', 'E', label='100')
dot.edge('A', 'O', label='200')
dot.edge('E', 'K', label='150')
dot.edge('O', 'İ', label='400')
dot.edge('K', 'İ', label='120')
dot.edge('E', 'O', label='500')
```

Sınıfın source isimli örnek özniteliği bize çizilecek grafın script dosya içeriğini verir. (Yani biz bu sctipt'i bir dosyaya yapıp dot programıyla da görüntü dosyası oluşturabiliriz.) Komut satırında grafi çizdirmek için tek yapılacak şey nesne ismini yazarak ENTER tuşuna basmaktadır. Örneğin:

In [2]: dot

Out[2]:



Sınıfın render isimli metodu grafik dosyasını oluşturmakta kullanılabilir. render metoduyla default durumda bu grafik dosyası hem oluşturulmakta hem de çizdirilmektedir. Örneğin:

```
dot.render('test.gv', format='png', view=True)
```

Elimizde dot diliyye yazılmış bir script yazısı varsa biz graphviz içerisindeki Source fonksiyonuyla bir Graph ya da Digraph nesnesi elde edebiliriz. Örneğin:

```
g = gv.Source(text)
```

Aslında Source bir sınıfıtır. Bu sınıfın from_file metoduyla da biz doğrudan dosyanın yol ifadesini vererek graf nesnesini elde edebiliriz.

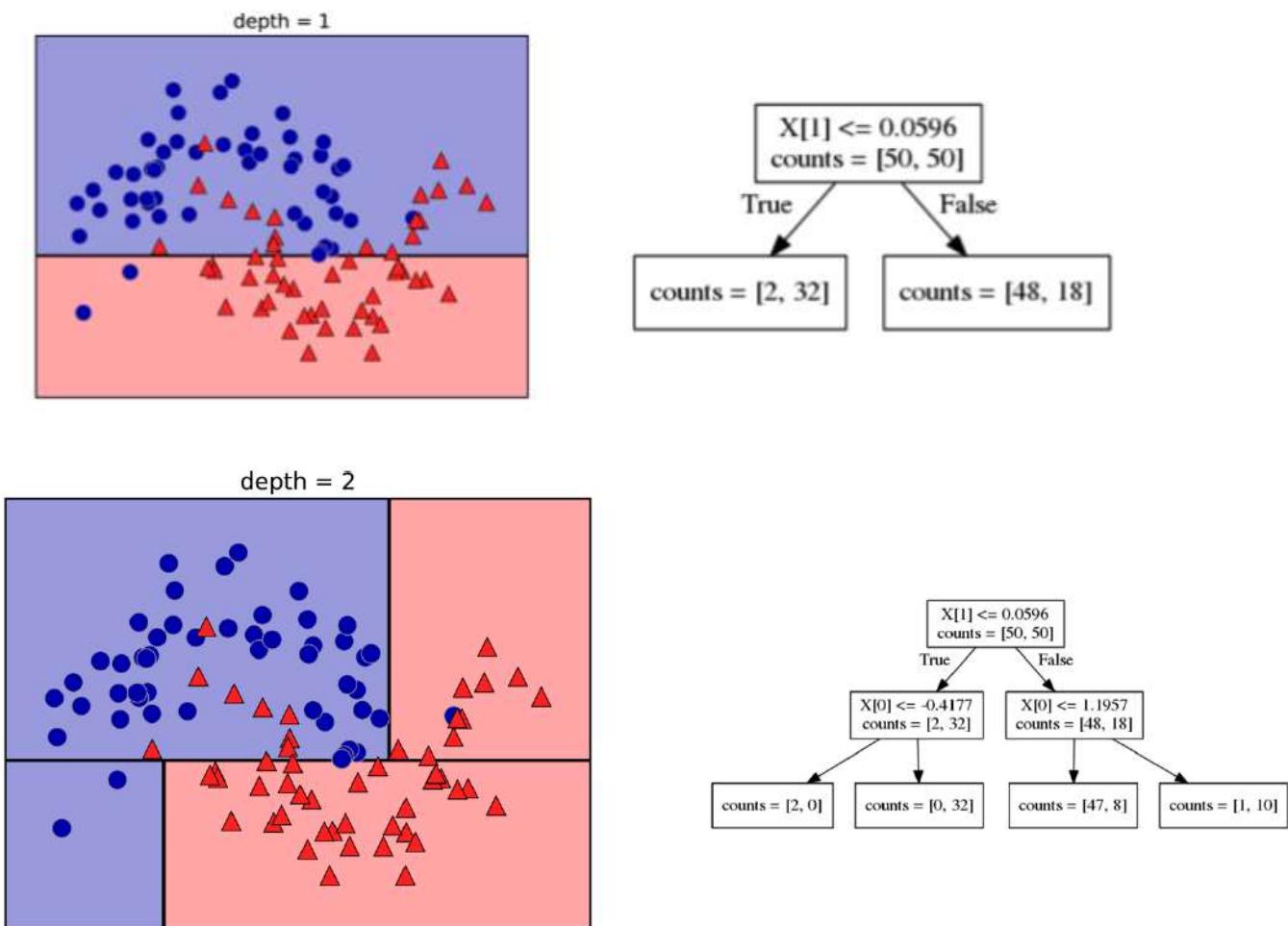
```
g = gv.Source.from_file(path)
```

KARAR AĞAÇLARI (DECISION TREES)

Karar ağaçları sınıflandırma ve regresyon tarzı problemlerde kullanılan alternatif yöntemlerden biridir. Bu bakımdan bu yöntemin diğer alternatif yöntemlere göre bazı avantajları ve dezavantajları söz konusu olabilmektedir. Karar ağaçlarının çalışma mekanizması şöyle bir oyunla anlatılabilir: Oyunda oyunculardan biri aklından bir nesne tutar. Diğer Doğru-Yanlış yanıt beklenen sorular sorarak bu nesneyi bulmaya çalışır. Örnek sorular şöyle olabilir:

- Canlı mı cansız mı?
- Hafif mi ağır mı?
- Hacim olarak büyük mü küçük mü?
- Hareketli mi sabit mi?
-

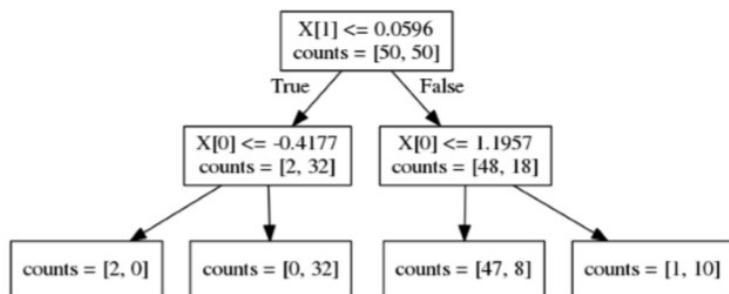
İşte bu soruların her birinde aslında biz gitgide alanı daraltarak sonuca yaklaşmış oluruz. Daha önce biz sınıflandırma problemlerini eğer doğrusal olarak ayırtılabilir (linearly separable) ise uygun doğruyla ayırtırmaya çalıştık. İşte karar ağaçları yöntemi de aslında yine doğrularla ayırtırmaya çalışır fakat bu yöntemde ayırtırma için tek bir doğru kullanılmamaktadır. Birden fazla doğruyla sınıflandırma yapılmaya çalışılır. Aşağıdaki şekilde iki sütunlu (x ve y) bir veri kümesinde sürekli bölme yöntemiyle uygun sınıfı elde etmenin grafiksel çizimi görülmektedir:



Çizimler ve karar ağıacı resimleri "Introduction To Machine Learning With Python" kitabından alınmıştır.

Burada görüldüğü gibi birinci soruda sınıflandırma iyi bir düzeyde yapılamamıştır. İkinci soruda sınıflandırma daha iyi bir düzeyde yapılmıştır. Burada iki sütunlu (x ve y) bir veri tablosu söz konusudur. Önce sütunlardan birine ilişkin (y sütunu) soru sorulmuş bu soruya göre ağaç ikiye ayrılmıştır. Sonra her iki alt ağaç için diğer bir soru (x sütunu) sorularak toplam dört bölge elde edilmiştir.

Yukarıdaki örnekte iki kademe bölme sonucunda elde edilen ağacı inceleyiniz:



Bu ağaçta kök düğümde sorulan soru $X[1] \leq 0.0596$ 'dır. Bu soruya göre bu koşula uyan ve uymayan satırlar elde edilmiştir. Ağaçtaki counts bilgisi ilgili bölgede kaç tane A sınıfından (yuvarlak mvi) kaç tane B sınıfından (üçgen kırmızı) nokta olduğunu belirtmektedir. Örneğin counts = [2, 32] o bölgede 2 tane A sınıfından 32 tane de B sınıfından nokta olduğunu anlatmaktadır. Bizim karar ağaçları yönteminde amacımız bölmelere devam etmek ve en sonunda homojen bölgeler elde etmektir. Bölgelerin homojen olması demek yalnızca tek gruptan nokta bulunması demektir. Yukarıdaki şekilde en soldaki yapraktaki counts değeri [2, 0] biçiminde olduğuna dikkat ediniz. Yani o bölgede 2 tane A sınıfından nokta vardır fakat hiç B sınıfından nokta yoktur. İşte bu homojen bir durumdur.

Karar ağaçları denetimli (supervised) bir öğrenme teknğini kullanmaktadır. Peki eğitimden sonra nasıl kestirim yapılacaktır? Kestirim oldukça kolaydır. Kestirilecek değerlere ağacın tepesinden itibaren sorular sorulur. Soruların yanıtlarına göre sola ya da sağa sapılarak bir yaprağa varılır. İşte o yaprağın durumu da bize sınıflandırmanın sonucunu verecektir.

Pekiyi karar ağaçları oluştururken hangi sorular hangi sırayla sorulacaktır? İşte bu noktada bilgi teorisi (information theory) konuları devreye girmektedir. Önce hangi, sorunun sorulması gereği ve bu sorunun ne olması gerekiği çeşitli matematiksel yöntemlerle belirlenebilmektedir. Tabii konuda tek bir algoritmik yöntem yoktur. Shannon etropisi, Gini yönetimi en çok kullanılan yöntemlerdendir.

Scikit-Learn Karar Ağaçlarında Sınıflandırma

Scikit-Learn kütüphanesinde karar ağaçları için hazır sınıflar bulunmaktadır. DecisionTreeClassifier isimli sınıf karar ağaçlarıyla sınıflandırma için kullanılmaktadır. Bu sınıfın kullanımı aslında daha önce gördüğümüz sınıfların kullanımına çok benzerdir. Önce DecisionTreeClassifier türünden bir nesne yaratılır. Sonra fit işlemi uygulanır. Kestirim için yine sınıfın predict metodu kullanılmaktadır. Sınıflandırmanın başarısı diğer sınıflarda olduğu gibi sınıfın score fonksiyonuyla elde edilmektedir.

Şimdi bu sınıfı daha önce çalıştığımız meme kanseri veri tablosu ile kullanalım.

```
from sklearn.datasets import load_breast_cancer
bc = load_breast_cancer()
print('Sütun isimleri: {}'.format(bc.feature_names))
print('Sınıf İsimleri: {}'.format(bc.target_names))

dataset_x = bc.data
dataset_y = bc.target

from sklearn.model_selection import train_test_split
training_dataset_x, test_dataset_x, training_dataset_y, test_dataset_y =
train_test_split(dataset_x, dataset_y, test_size=0.2)

from sklearn.tree import DecisionTreeClassifier
```

```

dtc = DecisionTreeClassifier()
dtc.fit(training_dataset_x, training_dataset_y)
score = dtc.score(test_dataset_x, test_dataset_y)
print(score)

```

Burada default değerlerle `DecisionTreeClassifier` nesnesi yaratılmıştır. Daha sonra fit işlemi yapılmıştır. Ağacın oluşturulması fit işlemi sırasında gerçekleştirilmektedir. `score` metodu bir çeşit test metodudur. Biz de burada test veri kümemizle işlemin başarısını tespit etmiş olduk. `score` işlemi sırasında her test verisi ağaç'a sokularak ağaçtan bulunan değer gerçek sonuçla karşılaştırılmış ve sınıflama için bir başarı yüzdesi hesaplanmıştır. Yukarıdaki örnek için başarı yüzdesi %92-%96 civarındadır. Yine sınıfın `predict` metodu kestirim yapmaka kullanılmaktadır. Yani biz ağacı oluşturdukten sonra kestirimde bulunabiliriz.

```

import random

r = random.randint(0, len(test_dataset_x) - 1)
result = dtc.predict(test_dataset_x[r].reshape((1, -1)))

print(bc.target_names[result[0]])
print(test_dataset_y[r] == result[0])

```

`DecisionTreeClassifier` sınıfı ile oluşturulan ağacın graphiz yardımıyla grafi çizdirilebilir. Bunun için `sklearn.tree` modülündeki `export_graphviz` metodu kullanılmaktadır. Bu metot çizimi yapmaz fakat çizim için gereken `.dot` ya da `.gv` dosysını oluşturur. Örneğin:

```

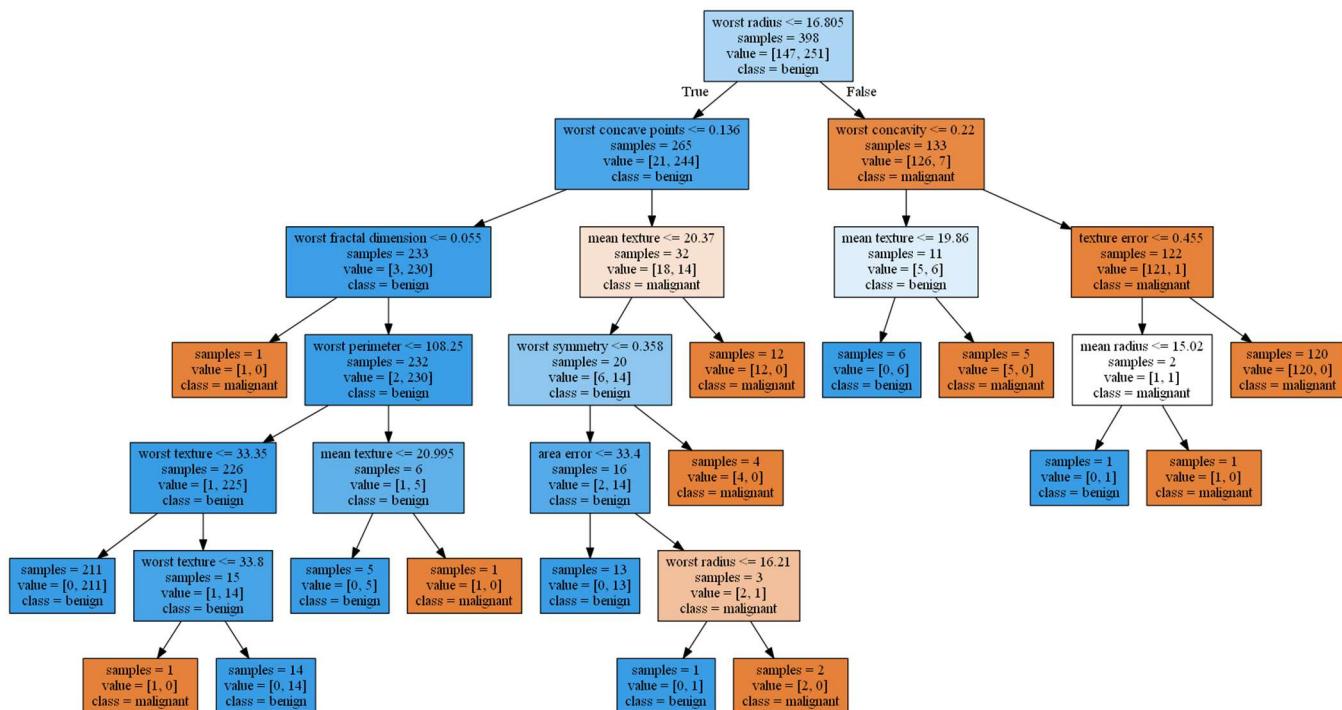
from sklearn.tree import export_graphviz
import graphviz

export_graphviz(dtc, out_file='dt.gv', class_names=['malignant', 'benign'],
feature_names=bc.feature_names, impurity=False, filled=True)

g = graphviz.Source.from_file('dt.gv')

```

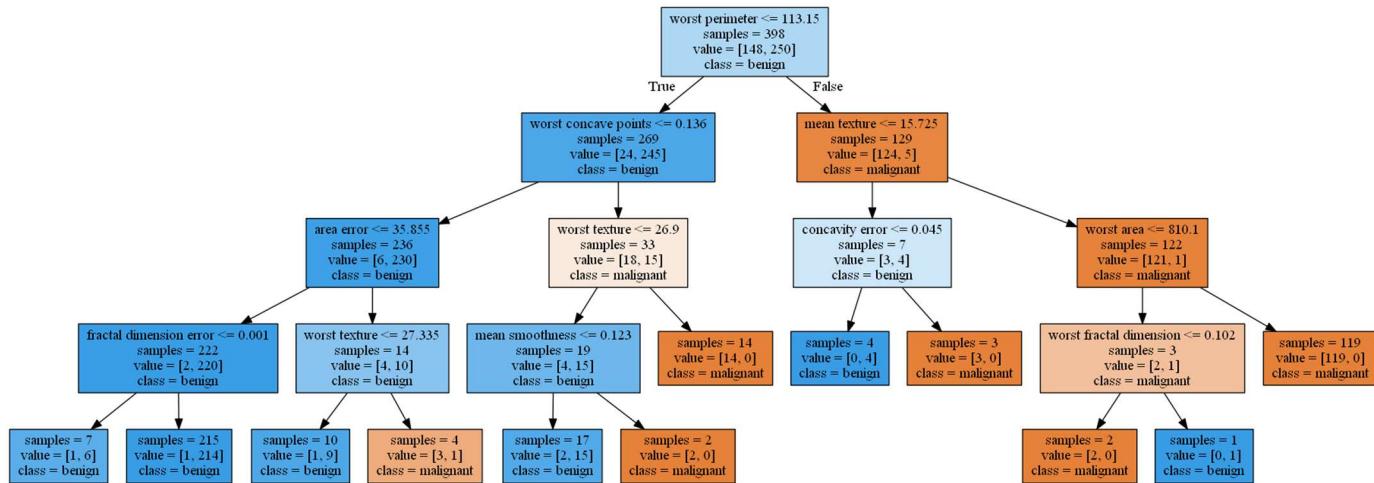
Elde edilen graf aşağıdaki gibidir:



Burada yaprakaların homojen (pure) durumda olduğuna dikkat ediniz. Toplam özellik sayısı 30 olduğu halde eğitimde 6 derinlikte sonuca ulaşılmıştır. Biz DecisionTreeClassifier tür fonksiyonunda bu ağacın oluşturulması ile ilgili çeşitli belirlemeler yapabilmekteyiz. DecisionTreeClassifier sınıfının `__init__` metodunun parametrik yapısı şöyledir:

```
sklearn.tree.DecisionTreeClassifier(criterion='gini', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, class_weight=None, presort='deprecated', ccp_alpha=0.0)
```

Buradaki criterion parametresi 'gini' ya da 'entropy' biçiminde girilebilmektedir. Karar ağaçlarında dallanma için iki temel algoritmanın kullanılabilirliğini belirtmiştir. `max_depth` parametresi ağacın maksimum derinliğini belirlemekte kullanılır. Örneğin bu parametreyi 4 yaparak kodu yeniden çalışıralım.



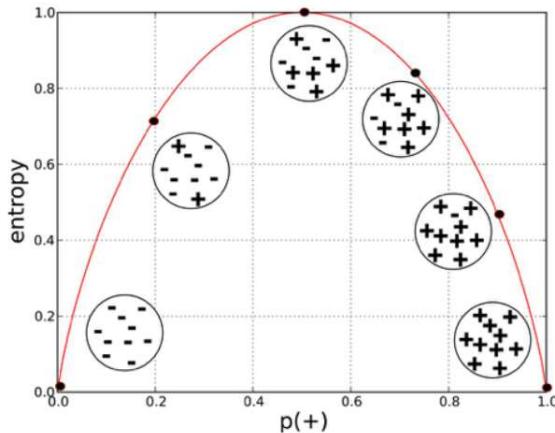
Burada ağacın 4 derinliği olduğu (okların sayısına bakınız) görülmektedir. Yaprakların "pure" olmadığına dikkat ediniz.

Karar Ağaçlarının Algoritmik Temeli

Karar ağaçlarının algoritmik temeli entropy kavramına dayanmaktadır. Entropy sistemdeki düzensizliğin bir ölçüsüdür. Eğer veriler rastgele ise yüksek bir entropy, düzenli ise düşük bir entropy söz konusudur. Karar ağaçlarında entropi hesaplamak için en çok kullanılan yöntem "shannon entropisi" yöntemidir. Shannon entropisi şöyle hesaplanmaktadır:

$$E(S) = \sum_{i=1}^c -p_i \log_2 p_i$$

Buradaki p_i ilgili özellik değerinin veri kümesindeki ortaya çıkma olasılığıdır. Bu olasılık ilgili özelliğin adet toplamının veri kümesindeki satır sayısına bölümü ile elde edilebilir. Entropi düzensizliğin yanı heterojenliğin bir ölçüsüdür. Homojenlik arttıkça entropi azalır, homojenlik azaldıkça entropi artar. Örneğin + ve - karakterlerinin veri kümesi içerisindeki durumlarına göre örnek bir entropi diyagramı şöyle çizilebilir:



Burada görüldüğü gibi homojenlik arttıkça entropi düşmüş, heterojenlik arttıkça entropi artmıştır. Düşey eksende entropi miktarı gösterilmektedir. Entropi miktarı genel olarak $[0, 1]$ aralığında bir gerçek sayı biçiminde ifade edilir. Şekildeki yatay eksende + şekillerinin olasılıkları bulunmaktadır.

Karar ağaçlarında bölünecek en iyi sütunun tespit edilmesi entropi ile ilgilidir. Bölünecek en iyi sütunun tespit edilmesi kabaca şöyle yapılmaktadır: Bütün alternatif sütunlardan bölme yapılır. Her sütun bölündüğünde iki alt düğümün entropileri toplamı hesaplanır. (Alt düğümlerin entropileri hesaplanırken ayrıca Shannon entropy değeri alt düğümün olasılığı ile çarpılmaktadır.) Bu iki alt düğümün entropileri toplamı ana düğümün entropisinden çıkartılır. Buna kazanç (gain) denilmektedir.

$\text{Kazanç} = \text{Üst Düğümün Entropisi} - \text{Alt Düğümlerin Entropileri Toplamı}$

Böylece en büyük kazanca yol açabilecek sütun tespit edilir. Bir sütundaki kazanç ne kadar yüksekse o sütundan bölme yapıldığında o kadar homojen bir alt düğümler elde edilmektedir. Başka bir deyişle kazancın en yüksek olduğu sütun aslında eğer ordan bölme yapılırsa entropinin en fazla düşeceği sütundur." Tabii her bölmenden sonra işlemler yine diğer sütunlar temel alınarak devam ettirilir. Tabii bu işlemin sonucunda aynı sütundan da birden fazla kez bölme yapılıyor olabilir.

Şimdi numpy kullanmadan örnek bir entropy hesabı yapan fonksiyon yazalım. Bu fonksiyonun girdisi çok boyutlu bir liste olsun. Bu listenin son sütun elemanı sınıf değerleri olsun. Örneğin:

```
dataset = [['a', 'x', 'yes'], ['b', 'x', 'yes'], ['a', 'y', 'yes'], ['a', 'x', 'no'], ['a', 'y', 'yes']]
```

Burada iki sütunlu bir veri kümesi vardır. Son sütun "yes", "no" biçiminde iki sınıf belirtmektedir. Bu veri kümesinde karar ağıacı oluşturarak bir tahminlemenin yapılmak istendiğini varsayıyalım. Buradaki dataset'in entropisi nasıl hesaplanacaktır? Entropi hesaplanırken yalnızca sonuçlara bakılır. Yani burada entropi hesabına yalnızca son sütun sokulacaktır. Entropi hesaplayan fonksiyon şöyle yazılabılır:

```
from math import log

def calc_shannon_entropy(dataset):
    nrows = len(dataset)
    label_dict = {}
    for row in dataset:
        label = row[-1]
        if label not in label_dict:
            label_dict[label] = 0
        label_dict[label] += 1
    entropy = 0
    for key in label_dict:
        prob = label_dict[key] / nrows
        entropy += - prob * log(prob, 2)

    return entropy
```

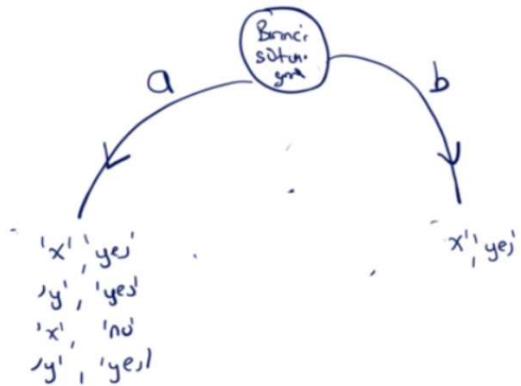
```

dataset = [['a', 'x', 'yes'], ['b', 'x', 'yes'], ['a', 'y', 'yes'], ['a', 'x', 'no'], ['a', 'y', 'yes']]

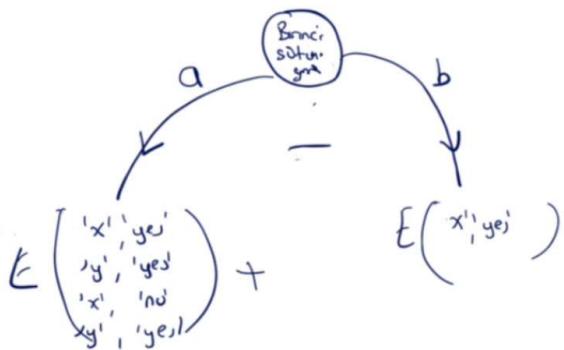
result = calc_shannon_entropy(dataset)
print(result)

```

Şimdi bizim hangi sütuna göre ikiye ayırma yapacağımızı belirlememiz gereklidir. İki tane aday sütun vardır: Birinci sütun ve ikinci sütun. Önce bizim bu iki aday sütuna göre bölme yapıp kazançları hesaplamamız gereklidir. Sonra kazancı en yüksek olan sütun gerçek bölmenin yapılacağı sütun olarak belirlenecek ve o sütun bölünecektir. Biz birinci sütuna göre bölme yaparsak sol ve sağ düğümler şöyle olur:



Biz burada kazancı şöyle hesaplarız:



Şimdi alt düğümlerin entropilerini ve kazancı hesaplayalım:

```

result = calc_shannon_entropy(dataset)
print(result)

first_left = [['x', 'yes'], ['y', 'yes'], ['x', 'no'], ['y', 'yes']]
first_right = [['x', 'yes']]

prob = len(first_left) / len(dataset)
first_left_result = prob * calc_shannon_entropy(first_left)

prob = len(first_right) / len(dataset)
first_right_result = prob * calc_shannon_entropy(first_right)

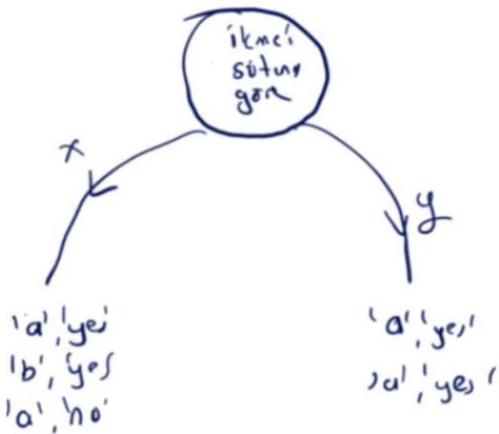
first_parent_result = calc_shannon_entropy(dataset)
first_gain = first_parent_result - (first_left_result + first_right_result)

print(first_gain)

```

Buradan elde edilen sonuç 0.072'dir.

Şimdi ikinci sütundan bölme yaptığımızı düşünelim:



Şimdi alt düğümlerin entropilerini ve kazancı hesaplayalım:

```
second_left = [[['a', 'yes'], ['b', 'yes'], ['a', 'no']]]
second_right = [[['a', 'yes'], ['a', 'yes']]]

prob = len(second_left) / len(dataset)
second_left_result = prob * calc_shannon_entropy(second_left)

prob = len(second_right) / len(dataset)
second_right_result = prob * calc_shannon_entropy(second_right)

second_parent_result = calc_shannon_entropy(dataset)
second_gain = second_parent_result - (second_left_result + second_right_result)

print(second_gain)
```

Buradan 0.170 sonucu elde edilmiştir. Göründüğü gibi $0.072 < 0.170$ biçimindedir. Bu durumda bölme ikinci sütuna göre yapılacaktır.

İşte algoritmada alt düğümler de benzer biçimde bölünerek ilerlenir. Eğer bir düğüm tamamen homojen (pure) biçimde gelirse (yani entropisi 0 olursa) artık o düğüm bir yaprak durumundadır ve daha fazla bölme yapılmaz.

Aslında bölme işlemini yapan bir fonksiyon da benzer biçimde yazılabilir. Örneğin:

```
def split_dataset(dataset, column, value):
    result = []
    for row in dataset:
        if row[column] == value:
            result.append(row[:column] + row[column + 1:])

    return result
```

Şimdi de en iyi sütunun seçilmesini yapan fonksiyonu yazalım:

```
def find_best_feature(dataset):
    nfeatures = len(dataset[0]) - 1
    parent_entropy = calc_shannon_entropy(dataset)
    max_gain = float('-inf')
    max_feature = -1
    for i in range(nfeatures):
        feature_list = [row[i] for row in dataset]
        uniques = set(feature_list)
```

```

sub_entropy = 0
for value in uniques:
    sub_dataset = split_dataset(dataset, i, value)
    sub_entropy += calc_shannon_entropy(sub_dataset)
gain = parent_entropy - sub_entropy
if gain > max_gain:
    max_gain = gain
    max_feature = i

return max_feature, max_gain

dataset = [['a', 'x', 'yes'], ['b', 'x', 'yes'], ['a', 'y', 'yes'], ['a', 'x', 'no'], ['a', 'y', 'yes']]

```

result = find_best_feature(dataset)
print(result)

Şimdi de yukarıdaki örneği scikit-learn ile çözelim. scikit-learn aslında sütunların kategorik olup olmadığını bilmez. Her şeyi sanki sayısal bilgiymiş gibi yapar. Bizim kategorik alanları sayısal biçimde dönüştürmemiz gereklidir. (Bunun için LabelEncoder sınıfının kullanıldığını anımsayınız.)

```

dataset = [['a', 'x', 'yes'], ['b', 'x', 'yes'], ['a', 'y', 'yes'], ['a', 'x', 'no'], ['a', 'y', 'yes']]

result = find_best_feature(dataset)
print(result)

import numpy as np
from sklearn.tree import DecisionTreeClassifier

training_dataset_x = np.array([[0, 0], [1, 0], [0, 1], [0, 0], [0, 1]])
training_dataset_y = np.array([1, 1, 1, 0, 1])

dt = DecisionTreeClassifier(criterion='entropy')
dt.fit(training_dataset_x, training_dataset_y)

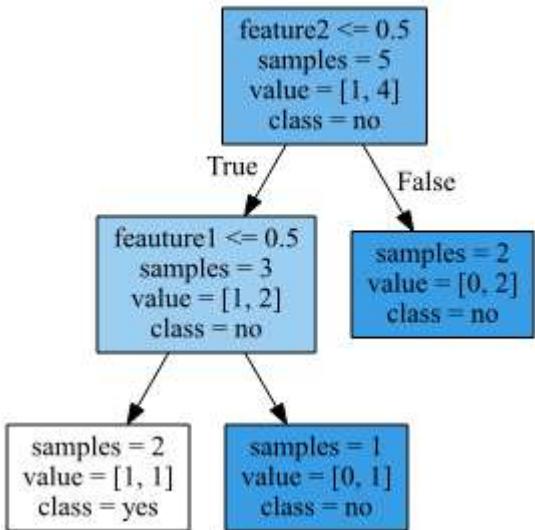
from sklearn.tree import export_graphviz
import graphviz

export_graphviz(dt, out_file='dt.gv', class_names=['yes', 'no'], feature_names=['feature1', 'feature2'], impurity=False, filled=True)

g = graphviz.Source.from_file('dt.gv')

```

Elde edilen grafik şöyledir:



Sınıflandırma İşlemlerinde Lojistik Regresyon İle Karar Ağaçlarının Karşılaştırılması

Lojistik regresyon ile karar ağaçları toplamda birbirlerine yakın performans göstermektedir. Ancak aralarındaki belirgin farklılıklar şöyle özetlenebilir:

- Karar ağaçları insan düşüncesine daha yakındır. Dolayısıyla verilen kararın izlediği yol sözcüklerle daha iyi açıklanabilmektedir.
- Lojistik regresyon parametrik, karar ağaçları ise parametrik olmayan bir modele sahiptir.
- Lojistik regresyonda verilerin dağılımı önemli olabilmektedir. Halbuki karar ağaçlarında dağılımin önemi yoktur.
- Lojistik regresyonda üç değerler sonucu karar ağaçlarına göre daha olumsuz etkilemektedir.
- Lojistik regresyon özellikle sürekli değişkenler üzerinde uygulanmaktadır. Değişkenler kesikli ise karar ağaçları daha iyi performans göstermektedir.
- Genel olarak karar ağaçlarının overfit durumuna yatkınlığı lojistik regresyondan fazla olma eğilimindedir. Yani karar ağaçlarının eğitimde uygulanan kümeye bağlı olarak sınıflandırmayı yanlış öğrenme potansiyeli lojistik regresyona göre daha fazladır.

Karar Ağaçları İle Lojistik Olmayan Regresyon İşlemleri

Karar ağaçları yalnızca lojistik regresyon (sınıflandırma) amacıyla değil lojistik olmayan regresyon amacıyla da kullanılabilmektedir. Kullanım mantığı çok benzerdir. Burada biz scikit-learn ile lojistik olmayan karar ağaç regresyonlarının nasıl olduğu yapılaceğü üzerinde duracağız. Karar ağaçlarının lojistik olmayan regresyon amaçlı kullanılması için scikit-learn içerisindeki DecisionTreeRegressor sınıfı kullanılmaktadır. Sınıfın `__init__` metodunun parametrik yapısı şöyledir:

```
sklearn.tree.DecisionTreeRegressor(criterion='mse', splitter='best', max_depth=None,
min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None,
random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None,
presort='deprecated', ccp_alpha=0.0)
```

Buradaki criterion 'mse', 'friedman_mse' ya da 'mae' olabilir. Yine max_depth parametresi ağacın maksimum derinliğini ayarlamakta kullanılır. DecisionTreeRegressor nesnesi yaratıldiktan sonra yine fit metoduyla eğitim yapılpredict metodu ile tahminleme yapılmaktadır. Aşağıda Boston ev fiyatlarıyla ilgili bir karar ağaç regresyonu görülmektedir:

```

import numpy as np

from sklearn.datasets import load_boston

boston = load_boston()
dataset_x = boston.data
dataset_y = boston.target

from sklearn.tree import DecisionTreeRegressor

dtr = DecisionTreeRegressor()
dtr.fit(dataset_x, dataset_y)

predict_x = np.array([[3.2370e-02, 0.0000e+00, 2.1800e+00, 0.0000e+00, 5.8800e-01,
                      6.9980e+00, 5.5800e+01, 6.0622e+00, 3.0000e+00, 2.2200e+02,
                      1.8700e+01, 3.9463e+02, 2.9400e+00]]))

result = dtr.predict(predict_x)
print(result)

from sklearn.tree import export_graphviz
import graphviz

export_graphviz(dtr, out_file='dtr.gv', impurity=False, filled=True)

g = graphviz.Source.from_file('dtr.gv')

```

Karar ağaçlarıyla regresyon işlemi aslında sınıflandırma işlemine çok benzemektedir. Ağaçtan elde edilen yapraklar eğitim veri kümesindeki değerler olmaktadır. Tabii birden fazla aynı değere ilişkin yaprak karar ağacında bulunabilmektedir. Şüphesiz bu yöntemde fazla sayıda veri ile eğitim uygulamak kestirimi iyileştirecektir. Mademki bu yöntemde yapraklar aslında gerçek sonuç gözlem değerleridir o halde bu değerlerin dışında bir sonuç elde edilmesi beklenmemelidir. Yani eğitim veri kümesinde olmayan sonuçların tahminlemesi bu yöntemde mümkün değildir. Karar ağaçlarıyla lojistik olmayan regresyonu n eğitimdeki veri kğmesi kadar sınıfa sahip lojistik regresyon gibi düşünebilirsiniz.

Karar Ağaçlarında Paketleme (Bagging)

Yukarıda da belirtildiği gibi karar ağaçlarının en önemli sorunu "overfit" durumuna yatkınlıktır. Overfit aslında yüksek varsayıyla kendini göstermektedir. Örneğin biz eğitim veri kümesini iki yarıya bölüp iki karar ağıacı oluşturursak bu ağaçlar bu iki kümenin o andaki durumuna bağlı olarak farklı olabilecektir. Bu durum yüksek varyansı anlatmaktadır. Oysa lojistik regresyonlarda karar ağaçlarına göre varyans düşüktür. Ancak lojistik regresyonlar da üç değerlerden kötü bir biçimde etkilenme eğilimindedir. O halde karar ağaçlarını iyileştirmek için yapılacak şey varyansı düşürmektir. Varyansı düşürmenin ilk akla gelen makul yöntemi ise tek bir ağaç değil birden fazla ağaç kullanmaktır. (Birden fazla ağaca bu terminolojide "orman (forest)" da denilmektedir.)

Paketleme işleminde eğitim veri kümesi içerisindeki belli miktarda satırlardan ağaçlar oluşturulmaktadır. Örneğin eğitim veri kümesinde 100 tane satır olsun. Biz 30'ar satırdan oluşan 500 tane örnek elde edip 500 tane karar ağıacı oluşturabiliriz. Tabii örnekler (samples) rastgele bir biçimde seçilmelidir. Örneklerde genellikle iadelî seçim uygulanmaktadır. (Yani böylece bir örneklem aynı elemanlara sahip olabilmektedir.) Paketleme yönteminde satırlardan örnekler oluşturulmakla birlikte sütunların (yani değişkenlerin) hepsi alınmaktadır.

Paketleme işlemi sonucunda toplam n tane karar ağıacı elde edilmektedir. Pekiyi bu ağaçlardan nasıl faydalılaşacaktır? İşte varyansı düşürebilmek amacıyla buradan bir ortalama değer hesaplanır. Ortalama değer sınıflandırma problemleri için en yüksek sınıf temelinde yapılmaktadır. Lojistik olmayan regresyon problemlerinde ise gerçekten bir aritmetik ortalama hesaplanabilmektedir. Böylece lojistik olmayan regresyon problemlerinde eğitim veri kümesinde olmayan değerler de elde edilebilmektedir.

Örneğin sınıflandırma problemi için 500 tane karar ağacının oluşturulduğu düşünelim. Sonra da kestirim (prediction) işlemi yapmak isteyelim. İşte yöntemde bu kestirilecek veri tüm bu 500 ağaca da sokulmaktadır. Bu 500 ağacın sonucunda elde edilen değerlere bakılarak en yaygın sınıf kestirimini sonucu olarak belirlenmektedir. Yine lojistik olmayan bir regresyon problemi için 500 karar ağacının oluşturulduğunu düşünelim. Kestirim yapıılırken kestirilecek değer bu 500 karar ağacına da sokulacak ve elde edilen değerlerin ortalaması regresyon sonucu olarak belirlenecektir.

Şüphesiz paketleme yöntemi normal karar ağacı yöntemine göre çok daha fazla bilgisayar zamanının kullanılmasına yol açmaktadır. Ancak bu yöntemin performansı normal karar ağacı yöntemine göre daha iyidir.

Pekiyi bu yöntemde eğitim veri kümesindeki toplam satır sayısı n olmak üzere kaçarlı kaç ağaç (yani alt veri kümesi) oluşturulmalıdır? Şüphesiz ağaç sayısı ne kadar fazla olursa o kadar iyidir. Ancak belirli sayıda ağaçtan sonra bilgisayar zamanı artmasına karşın iyileşme oranı da azalmaktadır. Örneğin ağaç sayısı için 100, 200, 300 gibi değerler kullanılabilir. Örneklem için kullanılacak satır sayıları eğitim veri kümesinin miktarına da bağlı olarak belirlenebilmektedir. Yüksek miktardaki eğitim veri kümesi için örneklem veri miktarı sınıflandırma problemlerinde \sqrt{n} , regresyon problemlerinde $n / 2$ ya da $n / 3$ gibi değerde tutulabilir. Fakat düşük miktarda eğitim veri kümesinde örneklem büyülüğu daha yüksek olabilir.

Scikit-learn kütüphanesinde sklearn.ensemble modülündeki BaggingClassifier sınıfı paketleme yöntemi ile karar ağacı sınıflandırması yapmakta kullanılır. Benzer biçimde aynı modüldeki BaggingRegressor sınıfı da paketleme yöntemiyle lojistik olmayan karar ağacı regresyonu için kullanılmaktadır. BaggingClassifier sınıfının `__init__` metodunun parametrik yapısı şöyledir:

```
class sklearn.ensemble.BaggingClassifier(base_estimator=None, n_estimators=10, max_samples=1.0, max_features=1.0, bootstrap=True, bootstrap_features=False, oob_score=False, warm_start=False, n_jobs=None, random_state=None, verbose=0)
```

Fonksiyonun birinci parametresi olan `base_estimator` kullanılacak yöntemi belirtmektedir. Bu parametre için argüman girilmezse default durum karar ağacılarının kullanılmasıdır. Fonksiyonun `n_estimators` parametresi toplamda oluşturulacak karar ağacılarının sayısını belirtir. Fonksiyonun `max_samples` parametresi tamsayı olarak girilirse örneklem büyülüğini float olarak girilse örneklem oranını belirtmektedir. `bootstrap` parametresi ise örneklemi iadelı mi iadesiz mi yapılacağını belirtmektedir. Default durum iadelı örneklemidir.

BaggingClassifier sınıfının `fit` ve `predict` metotları diğer sınıflarda olduğu gibidir. Şimdi BaggingClassifier sınıfını ile meme kanseri veri ile kullanalım.

```
from sklearn.datasets import load_breast_cancer

bc = load_breast_cancer()
print('Sütun isimleri: {}'.format(bc.feature_names))
print('Sınıf Isimleri: {}'.format(bc.target_names))

dataset_x = bc.data
dataset_y = bc.target

from sklearn.model_selection import train_test_split

training_dataset_x, test_dataset_x, training_dataset_y, test_dataset_y =
train_test_split(dataset_x, dataset_y, test_size=0.3)

from sklearn.ensemble import BaggingClassifier

dt = BaggingClassifier(n_estimators=300, max_samples=0.4)
dt.fit(training_dataset_x, training_dataset_y)
score = dt.score(test_dataset_x, test_dataset_y)
print(score)
```

Buradaki sonuçlar önceki normal karar ağacından elde ettiğimiz sonuçlardan genel olarak %3 civarında daha iyidir.

Paketleme ile lojistik olmayan regresyon işlemi de yapılabilmektedir. Bunun için ise BaggingRegressor sınıfı kullanılmaktadır. Aşağıda Boston verileriyle böyle bir lojistik olmayan regresyon örneği görüyorsunuz:

```
import numpy as np

from sklearn.datasets import load_boston
from sklearn.ensemble import BaggingRegressor

boston = load_boston()
dtr = BaggingRegressor(n_estimators=300, max_samples=0.4)
dtr.fit(boston.data, boston.target)

predict_x = np.array([[3.2370e-02, 0.0000e+00, 2.1800e+00, 0.0000e+00, 5.8800e-01,
6.9980e+00, 5.5800e+01, 6.0622e+00, 3.0000e+00, 2.2200e+02,
1.8700e+01, 3.9463e+02, 2.9400e+00]]))

result = dtr.predict(predict_x)
print(result)
```

RASSAL ORMANLAR (RANDOM FOREST)

Aslında rassal orman denilen yöntem paketleme (bagging) yönteminin bir ileri aşamasıdır. Anımsanacağı gibi paketleme yönteminde rastgele seçilen belli miktarda satırlardan karar ağaçları oluşturuluyordu. Ancak bu ağaçlarda sütunların hepsi kullanılıyordu. İşte paketleme yönteminde tüm sütunların (yani değişkenlerin) ağaç'a dahil edilmesi oluşturulan farklı ağaçların birbirlerine benzemesine yol açabilmektedir. Bu da varyansı yükseltten bir faktör durumuna gelebilmektedir. Ayrıca paketleme yönteminde tüm sütunların işleme sokulması ağırlığı yüksek sütunların tüm ağaçlarda etkili olmasına yol açmaktadır. İşte rassal orman yönteminde yalnızca eğitim veri kümelerindeki satırlar değil aynı zamanda sütunlar da örneklenmektedir. Bu nedenle rassal orman yöntemi paketleme yöntemine göre daha üstün kabul edilmektedir.

Rassal orman yöntemi için scikit-learn kütüphanesindeki ensemble modülünde RandomForestClassifier ve RandomForestRegressor isimli sınıflar bulunmaktadır. RandomForestClassifier sınıfının `__init__` metodunun parametrik yapısı şöyledir:

```
class sklearn.ensemble.RandomForestClassifier(n_estimators=100, criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False, class_weight=None, ccp_alpha=0.0, max_samples=None)
```

Yine fonksiyonun `n_estimator` parametresi toplam ağaç sayısını belirtir. `criterion` parametresi karar ağaçları için kullanılacak yöntemi belirtmektedir. Bu parametre 'gini', ya da 'entropy' biçiminde girilebilmektedir. Fonksiyonun pek çok parametresi ağaçın üretiminin durdurulması ile ilgilidir. Fonksiyonun `max_features` parametresi int, float ya da str türünden değer alabilmektedir. int değer doğrudan rastgele alınacak sütun sayısını belirtir. Float değer yine oran belirtmektedir. Eğer bu parametre yazışal biçimde 'sqrt' olarak girilirse sütun sayısı toplam sütunların karekökü kadar alınmaktadır. Zaten bu parametre için argüman girilmezse default olarak 'auto' alındığına dikkat ediniz. 'auto' taamen 'sqrt' ile aynı anlama gelmektedir.

RandomForestClassifier kullanımına şöyle bir örnek verebiliriz:

```
from sklearn.datasets import load_breast_cancer

bc = load_breast_cancer()
print('Sütun isimleri: {}'.format(bc.feature_names))
print('Sınıf İsimleri: {}'.format(bc.target_names))

dataset_x = bc.data
dataset_y = bc.target

from sklearn.model_selection import train_test_split

training_dataset_x, test_dataset_x, training_dataset_y, test_dataset_y =
train_test_split(dataset_x, dataset_y, test_size=0.3)

from sklearn.ensemble import RandomForestClassifier

dt = RandomForestClassifier(n_estimators=300, max_features=0.6)
dt.fit(training_dataset_x, training_dataset_y)
score = dt.score(test_dataset_x, test_dataset_y)
print(score)
```

Benzer biçimde rassal ormanlarla regresyon uygulaması da yapılmaktadır. Genel prensip sınıflandırma problemleriyle aynıdır. Bunun için `sklearn.ensemble` modülündeki `RandomForestRegressor` sınıfı kullanılmaktadır. `RandomForestRegressor` sınıfının `__init__` metodunun parametrik yapısı şöyledir:

```
class sklearn.ensemble.RandomForestRegressor(n_estimators=100, criterion='mse', max_depth=None,
min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto',
max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, bootstrap=True, oob_score=False,
n_jobs=None, random_state=None, verbose=0, warm_start=False, ccp_alpha=0.0, max_samples=None)
```

Aşağıda RandomForestRegressor sınıfı ile Boston örneği verilmiştir:

```
import numpy as np

from sklearn.datasets import load_boston
from sklearn.ensemble import RandomForestRegressor

boston = load_boston()
rfr = RandomForestRegressor(n_estimators=300, max_features=0.6)
rfr.fit(boston.data, boston.target)

predict_x = np.array([[3.2370e-02, 0.0000e+00, 2.1800e+00, 0.0000e+00, 5.8800e-01,
6.9980e+00, 5.5800e+01, 6.0622e+00, 3.0000e+00, 2.2200e+02,
1.8700e+01, 3.9463e+02, 2.9400e+00]])]

result = rfr.predict(predict_x)
print(result)
```

PEKİŞTİRMELİ ÖĞRENME (REINFORCEMENT LEARNING)

Yapay zekadaki öğrenme (reinforcement learning) psikolojideki edimsel koşullanma (operant conditioning) kuramı temel alınarak oluşturulmuştur. Psikolojide edimsel koşullanma "ödül getiren davranışların tekrarlandığı" fikrine dayanır. Bu konudaki ilk çalışmalar Edward Thorndike tarafından yapılmıştır. Thorndike kedileri belli bir alana kapatıp belli bir düğmeye bastıklarında onların çıkışmasına olanak sağlayan bir düzenek oluşturmuştur. Sonra kediler bu düğmeye tesadüfen bastıklarında ödül olarak dışarı çıkmışlardır. Thorndike bu deneyi aynı kediler üzerinde tekrarladıkça kediler düğmeye basma işini çok daha hızlı yapmaya başlamıştır. Thorndike buna "etki yasası (law of effect)" demiştir. Fakat edimsel koşullamayı kuramsallaştıran asıl kişi B.F. Skinner'dır. Skinner Thorndike'ın deneylerini genişleterek bu alanı çok daha kapsamlı hale getirmiştir. İşte "pekiştirme (reinforcement)" terimi de Skinner tarafından uydurulmuştur. Skinner organizmada hız uyandıran edimlerin (eylemlerin) yinelendiğini pek çok deneyle göstermiştir. Buna göre organizma birtakım faaliyetlerde bulunur. Bazı faaliyetleronda hız uyandırmaktadır. Başka bir deyişle organizma bazı faaliyetlerden bazı ödüller (rewards) elde etmektedir. Yapılan faaliyetler karşısında ödül elde etme sürecine "pekiştirme (reinforcement)"i burada verilen ödülü de "pekiştireç (reinforcer)" denir. Edimsel koşullanma davranış değişirmede ve davranışın şekillendirilmesinde en etkili yöntemlerden biri olarak kabul edilmektedir.

Edimsel koşullanmada davranışı tekrarlatmak için verilen uyarana pekiştireç (reinforcer) dendiğini belirtmişik. Pekiştireçler de "pozitif" ve "negatif" olmak üzere ikiye ayrılmaktadır. Organizmaya doğrudan hız veren pekiştireçlere pozitif pekiştireçler denir. (Örneğin bir çocuğa ödül olarak verilen şeker gibi, bir çalışانا ödül olarak verilen para gibi.) Ortamda olumsuz bir uyarani (yani organizmada gerginlik yaratılan bir uyarani) ortadan kaldırarak organizmanın yine hız elde etmesi sağlanabilir. İşte bu biçimde ortamdan acı verici bir uyarının kaldırılması durumunda bu uyarana da "negatif pekiştireç" denilmektedir. Hem pozitif hem de negatif pekiştireçler sonuç olarak organizmada hız uyandırmaktadır. Edimsel koşullanmada önemli bir olgu da "ceza (punishment)"dır. Organizmada hoş gitmeyen bir etki oluşturan uyararlara bu bağlamda "ceza" denilmektedir. Ceza da bir pekiştireçtir. Ancak araştırmalar davranış değiştirmede ve davranışın kalıcılığını sağlamada cezaların çok iyi iş görmediğini göstermektedir. Ceza çabuk etki göstermesi bakımından pratik bir yöntemdir ancak genellikle kalıcı bir davranış değişikliğine yol açmaz ve saldırganlığı artırmak gibi dolaylı yan etikileri vardır. Cezaalar da "birinci" ve "ikincil" olmak üzere ikiye ayrılmaktadır. Organizamada doğrudan acı veren uyarının uygulanmasına birincil ceza, organizmada hız uyandıran bir uyarının ortadan kaldırılması biçiminde verilen cezaya da ikincil ceza verilmektedir. Ceza verilecekse ikincil cezalar tercih edilmelidir.

Pekiştirme sürecinin diğer bir türü de "dolaylı pekiştirme (vicarious reinforcement)" denilen biçimdir. Dolaylı pekiştirme başlarını izleyerek ve başlarının ödül aldığı ya da ceza aldığına görerek bundan bir sonuç çıkartma sürecidir. Tabii burada bazı akıl yürütme faaliyetleri de deveye girdiği için sürecin bilişsel tarafı da vardır. Bu nedenle bu öğrenme modeline "sosyal öğrenme" ya da "sosyal bilişsel öğrenme" denilmektedir.

Pekiştirmeli Makine Öğrenmesinin Ana Fikri ve Temelleri

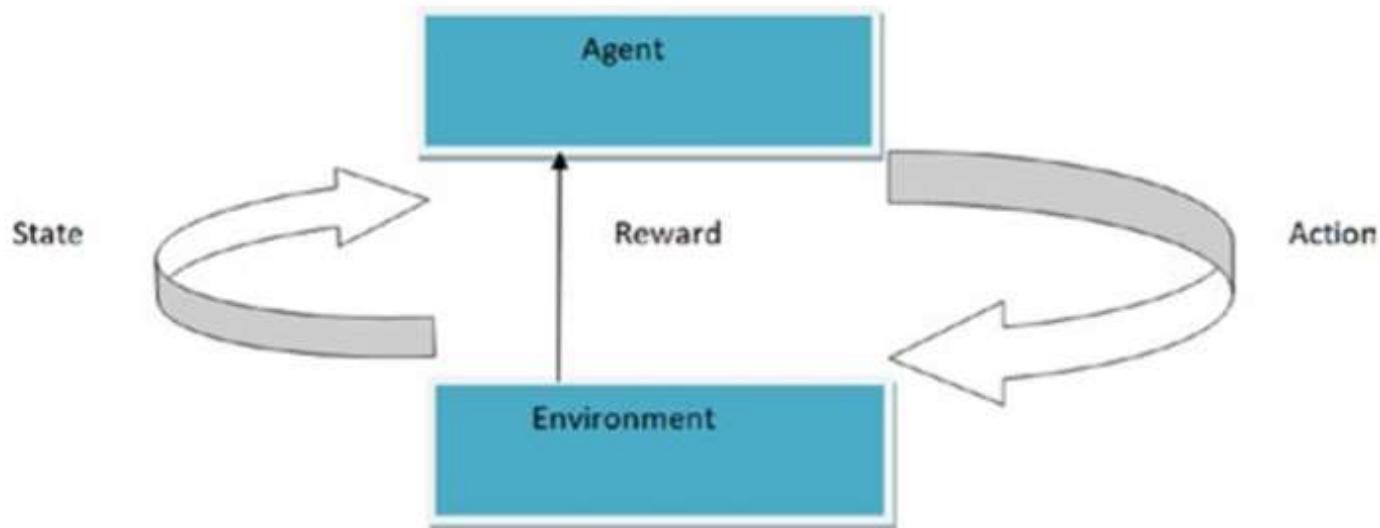
Pekiştirmeli öğrenmede ana fikir belli bir edimin (action) çevre (environment) ile etkileşime sokulması bunun sonucunda bir ödül ceza sisteminin uygulanması ve gitgide bu ödüllerin maksimize edilmeye çalışılmasıdır. Pekiştirmeli makine öğrenmesi sürecinde bazı terimler sıkça kullanılmaktadır:

- Yazılımsal Etmen (Software Agent): Pekiştirmeli öğrenmeyi gerçekleştiren yazılım ve algoritmalarla "yazılımsal etmen" denilmektedir. Genellikle "yazılımsal etmen" yerine yalnızca "etmen" terimi kullanılmaktadır. Biz de burada "yazılımsal etmen" yerine "etmen" terimini kullanacağız.
- Çevre (Environment): Etmenin faaliyette bulunduğu ortama denilmektedir. Pekiştirmeli öğrenmede çevreyi oluşturmak başka bir deyişle simüle etmek önemli bir problemdir. Bunun için hazır platformlardan faydalananmaktadır.
- Faaliyet (Action) Yazılımsal etmenin yaptığı edimlere faaliyet (action) denilmektedir.

- Durum (Status): Durum yazılımsal etmenin belli bir andaki verisel özellikleridir. Yazılımsal etmen bir faaliyette bulunduğu zaman durum değişikliği oluşur. İşte yazılımsal etmenin bir durumdan diğerine geçmesi söz konusudur. Bu haliyle konu "automata" teorisile ve durum makineleriyle (state machines) ilgili olmaktadır.

- Ödül / Ceza (Reward /Punishment): Yazılımsal etmen bir faaliyette bulunduğuanda algoritmik olarak ona bir ödül ya da ceza verilmektedir. Şüphesiz bu kavram pekiştirmeli öğrenmenin en önemli prensibini oluşturmaktadır.

Bu durumda yukarıdaki kavamlarla tipik bir pekiştirmeli öğrenme modeli aşağıdaki gibi bir şekilde betimlenebilir:



(Şekil "Reinforcement Learning With Open AI, TensorFlow and Keras Using Python" kitabından alınmıştır.

Bu şekilde anlatılmak istenen "etmenin çevre ile etkileşiminde bir faaliyette bulunduğu, bu faaliyet sonucunda bir durum değişikliğinin olduğu ve yeni durum için bir ödül/ceza sisteminin devreye sokulduğu"dur.

Etmenin içinde bulunduğu çevrenin özellikleri aşağıdakilere ilişkin olabilir:

- Deterministik Olan ya da Deterministik Olmayan Çevre: Eğer bir durumda gidilebilecek tek bir durumsal seçenek varsa ya da birden fazla durumsal seçeneklerin hangi seçileceği eldeki bilgilerle belirlenebiliyorsa bu tür çevrelere deterministik çevre denilmektedir. Deterministik çevrelerde aynı durumlardan aynı sonuçlar elde edilmektedir.

Ayırık ya da Sürekli Çevre: Ayırık çevreden kastedilen belli bir durumda olabilecek seçeneklerin sınırlı sayıda olmasıdır. Yani etmen belli bir durumda sınırlı sayıda faaliyette (action) bulunabilir. Bu matematikteki "ayırık değişken" kavramından biraz farklıdır. Sürekli çevrede ise bir durumda sonsuz miktarda olası faaliyet (action) söz konusu olmaktadır. Örneğin bir labirentte belli bir noktada sapılacak yolların sayısı sınırlıdır. Bu labirent ayırık bir çevre oluşturmaktadır.

Gözlemlenebilme (Observability) Derecesine Göre Çevreler: Gözlemlenebilir çevre demekle çevrenin bütünsel durumunun bilinmesi anlatılmak istenmektedir. Yani biz çevredeki tüm öğelerin yerlerini, değerlerini vs. biliyorsak bu gözlemlenebilir bir çevredir. Eğer biz çevreyi oluşturan öğelerin durumlarını tam olarak bilmiyorsak bu çevre gözlemlenebilir değildir. Tabii gözlemlenebilirlik bir derece iştir. Yani bir çevrede bazı öğeler gözlemlenebilirken bazıları gözlemlenebilir olmayıabilir.

Pekiştirmeli Öğrenmenin Uygulama Alanları

Pekiştirmeli öğrenmenin en önemli uygulama alanları şunlardır:

- Otomatik Kontrol Sistemleri: Otomatik kontrol sistemleri çevredeki değişimlere göre uygun konum alabilen sistemlerdir. Pekiştirmeli öğrenmenin tipik uygulama alanlarından birini oluşturmaktadır. Örneğin adaptif trafik

kontrollerinde kırmızı ışığın ne süreyle yanıp söndürüleceği bu biçimde sisteme öğretilebilir. Benzer biçimde havadaki nem, basınç gibi faktörlere bağlı olarak birtakım denegelme işlemlerinin yapılması da otomatik kontrol sistemlerine örnek olarak verilebilir.

- Finansal Problemler: Finansal süreçler tipik olarak birtakım alternatifler arasında uygun bir alternatifin belirlenmesi ile ilgilidir. Burada pekiştirmeli öğrenme "öğrenilecek şeyin net olarak bilinmediği" durumlarda kullanılabilketedir.

- Üretim Problemleri: Üretimdeki bazı problemler faaliyet ve durum değişikliği ile ilişkilidir. Bu tür durumlarda pekiştirmeli öğrenmeler kullanılabilir.

- Stok Yönetimi Problemleri: Optimal stok miktarlarının belirlenmesi üretimde önemli bir sorundur. Bu tür sistemler pek çok faktörü olan bu sorun için iyi bir seçenek sunabilmektedir.

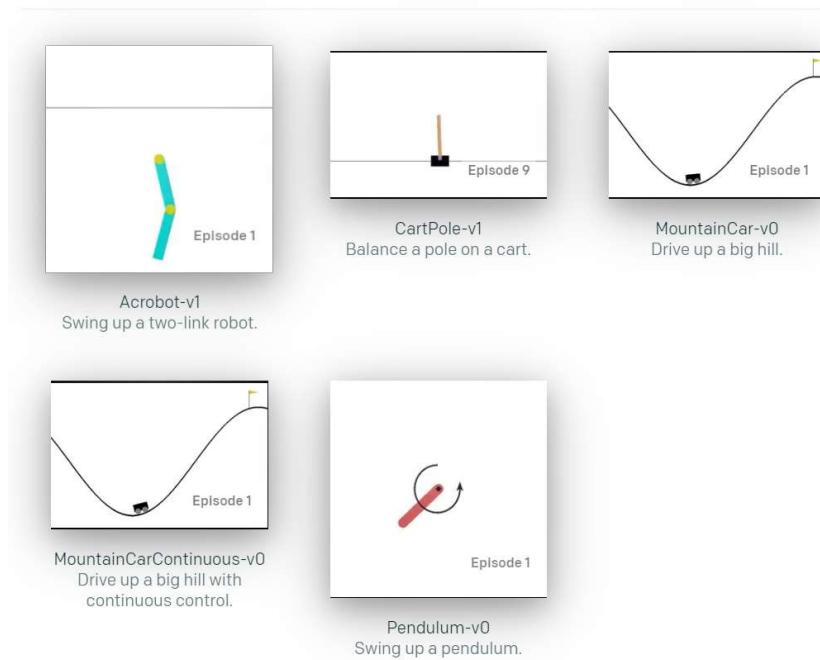
- Dağıtım Problemleri: Genel olarak bir grup ürünün ne zaman ve hangi rotayla dağıtılacagına ilişkin problemlerde de pekiştirmeli öğrenme sıkça kullanılmaktadır.

OpenAI GYM Ortamı

OpenAI yapay zeka konusunda faaliyet gösteren bir kurumdur. OpenAI'nın gym denilen ortamı (gym.openai.com) pekiştirmeli öğrenme için iyi bir simülasyon ortamı sunmaktadır. Tabii gym dışında başka ortamlar da mevcuttur. Örneğin benzer özellikler Google'ın DeepMind ortamında da bulunmaktadır. Biz kursumuzda OpenAI'nın Gym ortamını kullanacağımız.

OpenAI Gym ortamında çeşitli konulara ayrılmış çeşitli simülatörler vardır. Bu simülatörlerde pek çok şey hazırlıdır. Bütün bunlar aslında kullanıcıların algoritma geliştirmeleri için hazırlanmıştır. GYM ortamındaki simülatörler çeşitli kategorilere ayrılmıştır. Örneğin "Classical Control" kategorisindeki simülatörler şunlardır:

Classic control
Control theory problems from the classic RL literature.



Ortamı kurmak için aşağıdaki komut uygulanabilir:

```
pip install gym
```

Gym CartPole Simülatörü

Cartpole ("Cart" araba, "pole" ise direk anlamına gelmektedir) simülatöründe bir küçük arabanın tepesine bir direk yerleştirilmiştir. Bu direk arabaya katı biçimde monte edilmemiştir ve mafsallardan oynayabilmektedir. Araba hareket ederse bu direk sola ya da sağa düşer. Bu simülatörde kullanıcı arabaya soldan ya da sağdan bir kuvvet uygular. (Kullanıcı arabanın hızını doğrudan belirlemez. Yalnızca kuvvet uygular.) Uygulanan kuvvet sabittir. Eğer bu kuvvet aynı biçimde uygulanmaya devam ederse şüphesiz araba ivmeli bir biçimde yanı gittikçe hızlanarak hareket edecektir. Oyundan amaç arabaya monte edilmiş direğin devrilmeden +15, -15 derece arasında dik tutulmasıdır. Aynı zamanda arabanın merkezden 2.4 birim uzaklaşmaması gerekmektedir. Yani burada başarısızlığın iki nedeni vardır: Direğin +15, -15'lik dereceden aşağıda olması ve arabanın merkezden 2.4 birim uzaklaşmış olmasıdır. Özette modelin parametrik bilgileri şöyledir:

Faaliyetler (actions): İki faaliyet (action) tanımlıdır: Sola ya da sağa kuvvet uygulamak.

Gözlem Uzayı (Observation Space): Her durumda (state) ortamdan elde edilen gözlem bilgileri şunlardır:

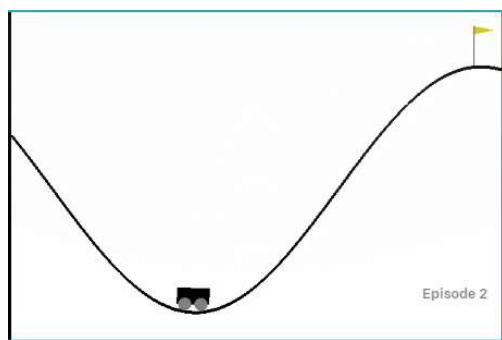
- Arabanın konumu (merkez 0 olmak üzere)
- Arabanın o andaki hızı
- Direğin açısı
- Direğin açısal hızı

Durumlar (States): Buradaki durumlar zamansal biçimde birbirlerinden ayrılmıştır. Zaman sürekli bir olgudur. Ancak burada belli zaman aralıklarındaki durumlar ele alınmaktadır. Yani bir durumdan (state) diğerine geçiş bir t zamanı sonra olmaktadır. Başka bir deyişle belli bir zaman aralığı sonucunda ortamda değişim yeni bir duruma karşı gelmektedir.

Ödül Sistemi: Bu simülatörde ödül sistemi de simülatörün kendisi tarafından verilmektedir. Tabii programcı başka ödül sistemlerini de kullanabilir. Ancak genel olarak ödül değeri direk -15, +15 açısı arasında ise +1 olarak değilse 0 olarak verilmektedir.

Gym MountainCar Simülatörü

MountainCar simülatöründe bir araba bir tepeyi aşmaya çalışmaktadır. Arabayı idare edenin yapabileceği üç şey vardır (action space): İleri doğru gaz vermek, geriye doğru gaz vermek ve hiç gaz vermemek.

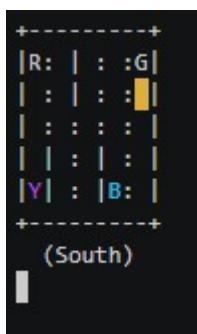


Araba ileri doğru gaz verildiğinde yokuştan dolayı tepeyi aşamaz. Tepe aşması için hızının artırılması gerekmektedir. Burada insanların (makinelerin değil) aklına gelen en uygun strateji önce ileri doğru gaz verip arabanın çıkışını sağlama, sonra geriye gaz verip ters yönde potansiyel enerji kazanmak ve böyle devam ederek tepeyi aşmasını sağlamaktır. Simülatör bize gözlem değeri olarak (observation space) arabanın konumunu ve hızını vermektedir. MountainCar simülatöründe tepe aşılırsa ödül olarak 1 puan verilmekte aşılmazsa 0 puan verilmektedir.

Gym Taxi Simülatörü

Bu simülatörde taksi için dört adet indirme bindirme yeri vardır. Amaç müşteriyi bu noktalardan birinden alıp diğerine bırakmaktır. Taksi eğer engel yoksa sola, sağa, yukarıya, aşağıya gidebilmektedir. Burada yolcu uygun yere bırakılırsa +20 ödül alınmaktadır. Her adımda -1 ceza verilmektedir. (Yolcu alımında herhangi bir puan verilmediği

gibi -1 ceza verilmektedir.) Yolcunun olmadığı yerden yolcu almak ya da hedef olmayan durağa yolcu bırakmak -10 ceza puanına yol açmaktadır. Bu simülörde taksiyi idare edenin yapabileceği 5 eylem vardır: Aşağıya gitmek, yukarıya gitmek, sağa gitmek, sola gitmek, yolcu almak ve yolcu bırakmak.



Taksi simülöründe bir tane gözlem değeri (observation space) vardır. O da taksinin hangi noktada olduğunu belirtir.

FrozenLake Simülörü

Bu simülörde donmuş bir gölde bir kişi bir noktadan (S noktası) başka bir noktaya (G noktası) gitmeye çalışmaktadır. Ancak yolu üzerinde delikler (yani çatılar) vardır (H noktaları). Kişi de ancak donmuş yüzeylerden (F noktaları) yürüyebilmektedir.



Eğer kişi hedef noktaya (G noktası) varırsa 1 puan ödül almaktadır. Ancak hedefe varamadığı her harekette 0 puan alır. Kişinin yapabileceği hareketler (action space): Yukarı gitmek, sağa gitmek, aşağı gitmek ve sola gitmektir.

Gym Simülörlerinin Çalıştırılması

Gym simülörlerinin çalıştırılması için önce modüldeki global make isimli fonksiyonun simülör ismi argüman biçiminde verilerek çağrılmazı gereklidir. Örneğin:

```
import gym  
env = gym.make('CartPole-v0')
```

Bundan sonra simülör nesnenin reset metoduyla başlangıç konumuna araylanır. Örneğin:

```
import gym  
env = gym.make('CartPole-v0')  
obs = env.reset()
```

reset işlemi ile araba ve direk rastgele bir konumdan başlatılmaktadır. reset metodu bize geri dönüş değeri olarak "gözlem nesnesini (observation object)" verir. Gözlem nesnesi ortamdaki gözlemlenebilir tüm öğelerin (observation space) oluşturduğu bir numpy dizisidir. Bu dizinin elemanları "CartPole-v0" için sırasıyla arabanın konumu, arabanın hızı, direğin açısı ve direğin açısal hızıdır. Örneğin:

```
In [22]: obs  
Out[22]: array([-0.04124306, -0.0085189 ,  0.01014184, -0.01133194])
```

reset işleminden sonra artık durumlar arasında geçiş oluşturmak gereklidir. Bunun için de faaliyet (action) lazımdır. İşte bu işlem step metoduyla yapılmaktadır. Simülatör sınıflarının step metotları faaliyet alanındaki (action space) bir faaliyeti argüman olarak alıp yeni oluşan çevre durumuna ilişkin bazı değerleri bize vermektedir. CartPole simülöründe step metodu 0 ve 1 olmak üzere iki değeri argüman olarak almaktadır. 0 sola, 1 sağa kuvvet uygulanacağı anlamına gelmektedir. step metodu dörtlü bir demete geri dönmektedir. Bu dörtlü demetin ilk elemanı o faaliyet uygulandıktan sonraki çevrenin gözlem değerleridir. Bu değerler yine CartPole örneğinde bize 4'lü numpy dizisi biçiminde verilmektedir. Demetin ikinci elemanı önerilen ödül miktarıdır. Üçüncü eleman oyunun bitip bitmediğini belirten bir flag niteliğindedir. Örneğin CartPole simülöründe bitiş koşulları iki tanedir. Son parametre bir sözlük nesnesidir. Bize başka bazı bilgileri vermektedir.

Gym simülörleri oluşan değerleri bize vermenin yanı sıra ortamı da görüntüleyebilmektedir. Bunun için ilgiki simülör nesnesinin render metodu kullanılır. Şimdi örnek olarak arabayı bir sağa bir sola hareket ettirelim:

Örneğin:

```
import gym  
  
env = gym.make('CartPole-v1')  
obs = env.reset()  
  
for i in range(1000):  
    obs, reward, done, info = env.step(0)  
    if done:  
        break  
    env.render()  
    obs, reward, done, info = env.step(1)  
    if done:  
        break  
    env.render()  
  
print(i)  
  
env.close()
```

Burada araba bir sağa bir sola hareket ettirilmiştir. Bitiş koşulları sağlandığında döngüden çıkıştır.

Pekiyi pekiştirmeli öğrenmede bu simülörle programcı ne yapacaktır? İşte bu tür OpenAI simülörlerinde programcı aslında gözlem değerlerini alıp faaliyeti (action) belirleyecek algoritmik yapıyı kurmaya çalışır. Bu simülörlerin temel amacı bu çalışmaların yapılabilmesi için ortam oluşturmaktır.

Genel olarak Gym nesnelerinin action_space elemanın sample metodunu bize faaliyet uzayındaki rastgele bir değeri vermektedir. Örneğin arabayı rasgele biçimde 500 adım şöyle ilerletebiliriz:

```
import gym  
  
env = gym.make('CartPole-v0')  
obs = env.reset()  
  
for i in range(500):  
    action = env.action_space.sample()  
    obs, reward, done, info = env.step(action)  
    env.render()  
  
env.close()
```

Biz bu simülatörleri aslında pekiştirmeli öğrenme algoritmalarını gerçekleştirmek ve denemek için kullanacağımız. Örneğin aşağıda yalnızca direğin açısına göre faaliyeti ayarlayan bir strateji uygulanmıştır. Bu stratejinin başarısı olacağı belliidir:

```
import gym

def get_action(obs, reward):
    return 0 if obs[2] > 0 else 1

env = gym.make('CartPole-v0')
obs = env.reset()

reward = 0
for i in range(500):
    action = get_action(obs, reward)
    obs, reward, done, info = env.step(action)
    env.render()

env.close()
```

Simülatör sınıflarının action_space örnek öznitelikleri faaliyet kümesi hakkında bize bilgi vermektedir. Bu örnek özniteligiden Discrete türünden bir nesne elde edilmişse bu nesne bize faaliyet elemanlarının ayrık değerlerden oluştuğunu söylemektedir. Discrete sınıfının n örnek özniteliği bu değerlerin kaç tane olduğunu verir. Benzer biçimde simülatör sınıflarının oservation_space örnek özniteliği de gözlemlerin özelliklerini bize vermektedir.

Şimdi MountainCar simülatörünü çalıştırıralım. Örneğin bu simülatörden doğrudan arabaya ileriye doğru gaz verelim:

```
import gym

env = gym.make('MountainCar-v0')
obs = env.reset()
print(obs)
while True:
    obs, reward, done, info = env.step(2)
    if done:
        break
    env.render()
    print(obs)

env.close()
```

Göründüğü gibi arabaya sürekli gaz vermek tepeyi aşması için yeterli olmamaktadır. Burada izlenecek en uygun strateji arabanın hızı sıfıra düşene kadar tepe yönünde gaz vermek, hız sıfıra düşünce ters yönde gaz vermek, böylece hız her sıfıra düştüğünde ters yönde gaz vererek tepeyi aşmasını sağlamaktır:

```
import gym

env = gym.make('MountainCar-v0')
obs = env.reset()
print(obs)
while True:
    if obs[1] > 0:
        action = 2
    else:
        action = 0
    obs, reward, done, info = env.step(action)
    if done:
        break
    env.render()
    print(obs)

env.close()
```

Şimdi de Taksi simülöründe taksiye rastgele işlemler yapalım:

```
import gym

env = gym.make('Taxi-v3')
obs = env.reset()
print(obs)
env.render()

for i in range(10):
    action = env.action_space.sample()
    obs, reward, done, info = env.step(action)
    if done:
        break
    env.render()

env.close()
```

Pekiştirmeli Öğrenmede Kullanılan Algoritmalar

Pekiştirmeli öğrenmede pek çok algoritmik yöntemden ve teknikten faydalılmaktadır. Örneğin yapay sinir ağları da başka tekniklerle birlikte pekiştirmeli öğrenmede kullanılabilir. Biz kursumuzun bu bölümünde en yaygın kullanılan algoritmik yöntemler üzerinde duracağız.

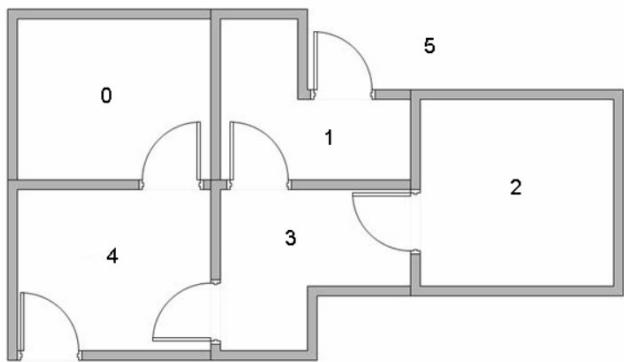
Q-Learning Algoritması

Q-Learning pekiştirmeli öğrenmede en fazla kullanılan algoritmik yöntem ve yaklaşımındır. Bu yaklaşımın sınır ağı versiyonu da vardır. Buna da "Deep Q Learning (DQN)" denilmektedir.

Q Learning algoritması aslında "Markov Zincirleri" denilen algoritmik modelle yakından ilişkilidir. Gerek Markov zincirlerinde gerekse Q-Learning algoritmasında en önemli unsur belli bir noktadaki kararın daha önceki veriler ve durumlar dikkate alınarak verilmesidir.

Q-Learning algoritmasında üç önemli olgu vardır: Durum (State), Eylem (Action) ve Ödül (Reward). Yöntemde öncelikle karşılaşılan problemden durumlar (states) oluşturulmaktadır. Yani içinde bulunulan koşullar durumlara ayırtılmalıdır. Durumlar ayrık olgulardır. Dolayısıyla sürekli olguların ayrık hale getirilmesi gereklidir. Örneğin MountainCar simülasyonunda arabanın içinde bulunduğu durum konum ve hızla belirlenmektedir. Fakat konum da hız da ayrık oldular değildir. O halde bizim bu konum ve hızı aralık temelli bir biçimde ayrık hale getirmemiz gereklidir. Sürekli olduların ayrık hale getirilmesiyle elde edilen durumların sayısı çok yüksek olabilmektedir. Uygulamacı durum sayılarını kendi olanaklarına göre makul bir biçimde belirlemelidir. (Pekiştirmeli öğrenmede oluşturulan durumların sayısı fazla olabileceğinden dolayı bu biçimde yapay öğrenme yöntemleri de yüksek bir bilgisayar zamanı ve kaynağının kullanılmasına yol açmaktadır.) Tabii bazen durumlar zaten ayrıktır. Bu durumda onların ayrık hale getirilmesine gerek kalmaz. Örneğin Taxi simülöründe ya da FrozenLake simülöründe durum zaten ayrıktır.

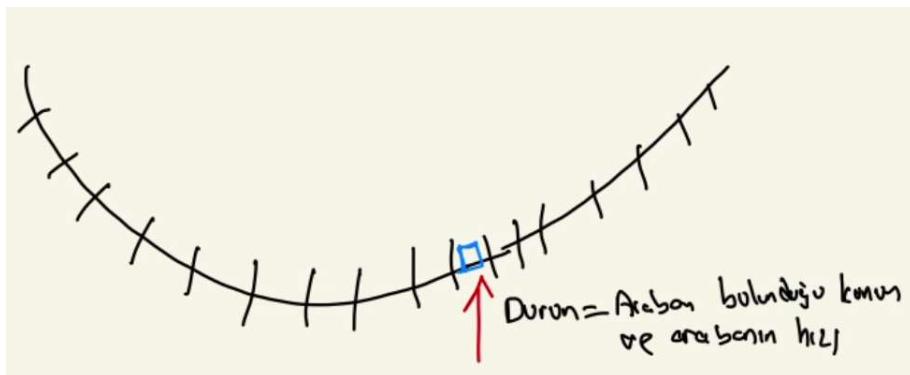
Örneğin bir labirentte odaların her biri bir durum olarak modellenebilir. Böylece labirentte kişinin bulunduğu yer zaten ayrık bir durum oluşturur.



(Şekil <http://mnemstudio.org/path-finding-q-learning-tutorial.htm> sitesinden alınmıştır.)

Burada durumlar 0, 1, 2, 3, 4, 5'ten oluşmaktadır. Bu labirentten amaç 5 noktasına ulaşmaktadır.

Süphesiz durum (state) tek değişkenli bir bilgi olmak zorunda değildir. Çünkü belli bir durumu ifade etmek için birden fazla değişkene ihtiyaç duyulabilmektedir. Örneğin MountainCar simülasyonunda durum yalnızca arabanın x eksenindeki yeri değildir. Arabanın o andaki hızı da durumun bir parçasıdır. Bu örnekte x eksenini 20 parça hızı da 20 parçaaya ayırsak toplamda 20x20'lük bir durum matrisi elde ederiz.



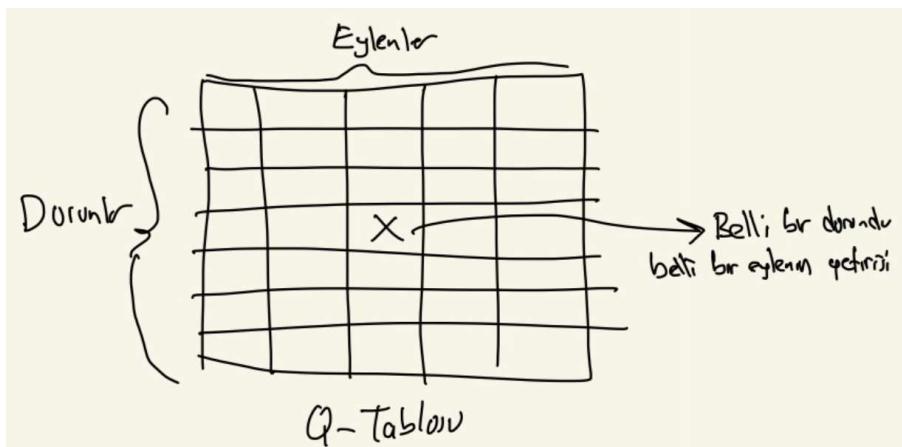
Göründüğü gibi MountainCar simülasyonunda arabanın belli bir konumda olması durum bilgisi için yetmemektedir. Araba aynı konumda bulunduğu halde o sırada farklı hızlarda olabilir. O halde buradaki durumlar konum ve hızı barındırmalıdır. Biz konum ve hızı 20 parçaya bölgerek ayrık hale getirdiğimizde toplam 20x20'lük bir durum matrisi elde ederiz. Tabii eğer problemde durumsal bilgi 2 değişkenli değil 3 değişkenli olsaydı bizim durum matrisimiz de 3 boyutlu olacaktı. O halde genel olarak n değişkenli bir durumsal problemde durum matrisi de n boyutlu olacaktır.

Q-Learning algoritmada ikinci önemli nokta belli bir durumdaki eylemlerin (actions) neler olduğunu söylemektedir. Yani belli bir durumda olan etmen (agent) o durumu değiştirmek için hangi geçerli eylemleri yapabilecektir? Eylemlerin sayısı ve genel olarak listesi çok fazla olabilir. Üstelik de eylemler içinde bulunulan duruma göre farklılaşabilir. Bu durumda programcının eylemleri uygun ve makul bir veri yapısıyla ifade etmesi gereklidir. Yukarıdaki labirent örneğinde eylem bir odadan (yani durumdan) diğerine geçmektir. Ancak her odadan diğerine bir geçişin olmadığı görülmektedir. Tabii bu tür gerçek problemlerde hangi odalardan hangi odalara geçişin olduğu başlangıçta biliniyor olabilir ya da süreç sırasında öğreniliyor olabilir. Yukarıdaki problemde labirentin durumunun önceden bilindiğini varsayırsak hangi durumlarda hangi eylemlerin yapılabileceğini bir sözlük nesnesiyle ya da matrisle temsil edebiliriz.

Pekiştirmeli öğrenmede bir ödül mekanizmasının olması gerekligidenden bahsetmiştik. Çünkü öğrenme aslında ödülü maksimize etmeye çalışmakla gerçekleşmektedir. Ödül mekanizması ortamı iyi bilenler tarafından makul bir biçimde oluşturulabilir. Ancak bazen ortam problem çözücü tarafından da tam olarak bilinmeyecektir. Bu durumda programcı açık birtakım durumlar için ödül koyabilir. Örneğin yukarıdaki labirentte ödül yalnızca labirentten çıkış için veriliyor olabilir. Yani herhangi iki oda arasındaki geçiş eyleminden 0 ödül verilirken bir odadan dışarıya çıkış için örneğin 100'lük ödül verilebilir.

Q Learning algoritmasında ismine "Q-Tablosu (Q-Table)" denilen bir tablo oluşturulmaktadır. Bu tabloda satırlar durumları (states) sütunlar ise eylemleri (actions) belirtir. Q tabloları sözlükler ya da matrisler biçiminde

oluşturulabilmektedir. Çünkü tablodan amaç belli durumda iken hangi eylemlerin uygun olacağına ilişkin bir puanlama oluşturmaktır. Yani algoritmada etmen belli bir durumdayken tabloya başvurarak o durumda hangi eylemin yapılması gerekiğine karar verir. Q-Tablosu başlangıçta 0'larla doldurulur.



Tabii Q-Tablosunu oluşturan durumların da aslında değişken sayısına bağlı olarak matrisel bir biçimde olabileceğine dikkat ediniz. Örneğin MountainCar simülasyonunda durum aslında 20×20 'lik bir matristir. Bu durumda MountainCar simülasyonundaki eylemler de 3 tane olduğuna göre bu simülasyon için oluşturacak Q-Tablosu aslında $20 \times 20 \times 3$ 'luk bir matris durumunda olacaktır. Tabii yukarıda da belirttiğimiz gibi Q-Tablosu bir matris biçiminde değil bir sözlük biçiminde de oluşturulabilir. Örneğin yukarıdaki labirent için Q-Tablosu aşağıdaki gibi bir sözlük biçiminde oluşturulabilir:

```
maze = {0: {4: 0}, 1: {3: 0, 5: 0}, 2: {3: 0}, 3: {1: 0, 2: 0, 4: 0}, 4: {0: 0, 3: 0, 5: 0}, 5: {5: 0}}
```

Burada bir sözlüğün her elemanı sözlük yapılmıştır. Bu tür problemlerde matris yerine sözlüklerin tercih edilmesinin en önemli nedeni her durumda yapılacak eylemlerin farklı olmasıdır.

Aynı labirent için Q tablosu aşağıdaki gibi bir matrisle de oluşturulabilir:

```
maze = np.zeros((6, 6))
```

		Eylemler					
		0	1	2	3	4	5
Durumlar	0	0	0	0	0	0	0
	1	0	0	0	0	0	0
	2	0	0	0	0	0	0
	3	0	0	0	0	0	0
	4	0	0	0	0	0	0
	5	0	0	0	0	0	0

Q Tablosu

Q tablosu öğrenme süreci sırasında her eylemden sonra güncellenen bir tablodur. Öğrenme gerçekleştikçe tablo gitgide güncellenir. Öğrenme süreci bittiğinde tablo hangi durumlarda hangi eylemlerin en yararlı olduğunu belirten bir yapıya kavuşur.

Q-Learning algoritmasının incelikleri yavaş yavaş ele alınacaktır. Ancak algoritma baştan rastgele bir durumdan başlatılır. İşin başında mademki etmen (agent) hiçbir şey bilmemektedir. O halde rastgele bir eylemde bulunur. İşte etmen belli s durumunda belli bir a hareketini yarlığında matrisin $[s, a]$ elemanı bir formülle güncellenmektedir. Böylece bir durumda bir eylem yapıldığında gittikçe tablo elemanları güncellenmiş olacaktır. Bu eylemler arttıkça tablo elemanları daha uygun değerlerle doldurulur. En sonunda Q-Tablosu tatminkar değerlerle doldurulduğunda artık

artık etmen belli bir durumda bırakıldığında ödülü maksimize etmek için ne yapması gerektiğini öğrenmiş olacaktır. Burada şüphesiz iki nokta önemlidir:

- 1) Tablo elemanı hangi formüle göre güncellenmektedir?
- 2) Belli bir durumda etmen hangi eylemi gerçekleştirecektir?

Tablo elemanlarının güncellenmesi aşağıdaki formüle göre yapılmaktadır:

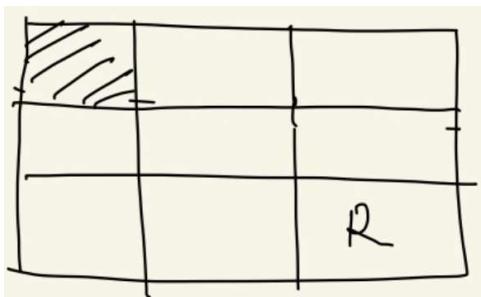
$$Q(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \left(\underbrace{r_{t+1}}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\substack{\text{learned value} \\ \text{estimate of optimal future value}}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \right)$$

Bu formülde iki önemli sabit vardır: Alfa (learning rate) ve gama (discount). Bu faktörler 0 ile 1 arasında bir değer olarak alınabilmektedir ve daha sonra bunların etkisinden bahsedilecektir. Buradaki $t+1$ alt indisler "sonraki" anlamına gelmektedir. Örneğin r_{t+1} eski durumdan yeni duruma geçiş için verilecek ödülü anlatmaktadır. (yani s_t 'den s_{t+1} 'e geçiş için verilecek ödül.) Formülün max ile ilgili kısmı şu anlamda gelmektedir: s_t durumundan s_{t+1} durumuna geçiş planlandığı zaman kendimizin s_{t+1} durumunda olduğunu varsayırsak oradan başka bir duruma geçiş için elde edilecek maksimum kalite (ya da memnuniyet diyebiliriz) bulunmak istenmektedir. Çünkü Q matrisi aslında nihayetinde her durumdan mümkün her eylemin amaç doğrultusundaki kalitesini (Quality) belirtir. Yani Q matrisinde $[s, a]$ elemanı s durumundayken a eylemi gerçekleştirildiğinde bundan ne kadar memnuniyet duyulacağını belirtmektedir.

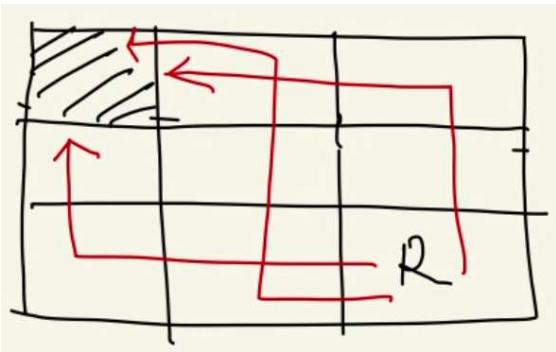
Peki her eylemden sonra Q -Tablosu güncelleneceğine göre öğrenme süreci toplamda nasıl olacaktır? İşte öğrenme süreci kabaca şöyle bir algoritmayla gerçekleştirilir:

- 1) Öğrenme süreci bölüm bölüm (episode) yapılmaktadır. Her bölüm rastgele bir durumdan başlatılır.
- 2) Her durumda en iyi eylem gerçekleştirilmeye çalışılır. Bunun için alternatifler arasındaki en iyi Q değeri seçilir. Tabii bu Q değeri yalnızca tablodan değil yine yukarıdaki formülden elde edilmelidir.
- 3) İşin başında Q tablosu tamamen boş olduğuna göre rastgele bir eylem seçilebilir. Tabloda duruma ilişkin satırın tüm elemanları 0 ise rastgele bir eylemin seçilmesi uygun olabilmektedir. Bazen alternatif Q değerleri aynı da olabilmektedir. Bu durumda yine rastgele bir eylem seçilebilir.

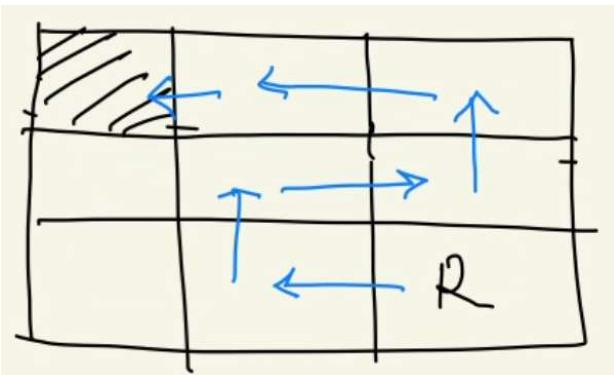
Şimdi çok basit bir problem üzerinde düşünelim. Etmenimiz bir robot olsun. Bu robot 3X3'lük bir matriste sola, sağa, yukarı aşağı gidebilsin. Ama çapraz olarak gidemesin. Burada robotun amacı belli bir hücreye hızlı (en az yol kat edecek biçimde) varmak olsun:



Burada taralı hücre hedefi belirtiyor olsun. Göründüğü gibi hedefe varmanın en kısa yolu 4 adımdır. Bu yol farklı biçimlerde oluşturulabilmektedir. Örneğin:



Ancak aşağıdaki gibi bir rota hedefe daha yavaş varmaktadır:



Böyle bir problemi rastgele hareketlerle çözmeye çalışabiliriz. Her hedefe varlığında yolu hesaplayabiliriz. Ancak bu bir öğrenmeye yol açmaz. Yani robot her defasında rastgele hareketler yapar. Halbuki istediğimiz şey robotu eğitmek ondan sonra onu herhangi bir hücreye koyduğumuzda en kısa yolu bulmasını sağlamaktır. Öğrenmenin "davranışta kalıcı bir değişiklik oluşturan süreçler" biçiminde tanımladığını anımsayınız. Rastgele hareketler daha sonra kullanılabilecek bir davranış değişikliğine yol açmamaktadır.

Burada öğretici olarak robota yaptığı hareketlerden ödül (ya da ceza) vermemiz gereklidir. Peki nasıl bir ödül mekanizması oluşturulabilir? Akla gelen makul ödül mekanizması hedefe yaklaştıkça artan, uzaklaştıkça azalan ya da negatif hale gelen bir mekanizma olabilir. Tabii bunun için bizim robotun durumuyla hedef arasındaki ilişkiyi biliyor olmamız gereklidir. Ancak her uygulamada buna benzer bir bilgiyi öğretici bilmiyor olabilir. Bu nedenle ödül sistemi genellikle daha basit bir biçimde oluşturulmaktadır. Örneğin burada en basit bir ödül mekanizması "hedefe varan hareketin 1, hedefe varmayan hareketin 0" olarak ödüllendirilmesidir. (Tabii 0 ödül aslında ödül vermemek negatif ödül ise ceza vermek anlamındadır.)

Bu problemi Q-Learning algoritmasıyla çözebilme için yukarıda da belirtildiği gibi bir Q-Tablosunun oluşturulması gereklidir. Q-Tablosu satırlarda durumların sütunlarda eylemlerin (actions) olduğu bir tablodur. Tipik olarak matrisel bir biçimde oluşturulabilen gibi sözlük gibi veri yapılarıyla da oluşturulabilmektedir. Q-Tablosunu oluşturabilmek için ise şüphesiz bizim önce problemi durumlar (states) ve eylemler (actions) biçiminde tanımlamamız gereklidir. Eylemler ise robotun hareketleridir. Biz her hücreyi bir durumla temsil edebiliriz:

S0	S1	S2
S3	S4	S5
S6	S7	R S8

Robotun bir durumdayken başka bir duruma geçmesi biçiminde tanımlanan eylemleri de dört tanedir: Yukarı, Sağ, Aşağı, Sola gitmek. Tabii bu problemde robotun her durumda her eylemi yapamayacağı da görülmektedir. Örneğin robot S8 durumunda sağa ve aşağı gidememektedir. Bu gidemeyiş programlamada birbir biçimde oluşturulmalıdır. Bazen mümkün hareketler sanka mümkün olmuş gibi düşünülüp ona yüksek cezalar da verilebilmektedir. Bu durumda Q-Tablosu aşağıdaki gibi ifade edilebilir:

	Yukarı	Sağ	Aşağı	Sola
S0				
S1				
S2				
S3				
S4				
S5				
S6				
S7				
S8				

Q - Tablosu

Q-Learning algoritmasında tablodaki hücreleri güncellerken kullanılan iki parametrik değerini olduğunu görüyorsunuz. Bunlar alfa ve gama değerleridir. Alfa değerine "öğrenme hızı (learning rate)" gama değerine ise "indirim (discount)" denilmektedir. Bu iki parametre de 0 ile 1 arasında bir değer almaktadır. Biz bu örneğimizde alfa değerini 1, gama değerini 0.90 olarak alalım:

$$Q(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \left(\underbrace{r_{t+1} + \gamma \max_a Q(s_{t+1}, a)}_{\substack{\text{learned value} \\ \text{reward} \\ \text{discount factor} \\ \text{estimate of optimal future value}}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \right)$$

Q-Learning algoritmasında oluşturulan Q tablosu baştan 0'larla doldurulmuştur. Q-Tablosu yukarıda da belirtildiği gibi aslında hangi durumdayken hangi eylemlerin daha iyi (kaliteli) olduğunu göstermektedir. İşin başında biz hangi eylemlerin hangi iyilikte olduğunu bilememekteyiz. Bu nedenle Q-Tablosu da 0'larla doludur.

Şimdi robotumuzun S1'de olduğunu ve S0'a gitmek istedğini düşünelim. Eğer robot S1'de ise ve S0'a gitmek istiyorsa bu eylem sonucunda bizim tablonun S1-Sola hücresini güncellememiz gereklidir. Bu güncelleme yukarıdaki formüle göre yapılacaktır. Alfa için 1, gama için 0.90 değerlerini uygun görmüştük. Formüldeki r_{t+1} S1'den Sola gidildiğinde elde edilecek ödülü belirtmektedir. Bu ödülün 1 olduğunu belirlemiştik. Formüldeki $\max_a Q(s_{t+1}, a)$ bulunulan

durumdan belli durumlara gidildiğiindeki o durumlara özgü en yüksek tablo değerini belirtir. Yani örneğin biz S1'deyken sağa gitmek istediğimizde kendimizi bu yeni S2'de olduğumuzu düşünerek burada yapabielceğimiz en iyi eylemi bulmak istemekteyiz.

Biz S1'de olalım ve S0'a gitmek isteyelim. Bu durumda tabloda güncelleyeceğimiz hücre Q(s1, sola) hücresi olacaktır. Yukarıdaki formüle göre güncellememizi yapalım:

$$Q(s1, \text{sola}) = Q(s1, \text{sola}) + 1 * (1 + 0.90 * 0 - Q(s1, \text{sola}))$$

Buradan 1 değeri elde edilecektir. S1'den sola gidildiğinde varılacak yer hedef olduğu için formülün max elemanından 0 elde edildiğini varsayıyoruz. Bu durumda Q-Tablosu şu hale gelecektir:

	Kısa	Sağ	Aşağı	Sola
S0	0	0	0	0
S1	0	0	0	1
S2	0	0	0	0
S3	0	0	0	0
S4	0	0	0	0
S5	0	0	0	0
S6	0	0	0	0
S7	0	0	0	0
S8	0	0	0	0

(Q - Tablosu)

Şimdi robotumuzun S2'de olduğunu varsayıyalım. S2'den sola gitmek isteyelim. Bu durumda biz tablomuzun Q(s2, sola) hücresini güncelleriz. Formülü uygulayalım:

$$Q(s2, \text{sola}) = Q(s2, \text{sola}) + 1 * (0 + 0.90 * 1 - Q(s1, \text{sağa}))$$

Buradan 0.90 değeri elde edilmektedir. Algoritmamızın kritik nokta formüldedek max değeridir. Biz S2'den sola gitmek istedik. Hedef olarak varacağımız yer S1'dir. İşte bu max değeri robotun S1'de olduğu fikriyle oradaki eylemlere ilişkin en büyük tablo değerini belirtir. Bu da 1'dir.

	Kısa	Sağ	Aşağı	Sola
S0	0	0	0	0
S1	0	0	0	1
S2	0	0	0	0
S3	0	0	0	0
S4	0	0	0	0
S5	0	0	0	0
S6	0	0	0	0
S7	0	0	0	0
S8	0	0	0	0

(Q - Tablosu)

güncellenecek hücre

S1'den S2'e gidildiğinde
S1'deki tablo değerinden en
büyük

Q-Tablosunun yeni durumu şöyledir:

	Xukarı	Sağrı	Aşağı	Solo
s0	0	0	0	0
s1	0	0	0	1
s2	0	0	0	0.90
s3	0	0	0	0
s4	0	0	0	0
s5	0	0	0	0
s6	0	0	0	0
s7	0	0	0	0
s8	0	0	0	0

Q - Tablosu

Şimdi robotun s1'de olduğunu ve s2'ye gitmek istediğini düşünelim. Şüphesiz bu iyi bir haraket değildir. Fakat Q(s1, sağa) hücresini formüle göre güncelleyelim:

$$Q(s1, \text{sağa}) = 0 + 1 * (0 + 0.90 * 0.90 - 0)$$

Buradan 0.81 değeri elde edilir. Buradaki max elemanın 0.90 değerinde olduğuna dikkat ediniz. Çünkü robot s1'den s2'ye girmek istediğiinde s2'de olduğu fikriyle oradaki eylemlerin en yüksek tablo değeri 0.90'dır:

	Xukarı	Sağrı	Aşağı	Solo
s0	0	0	0	0
s1	0	0	0	1
s2	0	0	0	0.90
s3	0	0	0	0
s4	0	0	0	0
s5	0	0	0	0
s6	0	0	0	0
s7	0	0	0	0
s8	0	0	0	0

Q - Tablosu

güncellendi hâle

Robotun s2'de olduğu
fikriyle orada yapabileceği,
eylulen en yüksek değer

Q-Tablosunun yeni hali şöyledir:

	Yukarı	Sağ	Aşağı	Sol
S0	0	0	0	0
S1	0	0.81	0	1
S2	0	0	0	0.90
S3	0	0	0	0
S4	0	0	0	0
S5	0	0	0	0
S6	0	0	0	0
S7	0	0	0	0
S8	0	0	0	0

Q - Tablosu

Şimdi robotumuzun S5'te olduğunu varsayıyalım ve robotumuz yukarıya gitmek istesi. Bu durumda Q-Tablosundaki Q(s5, yukarıya) hücresi güncellenecektir. Formülü uygulayalımlı:

$$Q(s5, \text{yukarıya}) = 0 + 1 * (0 + 0.90 * 0.90 - 0)$$

Buradan elde edilecek değer 0.81'dır. Çünkü S5'ten yukarıya gittiğimizde S2 durumuna geçeriz. Robotumuzun S2'de olduğu fikriyle oradan elde edebileceğim en yüksek değer (s2 satırından) 0.90'dır. O halde tablomuzun yeni durumu şöyledir olacaktır:

	Yukarı	Sağ	Aşağı	Sol
S0	0	0	0	0
S1	0	0.81	0	1
S2	0	0	0	0.90
S3	0	0	0	0
S4	0	0	0	0
S5	0.81	0	0	0
S6	0	0	0	0
S7	0	0	0	0
S8	0	0	0	0

Q - Tablosu

Şimdi de robotumuzun S8'de olduğunu varsayıyalım ve yukarıya gitmek istediğimi düşünelim. Bu durumda Q-Tablosunun Q(s8, yukarıya) elemanı güncellenecektir. Formülü uygulayalımlı:

$$Q(s8, \text{yukarıya}) = 0 + 1 * (0 + 0.90 * 0.81 - 0)$$

Buradan elde edilecek değer 0.729'dur. Tablomuzun yeni hali şöyledir:

	Yukarı	Sağ	Aşağı	Sola
S0	0	0	0	0
S1	0	0.81	0	1
S2	0	0	0	0.90
S3	0	0	0	0
S4	0	0	0	0
S5	0.81	0	0	0
S6	0	0	0	0
S7	0	0	0	0
S8	0.729	0	0	0

Q - Tablosu

Şimdi tablomuzu biraz daha doldurmaya çalışalım. Örneğin robotumuzun S8'de olduğunu ve sola gitmek istediğini varsayıyalım:

$$Q(s8, sola) = 0 + 1 * (0 + 0.90 * 0 - 0)$$

Buaradan 0 değeri elde edilecektir. Aslında S8'den sola gitmek izlenebilecek bir yoldur. Ancak tablo henüz boş olduğu için bu değer 0 olarak elde edilir. Burada biz başlangıçta boş olan tablonun ödül almadıkça anlamlı değerlerle doldurulamadığını görüyoruz. Tablonun doldurulma stratejisi Q-Learning algoritmasında önemli bir konudur.

Şimdi robotumuzun S3'te olduğunu varsayıyalım ve yukarıya gitmek isteyelim:

$$Q(s3, yukarıya) = 0 + 1 * (1 + 0.90 * 0 - 0)$$

Burada 1 değeri elde edilecektir. Tablomuzun hali şöyledir:

	Yukarı	Sağ	Aşağı	Sola
S0	0	0	0	0
S1	0	0.81	0	1
S2	0	0	0	0.90
S3	1	0	0	0
S4	0	0	0	0
S5	0.81	0	0	0
S6	0	0	0	0
S7	0	0	0	0
S8	0.729	0	0	0

Q - Tablosu

Şimdi sırasıyla birkaç elemanı güncelleylelim:

$$Q(s4, yukarıya) = 0 + 1 * (0 + 0.90 * 1 - 0) = 0.90$$

$$Q(s6, yukarıya) = 0 + 1 * (0 + 0.90 * 1 - 0) = 0.90$$

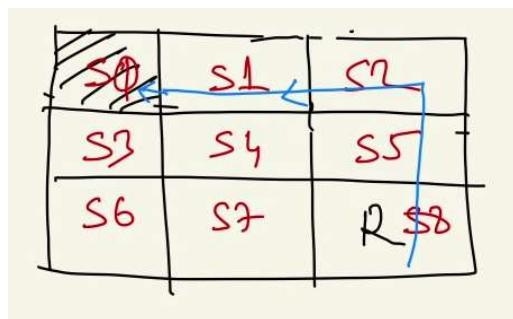
$$Q(s7, sağa) = 0 + 1 * (0 + 0.90 * 0.729 - 0) = 0.656$$

$$Q(s7, \text{yukarıya}) = 0 + 1 * (0 + 0.90 * 0.90 - 0) = 0.81$$

	<i>Yukarı</i>	<i>Sağ</i>	<i>Aşağı</i>	<i>Sola</i>
<i>S0</i>	0	0	0	0
<i>S1</i>	0	0.81	0	1
<i>S2</i>	0	0	0	0.90
<i>S3</i>	1	0	0	0
<i>S4</i>	0.90	0	0	0
<i>S5</i>	0.81	0	0	0
<i>S6</i>	0.90	0	0	0
<i>S7</i>	0.81	0.656	0	0
<i>S8</i>	0.729	0	0	0

Q - Tablosu

Böylesi işlemlerle tablonun doldurulduğunu ve hücrelerinde güncellendiğini düşünelim. En son varacağımız nokta ne olacaktır? İşte en sonunda biz robotumuzu herhangi bir hücreye bıraktığımızda robotumuz bu tabloya bakarak hedefe (*S0* hedef) en kısa sürede ulaşacaktır. Doldurulmuş ve güncellenmiş bir Q-Tablosunda etmenin hedefe gitmesi şöyle gerçekleşir: Etmen hangi konumdaysa her zaman o konumdaki en yüksek tablo değerine ilişkin eylemi gerçekleştirir. Sonra yeni durumda da aynı işlemleri yapar. Örneğin yukarıda zayıf doldurulmuş tabloda robotun *S8*'de olduğunu düşünelim. *S8* satırındaki en yüksek puanlı eylem yukarıya gitmektedir. Robot yukarı gidince *S5* durumuna gelmiş olur. *S5* satırındaki en yüksek puanlı eylem yukarı gitmektedir. Böylece robot *S2*'ye gelir. *S2* satırındaki en yüksek puanlı eylem ise sola gitmektedir. Böylece robot *S1*'ye gelmiş olur. *S1* satırındaki en yüksek puanlı eylem ise sola gitmektedir. Böylece robot hedefe ulaşmış olur.



Formüldeki indirim (discount) parametresi ne anlam ifade ettiğine yönelik bir ip ucu elde etmiş olabilirsiniz. Bu indirim parametresi hedeften uzaklaşıkça puan düşürmeye etkildir. 1'den küçük olursa puan o kadar çabuk düşürülür.

Gym Taxi-V3 Simülasyonu

Bu simülasyonda 5×5 'lik bir alanda bir taksi bulunmaktadır. Her biri ayrı bir renkle gösterilen 4 ayrı indirme-bindirme durağı vardır. Burada öğrenilecek iş taksinin yolcuya belirlenen bir duraktan alıp başka bir belirlenen durağa bırakmasıdır. Durakların yerleri hep sabit olmakla birlikte yolcu bu duraklardan herhangi birinde bulunuyor olabilir ve bu duraklardan herhangi birine gitmek istiyor olabilir. Alan içerisindeki matriste duvarlar da vardır. Dolayısıyla duvara çarptığında diğer tarafa geçilememektedir. Çizdirilen şekilde Mavi her zaman yolcunun alınacağı durağı, mor ise yolcunun bırakılacağı durağı belirtmektedir. Arabanın içerisinde yolcu yoksa araba kırmızı, yolcu varsa yeşil olarak gösterilmektedir. Örneğin:

```
+-----+
|R: | : : G| |
| : | : : |
| : | : : |
| : | : : |
|Y| : |B: |
+-----+
```

Burada yolcu sol üst duraktan alınacak, sol alt durağa bırakılacaktır.

Simülasyonu başlatmak için gym.make fonksiyonu ile simülatör ismi verilerek simülatör nesnesi elde edilir. Nesne reset edilerek başlangıç konumu oluşturulur. Örneğin:

```
import gym

env = gym.make('Taxi-v3')
obs = env.reset()
env.render()
```

```
+-----+
|R: | : : G| |
| : | : : |
| : | : : |
| : | : : |
|Y| : |B: |
+-----+
```

Simülasyonda toplam 6 tane eylem (action) vardır:

```
In [103]: env.action_space.n
Out[103]: 6
```

Bu eylemler taksinin yukarı, sola, sağa, aşağıya gitmesi, yolcu alma ve yolcu bırakma eylemleridir. Toplam durum sayısı (observation space) 500 tanedir. Örneğin:

```
In [105]: env.observation_space.n
Out[105]: 500
```

Bu sayı ilk bakışta size biraz tuhaf gelebilir. Gerçekten de bir t anında burada farklı 500 durum vardır. Taksi 5 X 5'lik alanda 25 farklı yerden birinde olabilir. Yolcu da 5 farklı yerde bulunabilir. (4 tane durak ve taksinin içi). Nihayetinde yolcunun bırakılacağı 4 farklı yer bulunmaktadır. Bunların çarpımı 500'dür. 5X5'lik matristeki hücre numaralandırmaları ekran koordinat sisteminde olduğu gibi yapılmıştır. Yani Sol-Üst köşe (0, 0) koordinatını belirtmektedir.

O anda bulunulan durum observation nesnesinden tek bir sayı olarak söyle elde edilmektedir: Yolcunun bırakılacağı yer + yolcunun konumu * 4 + arabanın sütun numarası * 20 + arabanın satır numarası * 100. Yolcunun bırakılacağı yer RGYB durumlarına göre 0, 1, 2, 3 ile, yolcunun konumu ise RGYB'ye göre 0, 1, 2, 3, 4 biçiminde kodlanmıştır. I (4) yolcunun taksinin içinde olduğu anlamına gelmektedir. Taksinin bulunduğu konum satır ve sütun olarak kodlanmıştır. Bu durumda taksi 25 yerden birinde olabilir.

Örneğin yukarıdaki reset durumunun observation değeri söyledir:

```
In [8]: obs
Out[8]: 453
```

Buradaki 453 değeri söyle elde edilmiştir:

$1 (\text{yolcunun bırakılacağı yer}) + 4 * 3 (\text{yolcunun konumu}) + 20 * 2 (\text{arabanın sütun numarası}) + 100 * 4 (\text{arabanın satır numarası}) = 453$. simülatör nesnesinin encode isimli metodu sırasıyla satır no, sütun no, yolcunun konumu ve yolcunun bırakılacağı koordinatları parametre olarak alıp bize simülasyonun konumunu tek bir değer olarak verir.

Örneğin:

```
In [23]: env.encode(4, 2, 3, 1)
Out[23]: 453
```

Bunun tersini yapan decode isimli metot da vardır. decode metodu dolaşılabilir bir nesne vermektedir. Örneğin:

```
In [26]: list(env.decode(453))
Out[26]: [4, 2, 3, 1]
```

Her step işleminde simülatör tarafaından verilen ödül mekanizması şöyledir:

- Yolcu uygun yere bırakılırsa +20 ödül alınmaktadır.
- Her adımda -1 ceza verilmektedir.
- Yolcunun olmadığı yerden yolcu almak ya da hedef olmayan durağa yere yolcu bırakmak -10 ceza puanına yol açmaktadır.
- Yolcunun doğru yerden almışında herhangi bir puan verilmediği gibi yine -1 ceza verilmektedir.

Belli bir durumda yapılabilecek 6 eylemin sayısal temsilleri şöyledir:

- 0: Aşağısı
- 1: Yukarı
- 2: Sağ
- 3: Sola
- 4: Yolcu alma
- 5: Yolcu bırakma

Bir eylem yine step metoduyla gerçekleştirilmektedir. Bu metot yine bize 4'lü bir demet verir. reset ve action metodlarından elde ettiğimiz observation nesnesi o andaki ortamın durumunu vermektedir. Örneğin:

```
In [27]: obs, reward, done, info = env.step(1)

In [28]: obs
Out[28]: 353
```

Burada araba yukarı yönlü hareket ettirilmiştir. Yeni konumu 353'tür. Bu değeri decode edip yeni çizimi yapalım:

```
In [30]: list(env.decode(obs))
Out[30]: [3, 2, 3, 1]
```

```
In [31]: env.render()
```

R:		:	:	G
:		:	:	
:	:	:	:	
	:	█	:	
Y	:	:	B:	

(North)

Şimdi arabaya rastgele 10 hareket yaptıran örnek bir program oluşturalım:

```
import gym
```

```

env = gym.make('Taxi-v3')
obs = env.reset()
env.render()

total_rewards = 0
for i in range(10):
    action = env.action_space.sample()
    obs, reward, done, _ = env.step(action)
    total_rewards += reward
    env.render()

print(total_rewards)

```

Şimdi de biz Q-Learning algoritmasını kullanarak taksinin yolcuyu alıp bırakma işini öğrenmesini sağlayalım. Bu problemdeki toplam durum sayısı 500'dür. Toplam eylemlerin sayısı da 6'dır. Bu durumda bizm Q tablomuz 500 X 6'lık bir matris olmalıdır. Eğitim işlemi aşağıdaki gibi yapılabilir:

```

NEPOCHS = 100000
MAX_ITER = 1000
EPSILON = 0.1
DISCOUNT = 0.6
LEARNING_RATE = 0.1

def train(env, nepochs=NEPOCHS, max_iter=MAX_ITER, epsilon=EPSILON,
learning_rate=LEARNING_RATE, discount=DISCOUNT):
    q_table = np.zeros((env.observation_space.n, env.action_space.n))

    for i in range(nepochs):
        obs = env.reset()
        for j in range(max_iter):
            if np.random.uniform(0, 1) < epsilon:
                action = np.random.choice(env.action_space.n)
            else:
                action = np.argmax(q_table[obs])

            next_obs, reward, done, _ = env.step(action)
            if done:
                break
            q_table[obs, action] = q_table[obs, action] + learning_rate * (
                reward + discount * np.max(q_table[next_obs]) - q_table[obs, action])
            obs = next_obs

    return q_table

```

Buradaki train fonksiyonu Q-Tablosunu oluşturarak oluşturulmuş olan tabloyla geri döner. Fonksiyonda her biri en fazla MAX_ITER (1000) kadar hareket yapan NEPOCHS (1000000) kadar deneme yapılmıştır. Yani bir deneme ortam reset edilerek rastgele bir yerden başlatılır, işlem bitene kadar ya da en fazla MAX_ITER kadar devam ettirilir. Bu denemeler de toplamda NEPOCHS kadar yinelenmektedir.

Q-Learning yönteminde tabloyu doldurmak için Q-Tablosundaki en iyi eylemlerin yanı sıra arada rastgele birtakım eylemlerin de yapılması gerekmektedir. Arada rastgele eylemlerin yapılmasına "keşif" (exploration), tabloya bakarak en iyi Q değerine ilişkin eylemi uygulamaya ise "işletme" (exploitation) denilmektedir. İşte ne zaman keşif (exploration) ne zaman işletme (exploitation) uygulanacağına yönelik "keşif stratejileri (exploration strategy)" vardır. En basit keşif stratejilerinden biri "epsilon greedy" stratejisidir. Burada çoğu kez tablodaki en iyi Q değerine ilişkin eylem seçilir. Ancak arada bir (epsilon olasılıkla) rastgele bir eylem seçilir. Biz de örneğimizde keşif stratejisi olarak "epsilon greedy" yöntemini kullandık:

```

if np.random.uniform(0, 1) < epsilon:
    action = np.random.choice(env.action_space.n)
else:
    action = np.argmax(q_table[obs])

```

Burada np.random.uniform fonksiyonu 0 ile 1 arasında rastgele noktalı bir değer üretmektedir. Bu değer epsilon değerinden küçükse rastgele bir eylem seçilmiştir. Epsilon için default değerin 0.1 olduğuna dikkat ediniz. O halde eğitim sırasında her 10 eylemin ortalama 1 tanesi rastgele alınmaktadır.

train fonksiyonunda Q-Tablosunun yukarı vermiş olduğumuz formüle göre güncellendiğini görüyorsunuz:

```
q_table[obs, action] = q_table[obs, action] + learning_rate * (reward + discount *  
np.max(q_table[next_obs]) - q_table[obs, action])
```

Burada obs ve next_obs önceki sonraki durumları belirtmektedir. Yani eylem uygulanıp yeni bir durum elde edilip eski değer güncellenmiştir.

Aşağıda oluşturulmuş olan Q-Tablosunu kullanarak problemi çözen fonksiyonu görüyorsunuz:

```
def run_episode(env, q_table):  
    count = 0  
    obs = env.reset()  
    env.render()  
  
    done = False  
    while not done:  
        action = np.argmax(q_table[obs])  
        obs, reward, done, _ = env.step(action)  
        env.render()  
        count += 1  
  
    return count
```

Göründüğü gibi bu fonksiyon doldurulmuş Q-Tablosundaki en yüksek değerler eşliğinde belli bir durumdaki en iyi eylemlemi belirlemektedir. Şimdi aşağıdaki gibi bir test işlemi yapalım:

```
env = gym.make('Taxi-v3')  
q_table = train(env)  
  
result = run_episode(env, q_table)  
print(result)
```

Çıkan sonuç aşağıdaki gibidir:

```
+-----+
|R: | : :G|
|B: | : : |
| : : : : |
| : | : |
|Y| : |B: |
+-----+
```

```
+-----+
|R: | : :G|
| : | : : |
| : : : : |
| : | : |
|Y| : |B: |
+-----+
```

(North)

```
+-----+
|G: | : :G|
| : | : : |
| : : : : |
| : | : |
|Y| : |B: |
+-----+
```

(Pickup)

```
+-----+
|R: | : :G|
|G: | : : |
| : : : : |
| : | : |
|Y| : |B: |
+-----+
```

(South)

```
+-----+
|R: | : :G|
| : | : : |
|G: : : : |
| : | : |
|Y| : |B: |
+-----+
```

(South)

```
+-----+
|R: | : :G|
| : | : : |
| : G: : : |
| : | : |
|Y| : |B: |
+-----+
```

(East)

```
+-----+
|R: | : :G|
| : | : : |
| : : G: : |
| : | : |
|Y| : |B: |
+-----+
```

(East)

```
+-----+
|R: | : :G|
| : | : : |
| : : : G: |
| : | : |
|Y| : |B: |
+-----+
```

(East)

```

(East)
+-----+
|R: | : :G|
| : | : : |
| : : : : |
| : | : |B: |
+-----+
(South)
+-----+
|R: | : :G|
| : | : : |
| : : : : |
| : | : : |
|Y| : |B: |
+-----+
(South)
+-----+
|R: | : :G|
| : | : : |
| : : : : |
| : | : : |
|Y| : |B: |
+-----+
(Dropoff)
10

```

Burada araç önce maviye gidip müsteriyi alıp sonra mora gidip müsteriyi bırakıyor. Bu işlemi toplam 10 adımda bitirebilmektedir. Siz de gözle bilişel yeteneklerinizi kullanarak bu işlemin kaç adımda yapılabileceğini hesaplayınız. Şimdi run_episode fonksiyonunu yeniden çalıştırıp oluşan durumları yan yana görüntüleyelim:

(South)

+-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+
R:	: :G	R:	: :G	R:	: :G	R:	: :G	R:	: :G	R:	: :G	R:	: :G	R:	: :G	R:	: :G									
:	: :	:	: :	:	: :	:	: :	:	: :	:	: :	:	: :	:	: :	:	: :	:	: :							
:	: :	:	: :	:	: :	:	: :	:	: :	:	: :	:	: :	:	: :	:	: :	:	: :							
:	: :	:	: :	:	: :	:	: :	:	: :	:	: :	:	: :	:	: :	:	: :	:	: :							
Y	:	B:	Y	:	B:	Y	:	B:	Y	:	B:	Y	:	B:	Y	:	B:	Y	:	B:	Y	:	B:	Y	:	B:
+-----+ (West) +-----+ (South) +-----+ (South) +-----+ (Pickup) +-----+ (North) +-----+ (North) +-----+ (North) +-----+ (North)

Bu işlemin hareketli bir biçimde görüntülenmesi için şöyle bir yol izlenebilir:

```

def run_episode_record(env, q_table):
    count = 0
    obs = env.reset()

    records = []
    done = False
    while not done:
        records.append(env.render(mode='ansi'))
        action = np.argmax(q_table[obs])
        obs, reward, done, _ = env.step(action)
        count += 1

    return count, records

count, records = run_episode_record(env, q_table)

```

Burada run_episode_record problemi çözmekle birlikte oradaki görüntüyü bir listede biriktirmektedir. render metodunun mode parametresinin 'ansi' biçiminde girildiğine dikkat ediniz. Bu parametre sayesinde çıktı ekranı basılmaz. ANSI terminal sürücüsünün anlayabileceği ESC dizimleri biçiminde görüntüyü yazışal biçimde bize verir. Yani aslında fonksiyonda birtakım yazılar biriktirilmektedir. Bu yazılar zaman aralıklarıyla aynı yere bastırılırsa sanki görüntü hareketlimiş gibi bir etki oluşmaktadır. Ekranın silinmesi için 'x1b[1J' özel yazısının yazdırılması yeterli olmaktadır. Eylemleri hareketli görüntülemek için de şöyle bir fonksiyon yazılabilir:

```

def disp_records(records):
    for record in records:
        print('\x1b[1J')
        print(record)

```

```

    time.sleep(0.5)
    print(count)

count, records = run_episode_record(env, q_table)
disp_records(records)

```

Gym FrozenLake8x8-v0 Simülasyonu

Gym'in bu simülasyonunda bir kişi buzlu bir zemin üzerinde bir yerden (S) bir yere (G) varmak istemektedir. Ancak buzlu zeminde çatıklar (delikler) vardır. Bu çatıklara düşüldüğünde oyun biter. Hedefe varlığında 1 puan ödül verilmektedir. Düşme ya da hedefe varmamaya yol açan her hareket 0 ödülün verilmesine yol açmaktadır. Simülasyon şu biçimde başlatılabilir:

```

import gym

env = gym.make('FrozenLake8x8-v0', is_slippery=False)
obs = env.reset()
env.render()

```

```

SFFFFFFF
FFFFFFF
FFFHFFF
FFFFFHFF
FFFHFFFF
FHHFFFHF
FHFFFHFF
FFFHFFFG

```

Buradaki karakterler şu anlamlara gelmektedir:

- S: Başlangıç noktası (Source)
- F: Donmuş yer (Frozen)
- H: Delik (Hole)
- G: Hedef (Goal)

Bu simülasyondaki toplam durumların sayısı yani observation_space 64'tür. Çünkü gözlemlenecek tek şey 8X8'lük matristir. Eylemlerin sayısı yani action_space ise 4'tür. Eylemler kişinin gideceği yönü (yukarı, aşağı, sola, sağa) belirtmektedir. Yön belirten değerler şunlardır:

- 0: Left
- 1: Aşağı
- 2: Sağ
- 3: Yukarı

Oyunun iki versiyonu vardır: Kaygan mod ve kaygan olmayan mod. Default durum kaygan moddur. Kaygan modda kişi her zaman kayganlıktan dolayı istenilen yönde hareket edemeyebilmektedir. (Yani örneğin sağa gitmek isterken kaayarak aşağıya gidebilmektedir.) Eğer make fonksiyonunda is_slippery isimli parametresi False geçilirse kaygan olmayan mod seçilmiş olur. Bu durumda kişi her zaman istediği yönde hareket edebilmektedir.

Oyundaki ödül mekanizması hedefe varlığında 1 puan diğer bütün eylemlerde 0 puan biçimindedir. Yani oyunda yalnızca hedefe varlığında ödül alınmaktadır.

Aşağıda rasgele hareketler yapan örnek bir program vardır:

```

import gym
import time

env = gym.make('FrozenLake8x8-v0', is_slippery=False)

```

```

obs = env.reset()

count = 0
for i in range(1000):
    action = env.action_space.sample()
    obs, reward, done, info = env.step(action)
    print('\x1b[1J')
    env.render()
    print(count)
    count += 1
    if done:
        break
    time.sleep(0.2)

(Down)
SFFFFFFF
FFFFFFFF
FFFHFFFF
FFFFFHFF
FFFHFFFF
FHFFFFHF
FHFFHFHF
FFFHFFFFG
12

```

Şimdi de Q-Learning algoritmasıyla problemi çözelim:

```

import gym
import time

import numpy as np

NEPOCHS = 5000
MAX_ITER = 100
DISCOUNT = 0.9
LEARNING_RATE = 0.6

def train(env, nepochs=NEPOCHS, max_iter=MAX_ITER, learning_rate=LEARNING_RATE,
discount=DISCOUNT):
    q_table = np.zeros((env.observation_space.n, env.action_space.n))

    for i in range(nepochs):
        obs = env.reset()
        for j in range(max_iter):
            action = np.argmax(q_table[obs] + np.random.randn(1, env.action_space.n) * (1. / (i + 1)))
            next_obs, reward, done, _ = env.step(action)

            q_table[obs, action] = q_table[obs, action] + learning_rate * (
                reward + discount * np.max(q_table[next_obs]) - q_table[obs, action])
            obs = next_obs
            if done:
                break

    return q_table

```

Eğitimde keşif stratejisi olarak normal dağılım yöntemi kullanılmıştır. Bu yöntemde Q-Tablosunun ilgili satırındaki eylemlerin Q değerlerine normal dağılıma uygun rastgele değerler toplanmıştır. Bu toplamın en yüksek Q değeri eylem olarak seçilmiştir:

```
action = np.argmax(q_table[obs] + np.random.randn(1, env.action_space.n) * (1. / (i + 1)))
```

Aynı zamanda her yinelemede bu rastgele normal dağıl değerlerinin etkisinin $1 / (i + 1)$ çarpanı ile düşürüldüğünü görüyorsunuz. Şimdi de simülatörü çalıştırıp hedefe bulan fonksiyonu yazalım:

```
def run_episode(env, q_table):
    count = 0
    obs = env.reset()
    done = False
    while not done:
        action = np.argmax(q_table[obs])
        obs, reward, done, _ = env.step(action)
        print('\x1b[1J')
        env.render()
        time.sleep(0.4)
        count += 1
    return count
```

Programı şöyle çalıştırabiliriz:

```
env = gym.make('FrozenLake8x8-v0', is_slippery=False)
q_table = train(env)

count = run_episode(env, q_table)
print(count)
```

Bu simülatörde simülatör reset edildiğinde her zaman başlangıç S pozisyonu (sol-üst köşe) olmaktadır. Kişinin başlangıç başlangıç pozisyonunun değiştirilmesine yönelik bir dokümantasyon yoktur. Ancak sınıfların yapısı incelendiğinde pozisyon bilgisinin env.env.s değişkeninde turulduğu görülmektedir. Yani biz bu değişkendeki değeri değiştirerek simülatörün S dışında başka bir yerden çalışmaya başlamasını sağlayabiliriz. Aşağıdaki örnekte eğitim sırasında her defasında rastgele bir başlangıç noktası seçilmiştir:

```
import gym
import time

import numpy as np

NEPOCHS = 50000
MAX_ITER = 1000
DISCOUNT = 0.8
LEARNING_RATE = 0.1

def train_random(env, nepochs=NEPOCHS, max_iter=MAX_ITER, learning_rate=LEARNING_RATE,
discount=DISCOUNT):
    q_table = np.zeros((env.observation_space.n, env.action_space.n))

    for i in range(nepochs):
        obs = env.reset()
        env.env.s = np.random.randint(env.observation_space.n)
        for j in range(max_iter):
            action = np.argmax(q_table[obs] + np.random.randn(1, env.action_space.n) * (1. / (i + 1)))
            next_obs, reward, done, _ = env.step(action)

            q_table[obs, action] = q_table[obs, action] + learning_rate * (
                reward + discount * np.max(q_table[next_obs]) - q_table[obs, action])
            obs = next_obs
            if done:
                break

    return q_table
```

```

def run_episode_random(env, q_table):
    count = 0
    obs = env.reset()
    env.env.s = np.random.randint(env.observation_space.n)
    done = False
    while not done:
        action = np.argmax(q_table[obs])
        obs, reward, done, _ = env.step(action)
        print('\x1b[1J')
        env.render()
        time.sleep(0.4)
        count += 1
    return count

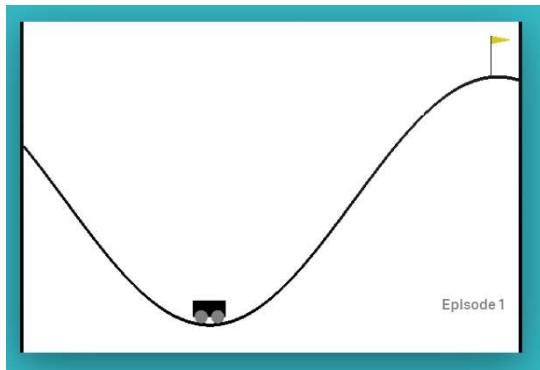
env = gym.make('FrozenLake8x8-v0', is_slippery=False)
q_table = train_random(env)

count = run_episode_random(env, q_table)
print(count)

```

MountainCar Örneğinin Q Learning Algoritmasıyla Çözümü

Anımsanacağı gibi OpenAI Gym içerisindeki MountainCar simülöründe bir araba ileri ya da geri gidebiliyor ya da olduğu yerde durabiliyordu. Belli bir t anında bu hareketlerden yalnızca birini yapabiliyordu. Amaç ise onun tepeyi aşmasıydı. Daha önceden de gördüğümüz gibi araba sürekli ileri gitmeye çalışırsa tepeyi aşmaya gücü yetmemektedir. Biz daha önce bu problemi sezgisel biçimde çözmüştük. Arabayı hız 0'a düşene kadar ileri, daha sonra yine hız sıfıra düşene kadar geri yönde hareket ettirerek tepeyi aşrtmıştık. Burada ise arabanın tepeyi aşması için yapması gereken şeyle Q-Learning algoritmasıyla pekiştirmeli bir biçimde arabaya öğretilecektir.



Anımsanacağı gibi bu problemdeki mevcut eylemler (action space) 0, 1 ve 2'dir. 0 "geri git", 1 "gaz verme (motoru durdur)", 2 ise "ileri git" anlamına gelmektedir. Problemdeki gözlem değerleri (observation_space) ise iki elemanlı bir numpy dizisi olarak bize verilmektedir. Dizinin 0'inci indisli elemanı arabanın x eksenindeki konumu, 1'inci indisli elemanı ise arabanın o andaki hızını belirtmektedir. Burada işlemin başarısını belirten done koşulu iki duruma bağlıdır: Birincisi arabanın tepeyi aşması, ikincisi ise maksimum deneme sayısı olan 200 denemenin bitmesidir.

Bu problemin Q-Learning algoritmasıyla çözümünde ilk yapılacak şey durumların (states) oluşturulmasıdır. Çünkü gözlemlerdeki konum ve hız değerleri sürekli değerlerdir. Bizim bu değerleri ayrık hale getirip durumsallaştırmamız gereklidir. Sürekli gözlem değerlerini ayrık biçimde durumsallaştırırken bir bölge değerini önceden belirleyebiliriz. Tabii aslında her gözlemin diğer gözlemlerle aynı sayıda parçaya ayrılması gerekmez. Fakat uygulamada tüm gözlemlerin aynı sayıda parçaya ayrılması kolay bir tasarımdır. Örneğin MountainCar simülasyonunda parça sayısını 20 olarak belirlediğimizi düşünelim. Bu durumda gözlem değerlerinden biri olan konum maksimum ve minimum değerler dikkate alınarak 20 parçaya bölünecektir. Benzer içimde arabanın hızı da maksimum ve minimum değerler dikkate alınarak yine 20 parçaya bölünebilir. Arabanın içinde bulunduğu durum (state) böylece aslında 20×20 durumdan bir tanesi olacaktır. Matrisel biçimde düşünüldüğünde bunun için 20×20 'lik iki boyutlu bir matris oluşturulabilir. Arabanın içinde bulunduğu durum konum ve hız ile temsil ediliyor olsa da biz bu durumu matrisel

birimde iki indeksle temsil edebiliriz. (Tabii bu tür örneklerde observation_space daha fazla elemana sahipse bu durumda matris de daha fazla boyuttan oluşacaktır. Bizim durumsal indekslerimiz de daha fazla bileşene sahip olacaktır. Örneğin CartPole simülasyonundaki gözlem değişkenleri 4 tanedir.) Aşağıda MountainCar simülasyonu için bir gözlemi indekslerden oluşan durumsal bir bilgiye dönüştüren fonksiyon örneği verilmiştir:

```
import gym

env = gym.make('MountainCar-v0')

DISCRETE_STATES = 20

def obs_to_state(env, obs):
    state_intervals = (env.observation_space.high - env.observation_space.low) /
DISCRETE_STATES
    return ((obs - env.observation_space.low) / state_intervals).astype(int)
```

Şimdi Q-Tablosunu oluşturmaaya çalışalım. MountainCar örneğindeki eylemlerin sayısı 3 olduğuna göre oluşturacağımız Q-Tablosu da $20 \times 20 \times 3$ boyutlarında olacaktır. Q-Tablosu istenilen boyutta şöyle oluşturulabilir:

```
q_table = np.zeros([20, 20, 3])
```

Buradaki boyutu ileride ele alacağımız bir neden dolayı bir fazla olarak oluşturacağız. Bu durumda Q-Tablosu env nesnesinden hareketle daha parametrik bir biçimde şöyle de oluşturulabilir:

```
index_list = [DISCRETE_STATES + 1] * env.observation_space.shape[0] + [env.action_space.n]
q_table = np.zeros(index_list)
```

Şimdi çeşitli denemeler yaptırarak sistemi öğrenme sürecine sokmaya çalışalım. Burada yaptırılan denemeler Q-Tablosunun uygun biçimde doldurulmasına yol açacaktır. Bu örnekte keşif stratejisi (exploration strategy) olarak "epsilon greedy" yönteminin bir varyasyonunu kullanacağız. Kullanacağımız yöntemde EPSILON olasılıkla tamamen rastgele bir eylem seçimi yapacağız. Ancak diğer durumlarda birbirlerine benzer Q değerlerinin arasında birz rassallık da oluşturacağız.

```
import gym

NEPOCHS = 500
MAX_ITER = 1000
LEARNING_RATE = 0.01
EPSILON = 0.2
DISCOUNT = 0.90
DISCRETE_STATES = 15

def obs_to_state(env, obs):
    state_intervals = (env.observation_space.high - env.observation_space.low) /
DISCRETE_STATES
    return ((obs - env.observation_space.low) / state_intervals).astype(int)

import numpy as np

def train(env):
    print('Training starts...')

    index_list = [DISCRETE_STATES + 1] * env.observation_space.shape[0] + [env.action_space.n]
    q_table = np.zeros(index_list)
    for i in range(NEPOCHS):
        obs = env.reset()
        for k in range(MAX_ITER):
            a, b = obs_to_state(env, obs)
            if np.random.uniform(0, 1) < EPSILON:
                action = np.random.choice(env.action_space.n)
            else:
```

```

logits = q_table[a, b]
logits_exp = np.exp(logits)
probs = logits_exp / np.sum(logits_exp)
action = np.random.choice(env.action_space.n, p=probs)

next_obs, reward, done, _ = env.step(action)
a_next, b_next = obs_to_state(env, next_obs)
q_table[a, b, action] = q_table[a, b, action] + LEARNING_RATE * (
    reward + DISCOUNT * np.max(q_table[a_next, b_next]) - q_table[a, b,
action])
obs = next_obs
print(i, end=' ')
print()

return q_table

```

Buradaki train fonksiyonu hakkında bazı açıklamalar yapmak istiyoruz:

- Fonksiyonda iç içe iki döngü vardır. Dış döngü NEPOCHS kadar dönmektedir. NEPOCHS toplam yapılan denemelerin sayısını belirtir. İç döngü ise MAX_ITER kadar dönmektedir. MAX_ITER de bir denemede en fazla kaç hareket yapılacağını belirtir. Teorik olarak NEPOCHS ve MAX_ITER sayıları ne kadar fazla olursa öğrenmenin de o kadar iyi olacaktır.

- Denemeler sırasında seçilecek eylem (action) aşağıdaki gibi bir koşulla belirlenmiştir:

```

if np.random.uniform(0, 1) < EPSILON:
    action = np.random.choice(env.action_space.n)
else:
    logits = q_table[a, b]
    logits_exp = np.exp(logits)
    probs = logits_exp / np.sum(logits_exp)
    action = np.random.choice(env.action_space.n, p=probs)

```

Yukarıdaki if deyiminde eğer rastgele üretilen bir değer EPSILON'dan (%2) küçükse bu durumda rastgele bir eylem seçilmektedir. Eğer bu rastgele değer EPSILON'dan küçük değilse Q-Tablosundaki en iyi eylemler arasında belli oranlarda rastgelelik ile eylem seçimi yapılmıştır. (Yani örneğin Q tablosundaki eylemlerin puanları farklıysa yüksek puanın olasılığı yüksek olacak biçimde bir rastgelelik oluşturulmuştur.) Bu sayede Q değerleri birbirlerine yakın olan eylemler arasında her zaman en yüksek Q değerine sahip olan değil alternatifler de değerlendirilmektedir.

- Döгüdeki Q-Tablo değerlerinin güncellemesi şöyle yapılmıştır:

```

next_obs, reward, done, _ = env.step(action)
a_next, b_next = obs_to_state(env, next_obs, state_intervals)
q_table[a, b, action] = q_table[a, b, action] + learning_rate * (
    reward + GAMMA * np.max(q_table[a_next, b_next]) - q_table[a, b, action])

```

Burada konuya girişte açıkladığımız Q değerinin hesaplanma işlemi ve tablo elemanın güncelleme işlemi yapılmaktadır.

- train fonksiyonun oluşturulan Q tablosuna geri döndüğüne dikkat ediniz.

train fonksiyonu yalnızca Q tablosunu doldurarak öğrenme işlemini gerçekleştirmektedir. Yani bu öğrenme sürecinin sonunda artık biz hangi durumdan (state) başlarsak başlayalım Q tablosundaki en iyi değerlerden hareketle eylemler peşi sıra yapılacaktır. İşin bu kısmını run fonksiyonu yapmaktadır:

```

def run(env, q_table):
    count = 0
    obs = env.reset()
    while True:

```

```

env.render()
a, b = obs_to_state(env, obs)
action = np.argmax(q_table[a, b])
obs, reward, done, _ = env.step(action)
count += 1
if done:
    break

env.close()

return count

```

Fonksiyonun yaptığı tek şey aslında Q tablosundan hareketler durum geçişleri yapmaktadır. Q tablosunda bir duruma ilişkin en iyi eylem şu ifadeyle bulunmuştur:

```
action = np.argmax(q_table[a, b])
```

`np.argmax` fonksiyonunun numpy dizisi içerisinde en yüksek değerin indeksini verdiğini anımsayınız. O halde yukarıdaki ifade aslında bize o durum için en yüksek Q değerli eylemi vermektedir.

Nihayet oyunu şöyle çalıştırabiliriz:

```

env = gym.make('MountainCar-v0')
q_table = train(env)

count = run(env, q_table)
print(count)

```

CartPole Simülasyonunun Q Learning Algoritmasıyla Çözümü

Cartpole Simülasyonu aşağıdaki gibi bir Q Learning algoritması kullanılabılır:

```

import numpy as np
import gym

NEPOCH = 2000
MAX_ITER = 1000
EPS = 0.02
GAMMA = 1.5
LEARNING_RATE = 0.001
MIN_LR = 0.003
INITIAL_LR = 0.003
DISCRETE_STATES = 70

def obs_to_state(env, obs, state_intervals):
    states = ((obs - env.observation_space.low) / state_intervals).astype(int)
    states[states > DISCRETE_STATES] = DISCRETE_STATES
    states[states < 0] = 0

    return states

def train(env):
    index_list = [DISCRETE_STATES + 1] * env.observation_space.shape[0] + [env.action_space.n]
    q_table = np.zeros(index_list)
    state_intervals = (env.observation_space.high - env.observation_space.low) /
DISCRETE_STATES
    for i in range(NEPOCH):
        obs = env.reset()
        learning_rate = LEARNING_RATE
        for j in range(MAX_ITER):
            a, b, c, d = obs_to_state(env, obs, state_intervals)

```

```

if np.random.uniform(0, 1) < EPS:
    action = np.random.choice(env.action_space.n)
else:
    logits = q_table[a, b, c, d]
    logits_exp = np.exp(logits)
    probs = logits_exp / np.sum(logits_exp)
    action = np.random.choice(env.action_space.n, p=probs)
obs, reward, done, _ = env.step(action)
a_next, b_next, c_next, d_next = obs_to_state(env, obs, state_intervals)
q_table[a, b, c, d, action] = q_table[a, b, c, d, action] + learning_rate * (
    reward + GAMMA * np.max(q_table[a_next, b_next, c_next, d_next]) -
q_table[a, b, c, d, action])
print('.', end='')

print()
return q_table

def run_episode(env, q_table, render=True, rnd=False):
    state_intervals = (env.observation_space.high - env.observation_space.low) /
DISCRETE_STATES
    obs = env.reset()
    for step in range(MAX_ITER):
        a, b, c, d = obs_to_state(env, obs, state_intervals)
        if not rnd:
            action = np.argmax(q_table[a, b, c, d])

        else:
            action = env.action_space.sample()

        obs, reward, done, _ = env.step(action)
        if render:
            env.render()
        if done:
            break
    return step + 1

env = gym.make('CartPole-v0')

env.observation_space.low[1] = -10
env.observation_space.high[1] = 10

env.observation_space.low[3] = -10
env.observation_space.high[3] = 10

q_table = train(env)

total_step = run_episode(env, q_table, render=False, rnd=False)
print(total_step)

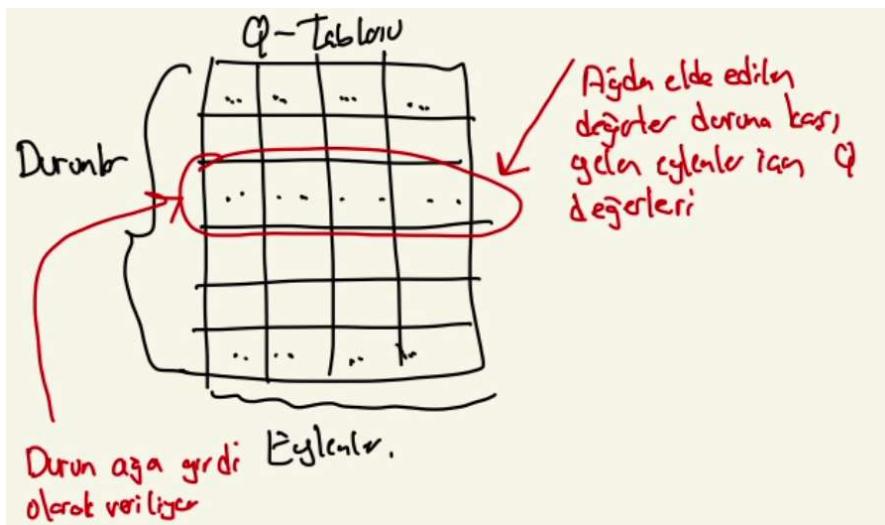
env.close()

```

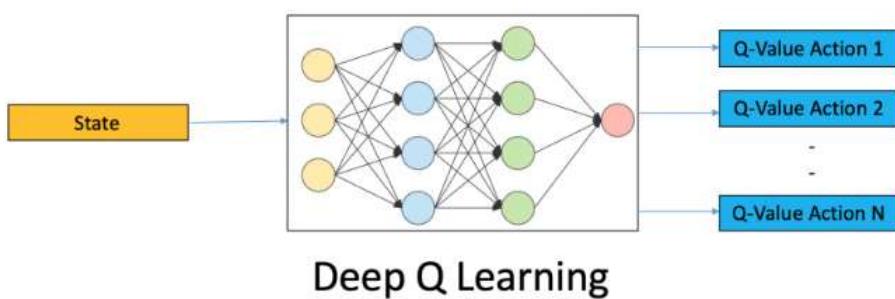
Pekiştirmeli Öğrenmede Deep Q-Learning (DQN) Yöntemi

Q Learning en çok tercih edilen pekiştirmeli öğrenme algoritmasıdır. Ancak bu algoritma durum sayısının (yani observation space'in) çok fazla olduğu özellikle de sürekli (continuous) gözlem değerlerinin söz konusu olduğu durumlarda bazı sorunlar oluşturmaktadır. Bu sorunlardan en önemlisi oluşturulacak Q-Tablosunun büyülüğu dolayısıyla kaplayacağı alandır. Aynı zamanda bu yöntemde Q-Tablosunun doldurulması için iyi bir eğitimin sağlanması gerekmektedir. Maalesef bu yöntemde çok fazla durum söz konusu olduğunda yetersiz bir eğitimde Q-Tablosunun önemli bir kısmı boş kalmamaktadır. Bu da çözümde bazı durumlarda etmenin (agent) ne yapacağını bilememesine yol açmaktadır. İşte Deep Q-Learning (DQN) yöntemi aslında yapay sinir ağları ile Q-Learning algoritmasının hibrit bir biçimidir.

Anımsanacağı gibi Q-Learning algoritmasında önce bir eğitim süreci gerçekleştiriliyor durumlar için en iyi eylemlerin ne olacağını belirten bir Q-Tablosu oluşturuluyordu. Sonra da bu Q-Tablosuna bakılarak bir rota izleniyordu. İşte Deep Q-Learning yönteminde bir Q-Tablosu oluşturulup ona başvurulmak yerine belli durumda en iyi eylemin ne olacağı yapay sinir ağından elde edilmektedir. Deep Q-Learning yönteminde oluşturulacak yapay sinir ağının girdilerini durum bilgisi oluşturmaktadır. Bu yapay sinir ağının çıktıları ise belli bir durumdaki Q değerleridir.



Deep Q-Learning yönteminde kullanılan yapay sinir ağının ilgili durum bilgisini alır, çıktı olarak da her eylem için kestirilen Q değerlerini vermektedir. Biz de çıktı Q değerlerinin en büyüğünü bulup ona karşı gelen eylemin en iyi olduğu sonucunu elde ederiz. Bu durumda buradaki ağın girdi katmanında `len(env.observation_space)` kadar nöron, çıktı katmanında da `env.action_space.n` kadar nöron olacaktır.



Sekil <https://www.analyticsvidhya.com/blog/2019/04/introduction-deep-q-learning-python/> sitesinden alınmıştır.

Deep Q-Learning yönteminde sinir ağının bir regresyon modeli oluşturduğu için bizim sürekli durum değerlerini ayrı hale getirmemize de gerek kalmamaktadır. Bu sürekli durum değerleri doğrudan ağa girdi olarak verilebilir.

Örneğin CartPole simülasyonunda bu ağın girdi katmanında 4 tane nöron olacaktır. (Arabanın x koordinatı, arabanın hızı, direğin açısı ve direğin açısal hızı). Çıktı katmanında da 2 nöron bulunacaktır. (Araba sola mı sağa mı gidecek?) Bu kestirim ağı bir lojistik regresyon değil normal çok değişkenli (multivariate) bir regresyondur. Dolayısıyla tipik olarak ağı oluştururken gizli katmanların aktivasyon fonksiyonları "relu", loss fonksiyonu "mean_squared_error" ya da "mean_absolute_error", optimizasyon algoritması ise "adam" ya da "rmsprop" ya da "sgd" alınabilir. Çıktı katmanın aktivasyon fonksiyonu da "linear" olmalıdır.

Bilindiği gibi regresyon modellerindeki sinir ağları "denetimli (supervised)" bir öğrenme teknigi kullanmaktadır. O halde Deep Q-Learning yöntemindeki sinir ağı da bir eğitim sürecine sahip olmak zorundadır. Peki biz işin başında sistem hakkında bir şey bilmemiğimize göre eğitimi nasıl uygulayacağız? İşte tipik olarak bu yönteme eğitimle kestirim çoğu kez bir arada yürütmektedir. Programcı önce bazı gözlem kümesi için Q değerlerini bulur, bunları eğitimde kullanır. Sonra da kestirim için sinir ağından faydalananır.

Deep Q-Learning Yöntemi kabaca şöyle gerçekleştirilmektedir:

- 1) Gözlemlere karşılık eylemler için Q değerlerini verecek bir yapay sinir ağı modeli oluşturulur.
- 2) Q-Learning algoritması uygulanarak belli sayıda gözle ve q değerleri elde edilir. Bu Q değerleriyle ağ eğitilir. Genellikle ağın eğitilmesi için bu Q değerlerinin belli bir sayıya (batch) erişmesi beklenir. Bunun için Q değerleri bir bellek alanında saklanabilmektedir. Bu Q değerleri belli sayıya eriştikten sonra artık iadelî seçimlerle ağıtım başlatılmaktadır. Tabii aslında eğitim hiç biriktirme yapmadan hemen de başlatılabilir.
- 3) Q-Learning algoritması işletilirken Q formülü yine uygulanır. Ancak bu formüldeki tablonun en yüksek elemanları yapay sinir ağından tahminleme yoluyla elde edilmektedir.

Deep Q-Learning Yönteminin CartPole Simülasyonunda Kullanılması

CartPole simülasyonunda Deep Q-Learning yöntemini kullanırken önce yapay sinir ağını oluştururuz. Sonra da bir keşif stratejisi ile ağı eğitiriz. Ağ yeterince eğitildikten sonra da en iyi Q değerlerinden hareketle eylemleri uygularız. Biz burada ağı eğitmırken her zaman belirleyeceğimiz bir BATCH miktarı kadar veriyi fit işlemeye sokacağız. Bu BATCH miktarı olusana kadar beklenecek sonra eğitime başlanacaktır. Bir kez biriktirdiğimiz değerler bu BATCH miktarına erişince artık her defasında biriktirilen değerler içerisinde BATCH miktarı kadarı rastgele seçilerek eğitime devam edilecektir.

```
import numpy as np
import gym
import random

DISCOUNT = 0.95
LEARNING_RATE = 0.001
BATCH_SIZE = 32
EPSILON_MAX = 1.0
EPSILON_MIN = 0.01
EPSILON_DECAY = 0.995

env = gym.make('CartPole-v1')
exploration_rate = EPSILON_MAX

observation_space = env.observation_space.shape[0]
action_space = env.action_space.n

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam

def train(env, epoch):
    model = Sequential()
    model.add(Dense(64, input_dim=observation_space, activation='relu', name='Hidden-1'))
    model.add(Dense(64, activation='relu', name='Hidden-2'))
    model.add(Dense(action_space, activation='linear', name='Output'))
    model.compile(optimizer=Adam(lr=LEARNING_RATE), loss='mse')

    epsilon = EPSILON_MAX
    memory = []

    for i in range(epoch):
        obs = env.reset()
        obs = obs.reshape((1, -1))

        count = 0
        while True:
            if np.random.rand() < epsilon:
                action = np.random.choice(action_space)
```

```

else:
    q_values = model.predict(obs)
    action = np.argmax(q_values[0])

obs_next, reward, done, _ = env.step(action)

env.render()

reward = reward if not done else -reward

obs_next = obs_next.reshape((1, -1))
memory.append((obs, action, reward, obs_next, done))

if len(memory) >= BATCH_SIZE:
    batch = random.sample(memory, BATCH_SIZE)
    for obs_mem, action_mem, reward_mem, obs_next_mem, done_mem in batch:
        q_update = reward_mem
        if not done_mem:
            q_update = (reward_mem + DISCOUNT *
np.amax(model.predict(obs_next_mem)[0]))
            q_values = model.predict(obs_mem)
            q_values[0, action_mem] = q_update
            model.fit(obs_mem, q_values, verbose=0)
    epsilon *= EPSILON_DECAY
    epsilon = max(EPSILON_MIN, exploration_rate)

obs = obs_next
count += 1

if done:
    break

print(f'{i} => {count}')

return model

def run_episode(env, model):
    obs = env.reset()
    count = 0
    while True:
        obs = obs.reshape(1, -1)
        action = np.argmax(model.predict(obs))
        obs, reward, done, info = env.step(action)
        if done:
            break
        env.render()
        count += 1

    return count

env = gym.make('CartPole-v1')
model = train(env, 300)
print('training ends...')

result = run_episode(env, model)
print(result)
env.close()

```

Burada kabaca şunlar yapılmıştır:

- Regresyon temelli bir sinir ağı modeli kurulmuştur. Bu sinir ağına CartPole gözlemleri girdi olarak verilip eylemlerin (actions) Q değerleri çıktı olarak alınmaktadır. Dolayısıyla ağ eğitildikten sonra qğın çıktısından elde edilen Q

değerlerinin en yüksek olanı eylem olarak tercih edilecektir. Programda ayrik hale getirme gibi bir işlem yapılmamıştır. Çünkü buna gerek yoktur. Zaten sinir ağı sürekli değerler için bir tahmin yapabilmektedir.

- Programda keşif stratejisi olarak "epsilon greedy" yöntemi kullanılmıştır. Ancak rassallık gitgide azaltılmıştır. Rassallık önce EPSILON_MAX değerinden başlatılır. Her yinelemede EPSILON_DECAY kadar azaltılmaktadır. Ancak rassalık en fazla EPSILON_MIN değerine kadar azaltılmış olmaktadır.

- Ağın eğitilmesi şöyle sağlanmaktadır: Önce BATCH_SIZE kadar veri toplanana kadar hiç eğitim uygulanmaz. İlk eğitim BATCH_SIZE bilgi toplandığında başlatılır. Bundan sonra da artık her yeni bilgi oluştuğunda yeniden rastgele bir BATCH_SIZE kadar bilgi çekilerek eğitim devam ettirilmektedir. Örneğimizde BATCH_SIZE 32 olarak alınmıştır. Bu durumda ilk 32 değer için hiçbir eğitim yapılmamakta sonra hep rasgele 32 değer seçilerek fit işlemi yapılmaktadır.

Bu biçimdeki modelin başarısı neye bağlıdır? Seçilen bazı parametrik değerlerin, sinir ağının genel yapısının ve diğer parametrelerin öğrenme hızında ve verilen kararlarda etkisinin olduğu açıklıdır. Programcı bu tür durumlarda deneme yanılma yöntemlerine de başvurabilir. Ancak ağın eğitilmesi için çok denemenin yapılması gerekebilir.

Pekiştirmeli Öğrenmede Kullanılabilecek Yüksek Seviyeli Kütüphaneler

Pekiştirmeli öğrenme problemleri genellikle probleme özgü bir biçimde tasarlanıp eğitilmektedir. Bu da model geliştirme zamanını artırmaktadır. İşte bu tür problemlerdeki geliştirme zamanını azaltmak için son birkaç yıldır yüksek seviyeli kütüphaneler oluşturulmaya başlanmıştır. Henüz bu kütüphaneler çok olgun bir noktada değildir. Ancak zaman içerisinde bunlar da daha iyi hale getirilecektir. Bu tür kütüphaneler arasında uygun olanını seçmek de henüz kolay bir karar değildir. Çünkü hangi kütüphanelerin ne kadar süre içerisinde olgunlaşacağını tahmin etmek zordur. Böyle bir seçimde göz önüne alınması gereken ölçütlerden bazıları şunlar olmalıdır:

- Kütüphanenin genel yeteneği (örneğin kaç algoritma destekleniyor gibi)
- Kütühanenin dokümantasyonu
- Kütüphanenin genel tasarıtı. Bazı kütüphaneler daha esnek tasarıma sahiptir.
- Kütüphaneyi oluşturan proje grubunun büyülüğu.
- Kütüphaneyi oluşturan proje grubunun destek aldığı sponsorlar.
- Kütüphanenin kolay kullanılabilir olması
- Projenin güncellenme aralığı

Kursun yapıldığı zaman aralığındaki belli başlı kütüphaneler şunlardır:

- KerasRL
- Tensorforce
- OpenAI-Baselines
- Stable Baselines
- ChainerRL
- Coach
- RLLib

Stable-Baselines Kütüphanesinin Kullanımı

Stable-Baselines Kütüphanesi aslında OpenAI kurumunun Baselines kütüphanesinin çatallanmış (fork edilmiş) bir versiyonudur. Ancak bu versiyonda orijinal BaseLines kütüphanesi pek çok bakımdan iyileştirilmiştir. Kütüphanein dokümantasyonu <https://stable-baselines.readthedocs.io/en/master/> adresinde bulunmaktadır. Kütüphane pip ile şöyle install edilebilir:

```
pip install stable-baselines
```

Kütüophane içerisindeki bazı algoritmalar paralel veri işleme yaptığı için OpenMPI kütüphanesinin de bunlar için yüklenmesi gerekmektedir. Ancak bu yalnızca kısıtlı birkaç algoritma için söz konusudur.

Stable-baselines kütüphanesi arka planda Tensorflow kullanmaktadır. Ancak kütüphanenin kurs yapıldığındaki güncel versiyonları Tensorflow'un 2'li versiyonlarını desteklememektedir. Bu nedenle Tensorflow'un 1.8.0 ve 1.15.0 arası versiyonlarının yüklenmesi gerekmektedir. Downgrade işlemi şöyle yapılabilir:

```
pip uninstall tensorflow
pip install tensorflow==1.15.0
```

Kütüphanelerin hangi versiyonlarının yüklü olduğunu anlamak için pip komutunda show seçeneği kullanılabilir. Örneğin:

```
(base) C:\Users\CSD>pip show tensorflow
Name: tensorflow
Version: 1.15.0
Summary: TensorFlow is an open source machine learning framework for everyone.
Home-page: https://www.tensorflow.org/
Author: Google Inc.
Author-email: packages@tensorflow.org
License: Apache 2.0
Location: c:\programdata\anaconda3\lib\site-packages
Requires: protobuf, google-pasta, wrapt, astor, six, opt-einsum, gast, numpy, keras-applications, tensorboard, termcolor, grpcio, wheel, tensorflow-estimator, keras-preprocessing, absl-py
Required-by:
```

Tensorflow'u downgrade yapmak yerine Python'da "sanal ortam (virtual environment)" da oluşturulabilir. Sanal ortamın nasıl oluşturulacağını Internet'teki çeşitli kaynaklardan öğrenebilirisiniz.

Stable-Baselines kütüphanesinin kullanımı oldukça basittir. Tipik çalışma şöyle yapılmaktadır:

- 1) Önce bir ortam (environment) nesnesi yaratılır. Ortam nesneleri doğrudan gym.make ile oluşturulabilmektedir. Tabii programcı isterse kendi problemi için kendi ortam nesnelerini de oluşturabilir (custom environment).
- 2) Uygulanacak algoritma belirlenir. Algoritmaların çoğu geneldir. Yani pek çok ortam için kullanılabilmektedir. En çok kullanılan algoritmalar DQN (Deep Q-Learning), PPO1 ve PPO2 (Proximal Policy Optimization), TRPO (Trust Region Policy Optimization), ACKTR (Actor Critic using Kronecker-Factored Trust Region), DDPG (Deep Deterministic Policy Gradient) algoritmalarıdır. Her algoritma aynı isimli sınıflarla temsil edilmiştir. Bu sınıf nesnelerinde genel olarak model nesneleri denilmektedir. Algoritma sınıflarına ilişkin __init__ metodlarının pek çok parametresi vardır. Parametrelerin çoğu default değerler almaktadır. Zorunlu parametrelerin birisi kullanılacak politikadır. Tipik politikalar şunlardır: MlpPolicy, LnMlpPolicy, CnnPolicy, LnCnnPolicy. En çok kullanılan politikalar MlpPolicy (Multi Layer Perceptron, 64 nörondan oluşan iki katmanlı ağ) ve CnnPolicy (Convolutional Neural Network, 64 nörondan oluşan iki katmanlı ağ)
- 3) model nesnelerinin learn metotları ile eğitim gerçekleştirilir. Artık eğitilmiş bir model elde edilmiş olur.
- 4) Eğitilmiş model gerçek durumlara uygulanır. Bunun için model sınıflarının predict metotları kullanılır. Bu metotlar parametre olarak bizden durumu (observation) alırlar ve bize uygulacak eylemi verirler biz de bu işlemleri bir döngü içerisinde yaparak etmeni uygun biçimde çalıştırımız oluruz.

Örneğin:

```
import gym

from stable_baselines.deepq.policies import MlpPolicy
from stable_baselines import DQN

env = gym.make('CartPole-v1')

model = DQN(MlpPolicy, env, verbose=0)
model.learn(total_timesteps=100000)

obs = env.reset()
for i in range(2000):
```

```

action, _states = model.predict(obs)
obs, reward, done, info = env.step(action)
env.render()
print(i, end=' ')
env.close()

```

Burada algoritma olarak DQN (Deep Q-Learning) uygulanmıştır. Kullanılan politika MlpPolicy biçimindedir. Bu politika iki katmanlı her biri 64 nörondan oluşan bir yapay sinir ağı kullanmaktadır. learn fonksiyonunda eğitim için toplam 1000000 yinelemenin yapıldığını görüyoruz. Modelimn parametreleri (epsilon, gamma vs.) default biçimde seçilmiştir. Eğitimden sonra örnek bir deneme yapılmıştır.

Şimdi de MountainCar örneğini uygulayalım. MountainCar için de yukarıdaki programın neredeyse aynısı kullanılabilir. Örneğin:

```

import gym

from stable_baselines.deepq.policies import MlpPolicy
from stable_baselines import DQN

env = gym.make('MountainCar-v0')

model = DQN(MlpPolicy, env, verbose=0)
model.learn(total_timesteps=100000)

obs = env.reset()
i = 0
while True:
    action, _states = model.predict(obs)
    obs, reward, done, info = env.step(action)
    if done:
        break
    env.render()
    print(i, end=' ')
    i += 1

env.close()

```

Model eğitildikten sonra saklanıp geri yüklenebilir. Modeli saklamak için model sınıfının save isimli metodu kullanılır. Örneğin:

```
model.save('mountaincar')
```

Bu metot modeli bir .zip dosyası içerisinde saklamaktadır. Modeli geri yüklemek için ise algoritma sınıflarının statik load metodları kullanılmaktadır. Örneğin:

```
model = DQN.load('mountaincar')
```

Örneğin yukarıdaki MountainCar simülasyonunda eğitim sonrası değerleri saklayıp geri yükleyelim:

```

import gym

from stable_baselines.deepq.policies import MlpPolicy
from stable_baselines import DQN

env = gym.make('MountainCar-v0')

model = DQN(MlpPolicy, env, verbose=0)
model.learn(total_timesteps=100000)

model.save('mountaincar')

```

```

del model # model nesnesi kasti olarak siliniyor.

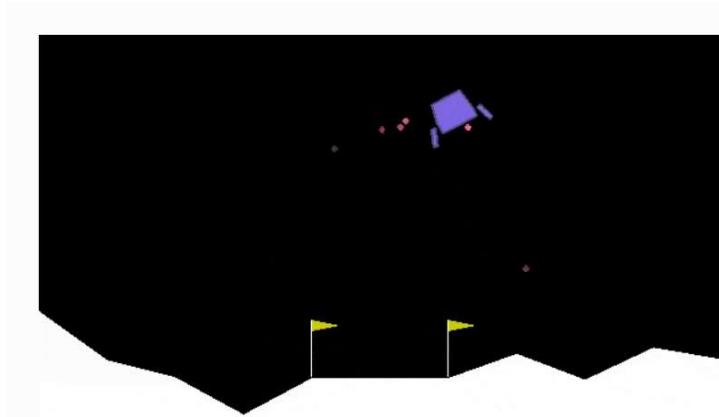
model = DQN.load('mountaincar')

obs = env.reset()
i = 0
while True:
    action, _states = model.predict(obs)
    obs, reward, done, info = env.step(action)
    if done:
        break
    env.render()
    print(i, end=' ')
    i += 1

env.close()

```

Stable-Baselines dokğanlarında içerisinde çeşitli örnekler vardır. Örneğin LunarLanding simülörü ayda inmeye çalışan örümceği takit eder. Bu simülörde amaç örümceğin iki bayrak arasına indirilmesidir. Örümcek sağa sola hareket ettirilebilmektedir.



Modelin eğitimi ve örnek uygulaması şöyle yapılmaktadır:

```

import gym
from stable_baselines import DQN
from stable_baselines.common.evaluation import evaluate_policy

# Create environment
env = gym.make('LunarLander-v2')

# Instantiate the agent
model = DQN('MlpPolicy', env, learning_rate=1e-3, prioritized_replay=True, verbose=1)
# Train the agent
model.learn(total_timesteps=int(2e5))

# Evaluate the agent
mean_reward, std_reward = evaluate_policy(model, model.get_env(), n_eval_episodes=10)
print(mean_reward, std_reward)

# Enjoy trained agent
obs = env.reset()
for i in range(1000):
    action, _states = model.predict(obs)
    obs, rewards, dones, info = env.step(action)
    env.render()

```

```
env.close()
```

Stable-Baselines İçin Custom Environment Yazımı

Örneklerden de görüldüğü gibi stable-baselines OpenAI gym ortamını temel almaktadır. Gerçekten de model sınıfının learn metodu bizden bir environment nesnesi istemektedir. Pekiyi biz kendi problemlerimizi bu kütüphaneye nasıl çözebiliriz? İşte bunun için bizim gym uyumlu environment nesnesi oluşturmamız gereklidir.

Gym uyumlu environment nesnesi elde edebilmek için öncelikle gym.Env sınıfından türetme yapılarak bir environment sınıfının oluşturulması gereklidir. Örneğin:

```
import gym

class MyEnv(gym.Env):
    pass
```

Bizim bu kendi environment sınıfımız için birkaç metod ve örnek özniteliğini yazmamız gereklidir. Bunlar şöyledir:

Örnek öznitelikleri: action_space, observation_space
Metotlar: __init__, reset, step, render

Örneğin:

```
import gym

class MyEnv(gym.Env):
    metadata = {'render.modes': ['human']}

    def __init__(self):
        super().__init__()
        # ....
        self.action_space = < actions >
        self.observation_space = < observations >

    def reset(self):
        pass

    def step(self):
        pass

    def render(self):
        pass
```

Internet'te çeşitli custom environment örnekleri bulunmaktadır. Gym'in üçüncü parti simülatörleri de bu amaçla incelenebilir.