

Makine Öğrenmesi Algoritmaları

Dr. Cahit Karakuş

“Balbiti”

İçindekiler

Giriş.....	3
1. Makine Öğrenmesi Algoritmalarının Temel Bileşenleri	4
1.1. Makine Öğrenmesi Algoritmaları	5
1.2. Makine Öğreniminde Yanlılık(Bias) ve Varyans	11
2. Kümeleme (Clustering)	17
2.1.1. Hiyerarşik Kümeleme	26
2.1.2. K-ortalama	32
2.1.3. Temel Bileşen Analizi (Principal Component Analysis - PCA)	37
3. Regresyon Algoritmaları	48
3.1.1. Doğrusal Regrasyon	50
3.1.2. Çok Değişkenli Doğrusal Regresyon	65
3.1.3. Polynomial Regression	83
3.1.4. Hypothesis Function for Polynomial Regression	90
3.1.5. Üssel Regrasyon	91
3.1.6. Sinusoidal Regression	93
3.1.7. Logarithmic Regression	94
4. Lojistik Regresyon	96
5. Sınıflandırma.....	111
5.1.1. Karar Ağaçları	112
5.1.2. Önyükleme Toplama (Torbalama - Bagging).....	136
5.1.3. K-NN En Yakın Komşu	144
5.1.4. Destek Vektör Makineleri	163
5.1.5. Naive Bayes.....	173
5.1.6. Rassal Orman Modeli	177
6. Derin Öğrenme Algoritmaları	188
6.1.1. Anomali Tespiti	191
6.1.2. Yapay Sinir Ağları	192
6.1.3. Types of Algorithms used in Deep Learning	203
7. Genetik Algoritmalar	233
8. Python Yükleme.....	243
9. Yararlanılan Kaynaklar	244

Giriş

İçinde bilgisayar bulunan makinelerin minimum insan müdahalesi ile bir görevi öğrenmesini sağlamak için verilere, algoritmalara ve matematiksel modellere odaklanan bir yapay zeka dalına Makine öğrenimi denir. Makine öğrenimi, deneyimle veri yığınınından otonom öğrenmek ve performansını geliştirmek için sistem programlamayla ilgilenen bir bilgisayar bilimi dalıdır.

Makine öğreniminin yaptığı şey, verileri farklı algoritmalarla işlemektir ve bize bunun bir kedi mi yoksa köpek mi olduğunu belirlemede hangi özelliğin daha önemli ve etkin olduğunu söyler. O halde sonuca etkisi fazla olan parametrelerin belirlenmesi çok daha önemsenmelidir. Bu nedenle, birçok kural seti uygulamak yerine, bunu iki veya üç özelliğe göre basitleştirebiliriz ve sonuç olarak bu bize daha yüksek bir doğruluk sağlar.

Makine Öğrenimi, kurallara dayalı çıkarımlar yapar. İlişkilendirme kuralı öğrenme, büyük veri tabanlarındaki değişkenler arasındaki ilişkileri keşfetmek için kural tabanlı bir makine öğrenme yöntemidir. Makine öğrenmede veritabanlarında keşfedilen güçlü kuralların bir miktar "ilginçlik" ölçüsü kullanılarak belirlenmesi amaçlanmıştır.

Makine öğrenmesi, bilgisayarları açıkça programlanmaksızın görevleri nasıl gerçekleştirebileceklerini keşfetmeyi içerir. Belirli görevleri yerine getirmeleri için verilerden öğrenen algoritmaları içerir. Yapısal işlev olarak veri yığınınından öğrenebilen ve veriler üzerinden karar vermeye yönelik tahmin yapabilen algoritmaların çalışma ve inşalarını araştıran bir sistemdir. Makine Öğrenmesi uygulamasında, programcıların gereken her adımı belirtmesinden ziyade makinenin kendisinin algoritmaları geliştirmesine yardımcı olmaktadır.

Kural tabanlı öğrenme, bilgiyi depolamak, manipüle etmek veya uygulamak için "kuralları" tanımlayan, öğrenen veya geliştiren herhangi bir makine öğrenme yöntemi için genel bir terimdir. Kural tabanlı öğrenme algoritmasının belirleyici özelliği, sistem tarafından yakalanan bilgileri topluca temsil eden bir dizi ilişkisel kuralın tanımlanması ve kullanılmasıdır. Bu, bir tahminde bulunmak için herhangi bir örneğe evrensel olarak uygulanabilen tekil bir modeli yaygın olarak tanımlayan diğer makine öğrenme algoritmalarının aksine, kural tabanlı makine öğrenme yaklaşımları, öğrenme sınıflandırıcı sistemleri, ilişkilendirme kuralı öğrenme ve yapay bağıklık sistemlerini içerir.

Bilgisayarlara otonom olarak görevlerini nasıl gerçekleştireceklerini öğreten makine öğrenmesi algoritmaları ve matematiksel modelleri geliştirmede kullanılan yörünge denklemleri geniş bir uygulama alanı bulmaktadır.

Profesör Andrew Moore tarafından tanımlanan yapay zeka (AI), yakın zamana kadar insan zekası gibi bilgisayarların davranış geliştirmesini sağlayan matematik ve mühendislik alanıdır. Bunlar şunları içerir:

- Bilgisayar görüşü
- Dil işleme
- Yaratıcılık
- Özetleme

Profesör Tom Mitchell tarafından tanımlandığı gibi, makine öğrenimi, bilgisayar programlarının deneyim yoluyla otomatik olarak öğrenmesine izin veren bilgisayar algoritmaları çalışmasına odaklanan yapay zekanın bilimsel bir dalını ifade eder. Bunlar şunları içerir:

- Sınıflandırma
- Sinir ağı
- Kümeleme

Derin öğrenme modeli, verinin yapısına göre filtreleme yapmak amacıyla hangi parametrelere ne ağırlık verileceğini kendisinin keşfetmesidir. Derin öğrenme algoritması da veriye dayalı öğrenme gerçekleştirmekle birlikte, öğrenme süreci standart makine öğrenmesi algoritmalarında olduğu gibi tek bir matematiksel modele değil sinirsel ağ (neural network) olarak ifade edilen ağ diyagramlarına benzeyen yapıda geliştirilen hesaplamalarla çalışmaktadır. Her derin öğrenme algoritması bir makine öğrenmesi algoritmasıdır çünkü verilerden öğrenme gerçekleştirmektedir. Ancak her makine öğrenmesi algoritması derin öğrenme algoritması değildir; nitekim derin öğrenme, makine öğrenmesinin spesifik bir türüdür. Derin öğrenme, Yüksek bilgi işleme gücü ve büyük veri kümeleriyle birlikte katmanlı yapay sinir ağlarının güçlü matematiksel modelleri oluşturabildiği bir makine öğrenimi alt kümesidir.

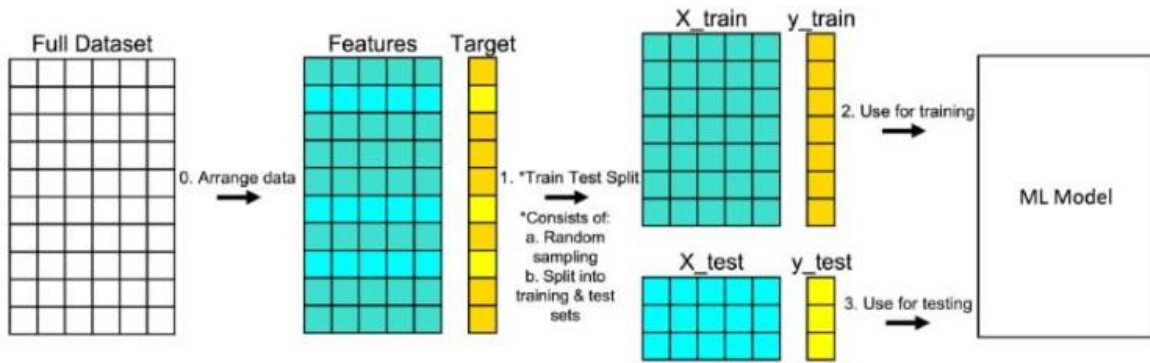
1. Makine Öğrenmesi

Algoritmalarının Temel Bileşenleri

Makine Öğrenmesi algoritmaları çalıştırılırken,

- Hesaplamalar için çeşitli kütüphaneler kullanıldığından gerekli kütüphaneler aktarılır.
- Veritabanı dosyaları okunur.

- Etiketli verilerin ağırlıklı ortalama, standart sapma gibi istatistiksel analiz özellikleri belirlenir ve yorumlanır.
- **Verilerin grafiği çizilir.**
- Değerler tahmin edilirken dikkate almak istenilen özellikler ve katsayılar seçilir.
- Bir modelin doğruluğunu kontrol etmek için, veriler eğitim ve test veri setlerine ayrılır.
- Model eğitilir
- Test veri seti için bir tahmin fonksiyonu bulunur.
- Test verilerinin doğruluğu kontrol edilir: Gerçek değerleri veri setindeki tahmin edilen değerlerle karşılaştırılarak bir modelin doğruluğu kontrol edilebilir.



Makine Öğreniminin beş popüler algoritması:

- Karar ağaçları
- Sinir Ağları (geri yayılım)
- Olasılık ağları
- En yakın komşu
- Vektör makineleri desteklemek

1.1. Makine Öğrenmesi Algoritmaları

1- Regresyon (Tahmin)

Sürekli değerleri tahmin etmek için regresyon algoritmaları kullanılır.

Regresyon algoritmaları:

- Doğrusal Regresyon
- Polinom Regresyon
- Üstel Regresyon

- Lojistik regresyon
- Logaritmik Regresyon

2-Sınıflandırma

Bir dizi ögenin sınıfını veya kategorisini tahmin etmek için sınıflandırma algoritmaları kullanılır.

Sınıflandırma algoritmaları:

- K-En Yakın Komşular
- Karar ağaçları
- Rastgele Orman
- Destek Vektör Makinesi
- Naive Bayes

3- Kümeleme

Özetlemek veya verileri yapılandırmak için kümeleme algoritmaları kullanılır.

Kümeleme algoritmaları:

- K-means
- DBSCAN
- Mean Shift
- Hierarchical

4- İlişkilendirme

Birlikte meydana gelen öğeleri veya olayları ilişkilendirmek için ilişkilendirme algoritmaları kullanıyoruz.

İlişkilendirme algoritmaları:

- Apriori

5- Anomali (Sapma) Algılama:

Anormal etkinlikleri ve dolandırıcılık tespiti gibi olağandışı durumları keşfetmek için anormallik algılama kullanılır. **Veri madenciliğinde, aykırı tespit olarak da bilinen anomali tespiti, verilerin çoğundan önemli ölçüde farklılık göstererek şüphe uyandıran nadir maddelerin, olayların veya gözlemlerin tanımlanmasıdır.** Tipik olarak, anormal kelimeler banka sahtekarlığı, yapısal bir kusur, tıbbi sorunlar veya bir metindeki hatalar gibi bir konuyu temsil eder. Anomalilere aykırı değerler, yenilikler, gürültü, sapmalar ve istisnalar denir.

Özellikle, kötüye kullanım ve ağ izinsiz giriş tespiti bağlamında, ilginç nesneler genellikle nadir nesneler değil, faaliyette beklenmedik patlamalardır. Bu model, bir aykırı değer nadir bir nesne olarak ortak istatistiksel tanımına uymaz ve uygun şekilde toplanmadığı sürece birçok aykırı algılama yöntemi (özellikle denetimsiz algoritmalar) bu verilerde başarısız olur. Bunun yerine, bir küme analiz algoritması bu örüntüler tarafından oluşturulan mikro kümeleri tespit edebilir.

Üç geniş anomali tespit tekniği kategorisi bulunmaktadır. Gözetimsiz anomali tespit teknikleri, veri kümesindeki örneklerin çoğunun normal olduğu varsayımıyla etiketlenmemiş bir test veri kümesindeki anormallikleri, veri kümesinin geri kalan kısmına en az uyan görünen örnekleri arayarak tespit eder. Denetimli anomali algılama teknikleri, "normal" ve "anormal" olarak etiketlenmiş ve bir sınıflandırıcıyı (diğer birçok istatistiksel sınıflandırma problemi için temel fark, aykırı algılamanın doğasında dengesiz doğasıdır) içeren bir veri seti gerektirir. Yarı denetimli anomali tespit teknikleri, belirli bir normal eğitim veri setinden normal davranışı temsil eden bir model oluşturur ve daha sonra model tarafından bir test örneğinin üretilme olasılığını test eder.

6- Sıra Desen Madenciliği

Örüntü - Pattern: Bir nesnenin ya da olayın iki veya üç boyutlu, uzaysal ve geometriksel davranışının matematiksel ifadesidir. Diğer bir ifadeyle , nesnenin davranışı ile ilgili uzayda gözlemlenebilir veya ölçülebilir geometrik bilgilerdir. Tersinden desen madenciliği. Örüntü hazırlanır. Veri yığını içerisinde gezinir. Hedef geldiğinde uyarır, kendisini kabul ettirir. Ayrıca veri yığı içerisinde dolaşan ve arayan desenler geliştirilmektedir.

Örüntü Tanımanın kullanıldığı alanlar:

- Örüntü Tanıma şu alanlarda kullanılabilir:
- Bilgisayar görüşü
- Konuşma tanıma
- Veri madenciliği
- İstatistik
- Gayri Resmi Erişim
- Biyo-Bilişim

Bir dizideki veri örnekleri arasındaki sonraki veri olaylarını tahmin etmek için sıralı model madenciliği kullanıyoruz.

Dünya ve ötesinde kainat sürekli veri oluşturan sinyal kaynakları (Elektromanyetik, elektrik, ısı, renk, ses, titreşim, çekimsel kuvvetler, ...) ile doludur. Bu kaynaklarından yayılan sinyal örüntüleri yayıldıkları ortamlar ile etkileşime girerler. O halde bu örüntülerin davranış değişikliklerinden kestirimsel tahminler yapılabilir.

7- Boyut Küçültme (Dimensionality reduction)

Makine Öğrenimi ve istatistikte boyut küçültme, dikkate alınan rastgele değişkenlerin sayısını azaltma işlemidir ve özellik seçimi ve özellik çıkarımı olarak ikiye ayrılabilir.

Bir veri kümesinden yalnızca yararlı özellikleri çıkarmak için verilerin boyutunu küçültmek için boyut azaltma kullanılır. Boyut azaltma, denetimsiz bir öğrenme tekniğidir.

Veri biliminde, boyut indirgeme, bir verinin yüksek boyutlu bir uzaydan, düşük boyutlu bir uzaya, anlamını kaybetmeyecek şekilde dönüştürülmesidir. Yüksek boyutlu bir veriyi işlemek daha fazla işlem yükü gerektirir. Bu yüzden, yüksek sayıda gözlemin ve değişkenin incelendiği sinyal işleme, konuşma tanıma, nöroinformatik, biyoinformatik gibi alanlarda boyut indirmesi sıkça kullanılır.

Boyut indirgeme yaklaşımları doğrusal ve doğrusal olmayan olarak ikiye ayrılır. Boyut indirgeme var olan özniteliklerin bir alt kümesini seçerek ya da yeni öznitelikler çıkararak yapılabilir. Boyut indirgemesi gürültü filtreleme, veri görselleştirme ya da kümeleme analizi amacıyla kullanılabileceği gibi, diğer makine öğrenimi yöntemlerinin ön adımı olarak uygulanabilir.

8- Önerilerden eğilim ya da yön bulma

Öneri motorları oluşturmak için öneri algoritmalarını kullanılır.

Örnekler:

- Netflix öneri sistemi.
- Bir kitap tavsiye sistemi.
- Amazon'da bir ürün öneri sistemi.

Günümüzde yapay zeka, makine öğrenimi, derin öğrenme ve diğerleri gibi pek çok vızıltılı kelime duyuyoruz.

Makine öğrenimi algoritmalarını uygulamak için neden Python?

Python, popüler ve genel amaçlı bir programlama dilidir. Python kullanarak makine öğrenimi algoritmaları yazabiliriz ve bu iyi çalışır. Python'un veri bilimcileri arasında bu kadar popüler olmasının nedeni, **Python'un çeşitli modül ve kütüphanelerin halihazırda uygulanmış olmasıdır.**

Bazı heyecan verici Python kütüphanelerine kısaca bir göz atalım:

1. Numpy: Python'da n boyutlu dizilerle çalışmak için bir matematik kitaplığıdır. Hesaplamaları etkili ve verimli bir şekilde yapmamızı sağlar.
2. Scipy: Sinyal işleme, optimizasyon, istatistik ve çok daha fazlasını içeren sayısal algoritmalar ve alana özgü araç kutusudur. Scipy, bilimsel ve yüksek performanslı hesaplamalar için işlevsel bir kitaplıktır.
3. Matplotlib: 2D çizimin yanı sıra 3D çizim de sağlayan modaya uygun bir çizim paketidir.
4. Scikit-learn: Python programlama dili için ücretsiz bir makine öğrenimi kitaplığıdır. Sınıflandırma, regresyon ve kümeleme algoritmalarının çoğuna sahiptir ve Numpy, Scipy gibi Python sayısal kütüphaneleri ile çalışır.

Types of Machine Learning

Supervised Learning

Classification

- Fraud detection
- Email Spam Detection
- Diagnostics
- Image Classification

Regression

- Risk Assessment
- Score Prediction

Unsupervised Learning

Dimensionality Reduction

- Text Mining
- Face Recognition
- Big Data Visualization
- Image Recognition

Clustering

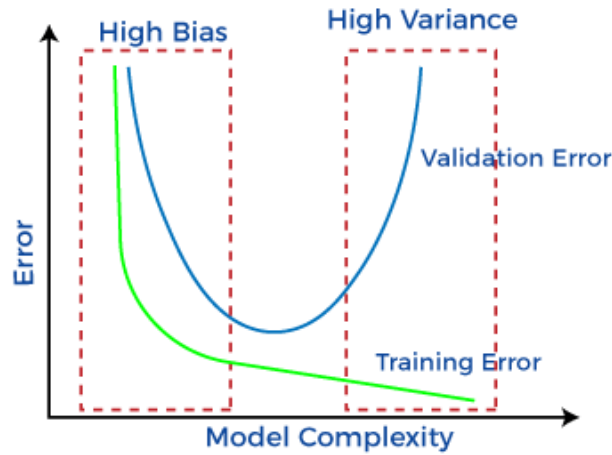
- Biology
- City Planning
- Targetted Marketing

Reinforcement Learning

- Gaming
- Finance Sector
- Manufacturing
- Inventory Management
- Robot Navigation

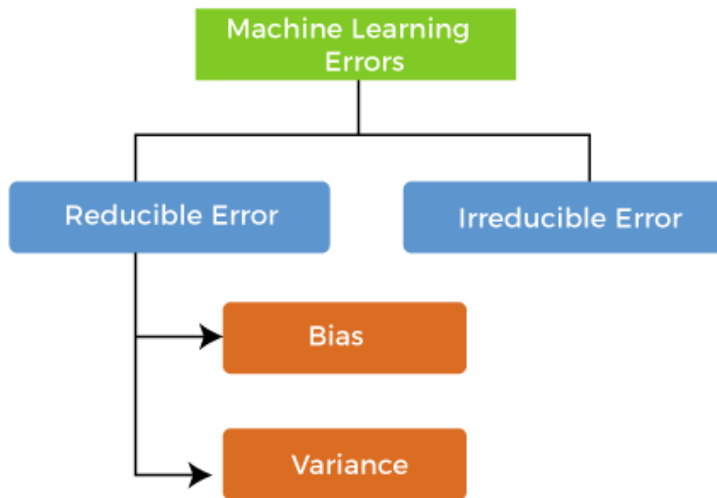
1.2. Makine Öğreniminde Yanlılık(Bias) ve Varyans

Makine öğrenimi, makinelerin veri analizi yapmasına ve tahminler yapmasına izin veren Yapay Zekanın bir dalıdır. **Makine öğrenimi modeli doğru değilse tahmin hataları yapabilir ve bu tahmin hataları genellikle Yanlılık ve Varyans olarak bilinir.** Makine öğreniminde, model tahminleri ile gerçek tahminler arasında her zaman küçük bir fark olduğu için bu hatalar her zaman mevcut olacaktır. Makine öğrenimi/veri bilimi analistlerinin temel amacı, daha doğru sonuçlar elde etmek için bu hataları azaltmaktır.



Makine Öğrenimindeki Hatalar?

Makine öğreniminde hata, bir algoritmanın önceden bilinmeyen veri kümesi için ne kadar doğru tahminler yapabileceğinin bir ölçüsüdür. **Makine öğrenmesinde hatalar temelinde, belirli veri kümesinde en iyi performansı gösterebilecek makine öğrenimi modeli seçilir.** Makine öğreniminde temel olarak iki tür hata vardır, bunlar:



- **İndirgenebilir hatalar:** Bu hatalar, model doğruluğunu artırmak için azaltılabilir. Bu tür hatalar ayrıca önyargı ve Varyans olarak sınıflandırılabilir.
- **İndirgenemez hatalar:** Bu hatalar, hangi algoritma kullanılırsa kullanılsın modelde her zaman mevcut olacaktır. Bu hataların nedeni, değeri azaltılamayan bilinmeyen değişkenlerdir.

Yanlılık nedir?

Genel olarak, bir makine öğrenimi modeli verileri analiz eder, içindeki kalıpları bulur ve tahminler yapar. Model, eğitim sırasında veri kümesindeki bu kalıpları öğrenir ve bunları tahmin için test verilerine uygular. Öngörü yapılırken, model tarafından yapılan tahmin değerleri ile gerçek değerler/beklenen değerler arasında bir fark oluşur ve bu fark, yanlılık hataları veya Önyargıdan kaynaklanan hatalar olarak bilinir. Doğrusal Regresyon gibi makine öğrenimi algoritmalarının veri noktaları arasındaki gerçek ilişkiyi yakalayamaması yanlılık olarak tanımlanabilir. Her algoritma bir miktar yanlılıkla başlar, çünkü yanlılık modeldeki varsayımlardan kaynaklanır ve bu da hedef işlevin öğrenilmesini kolaylaştırır. Bir model aşağıdakilerden birine sahiptir:

- **Düşük Sapma:** Düşük sapma modeli, hedef fonksiyonun biçimi hakkında daha az varsayımda bulunacaktır.
- **Yüksek Sapma:** Yüksek sapmaya sahip bir model, daha fazla varsayımda bulunur ve model, veri kümemizin önemli özelliklerini yakalayamaz hale gelir. Yüksek sapmalı bir model, yeni veriler üzerinde de iyi performans gösteremez.

Genel olarak, doğrusal bir algoritma, hızlı öğrenmelerini sağladığı için yüksek yanlılığa (önyargıya) sahiptir. Algoritma ne kadar basitse, tanıtılması muhtemel önyargı o kadar yüksek olur. Oysa doğrusal olmayan bir algoritma genellikle düşük önyargıya sahiptir.

Karar Ağaçları, k-En Yakın Komşular ve Destek Vektör Makineleri, düşük önyargılı makine öğrenimi algoritmalarına bazı örneklerdir. Aynı zamanda yüksek yanlılığa sahip bir algoritma Lineer Regresyon, Lineer Diskriminant Analizi ve Lojistik Regresyon'dur.

Yüksek Önyargıyı azaltmanın yolları:

Yüksek yanlılık esas olarak çok basit bir modelden kaynaklanır. Yüksek yanlılığı azaltmanın bazı yolları aşağıda verilmiştir:

- Model yetersiz takıldıkça giriş özelliklerini artırın.
- Düzenleştirme terimini azaltın.
- Bazı polinom özelliklerini dahil etmek gibi daha karmaşık modeller kullanın.

Varyans Hatası Nedir?

Varyans, farklı eğitim verileri kullanılmışsa tahmindeki varyasyon miktarını belirtir. Basit bir ifadeyle varyans, rastgele bir değişkenin beklenen değerinden ne kadar farklı olduğunu söyler. İdeal olarak, bir model bir eğitim veri setinden diğerine çok fazla değişmemelidir, bu da algoritmanın girdiler ve çıktı değişkenleri arasındaki gizli eşlemeyi anlamada iyi olması gerektiği anlamına gelir. Varyans hataları, düşük varyanslı veya yüksek varyanslıdır.

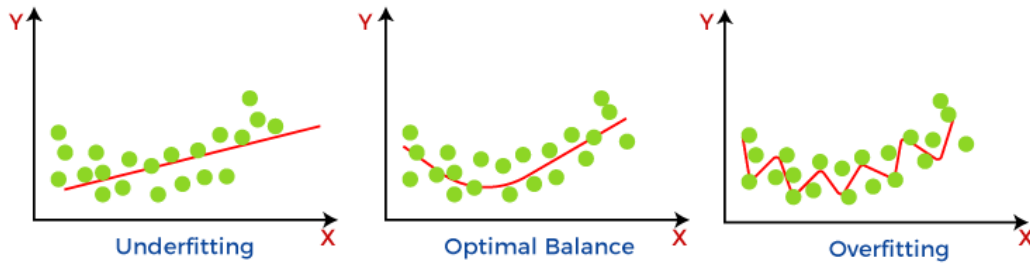
Düşük varyans, eğitim veri setindeki değişikliklerle hedef fonksiyonun tahmininde küçük bir varyasyon olduğu anlamına gelir. Aynı zamanda, Yüksek varyans, eğitim veri setindeki değişikliklerle hedef fonksiyonun tahmininde büyük bir varyasyon gösterir.

Yüksek varyans gösteren bir model, çok şey öğrenir ve eğitim veri kümesiyle iyi performans gösterir ve görünmeyen veri kümesiyle iyi genelleme yapmaz. Sonuç olarak, böyle bir model, eğitim veri setinde iyi sonuçlar verirken, test veri setinde yüksek hata oranları göstermektedir.

Model, yüksek varyans ile veri kümesinden çok fazla şey öğrendiğinden, modelin aşırı uyumuna yol açar. Yüksek varyansa sahip bir modelde aşağıdaki problemler vardır:

- o Yüksek varyanslı bir model fazla uydurmaya yol açar.
- o Model karmaşıklığını artırır.

Genellikle, doğrusal olmayan algoritmalar, modele uyma konusunda çok fazla esnekliğe sahiptir ve yüksek varyansa sahiptir.



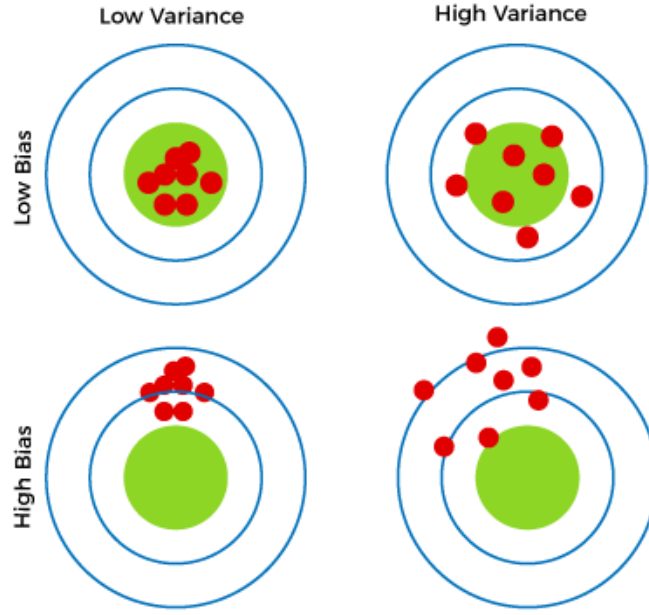
Düşük varyansa sahip makine öğrenimi algoritmalarına bazı örnekler, Lineer Regresyon, Lojistik Regresyon ve Lineer diskriminant analizidir. Aynı zamanda, yüksek varyansa sahip algoritmalar karar ağacı, Destek Vektör Makinesi ve K-en yakın komşulardır.

Yüksek Varyansı Azaltma Yolları:

- o Bir model aşırı donatıldığı için giriş özelliklerini veya parametre sayısını azaltılır.
- o Çok karmaşık bir model kullanılmaz.
- o Eğitim verileri artırılır.
- o Düzenleme süresi artırılır.

Bias-Varyance'ın Farklı Kombinasyonları

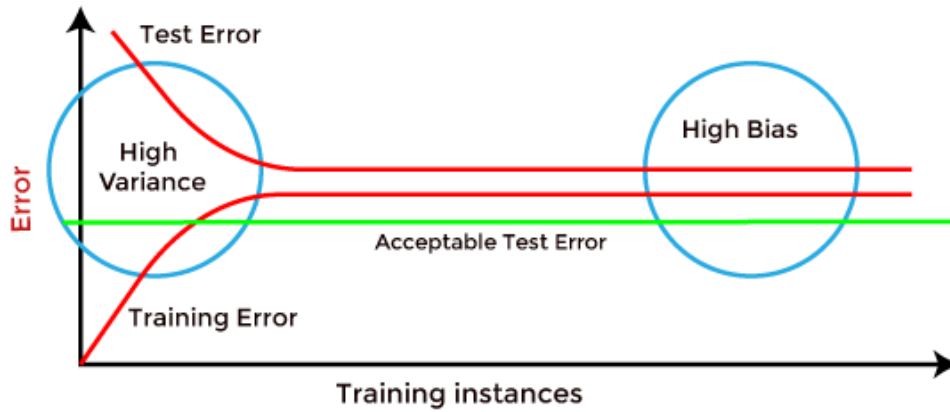
Aşağıdaki diyagramda gösterilen dört olası yanlışlık ve varyans kombinasyonu vardır:



- 1) **Düşük Sapma, Düşük Varyans:** Düşük önyargı ve düşük varyansın kombinasyonu, ideal bir makine öğrenimi modeli gösterir. Ancak pratikte mümkün değil.
- 2) **Düşük Sapma, Yüksek Varyans:** Düşük sapma ve yüksek varyansla, model tahminleri ortalama olarak tutarsız ve doğrudur. Bu durum, model çok sayıda parametre ile öğrendiğinde ortaya çıkar ve bu nedenle aşırı uydurmaya yol açar.
- 3) **Yüksek Sapma, Düşük Varyans:** Yüksek sapma ve düşük varyans ile tahminler tutarlıdır ancak ortalama olarak yanlıştır. Bu durum, bir model eğitim veri seti ile iyi öğrenmediğinde veya parametrenin birkaç sayısını kullandığında ortaya çıkar. Modelde yetersiz uyum sorunlarına yol açar.
- 4) **Yüksek Önyargı, Yüksek Varyans:** Yüksek yanlışlık ve yüksek varyans ile tahminler tutarsızdır ve ayrıca ortalama olarak yanlıştır.

Yüksek varyans veya Yüksek Sapma nasıl belirlenir?

Model aşağıdakilere sahipse yüksek varyans tanımlanabilir:



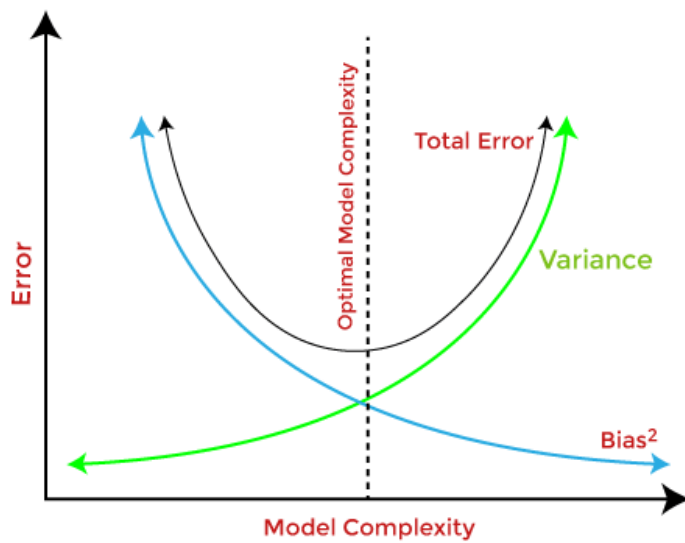
- Düşük eğitim hatası ve yüksek test hatası.

Model aşağıdakilere sahipse Yüksek Sapma belirlenebilir:

- Yüksek eğitim hatası ve test hatası neredeyse eğitim hatasına benzer.

Önyargı-Varyans Takası

Makine öğrenimi modelini oluştururken, modelde gereğinden fazla ve eksik uydurmayı önlemek için yanlılık ve varyansa dikkat etmek gerçekten önemlidir. Model, daha az parametre ile çok basitse, düşük varyansa ve yüksek yanlılığa sahip olabilir. Oysa model çok sayıda parametreye sahipse, yüksek varyansa ve düşük sapmaya sahip olacaktır. Bu nedenle, önyargı ve varyans hataları arasında bir denge kurmak gerekir ve önyargı hatası ile varyans hatası arasındaki bu denge, Bias-Variance trade-off olarak bilinir.



Modelin doğru bir şekilde tahmin edilmesi için, algoritmaların düşük bir varyansa ve düşük sapmaya ihtiyacı vardır. Ancak bu mümkün değildir çünkü yanlılık ve varyans birbiriyle ilişkilidir:

- Varyansı azaltırsak sapmayı artırır.
- Sapmayı azaltırsak, varyansı artıracaktır.

Önyargı-Varyans değiş tokuşu, denetimli öğrenmede merkezi bir konudur. İdeal olarak, eğitim verilerindeki düzenlilikleri doğru bir şekilde yakalayan ve aynı anda görünmeyen veri kümesiyle genelleme yapan bir modele ihtiyacımız var. Ne yazık ki, bunu aynı anda yapmak mümkün değil. Çünkü yüksek varyanslı bir algoritma, eğitim verileriyle iyi performans gösterebilir, ancak gürültülü verilere fazla uydurmaya yol açabilir. Oysa, yüksek yanlılık algoritması, verilerdeki önemli düzenlilikleri bile yakalayamayan çok daha basit bir model üretir. Bu nedenle, optimal bir model yapmak için önyargı ve varyans arasında tatlı bir nokta bulmamız gerekiyor.

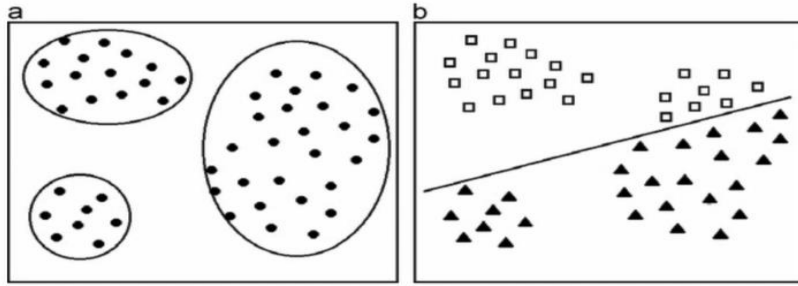
Bu nedenle, Eğilim-Variyans takası, sapma ve varyans hataları arasında bir denge kurmak için tatlı noktayı bulmakla ilgilidir.

2. Kümeleme (Clustering)

Verilerin yakınlık, uzaklık, benzerlik gibi ölçütlere göre analiz edilerek sınıflara ayrılmasına kümeleme denir. Kümeleme, denetimsiz öğrenmenin bir yöntemidir ve birçok alanda kullanılan istatistiksel veri analizi için yaygın bir tekniktir. Denetimsiz öğrenme, veri kümesi ile çıktıların olmadığı bir öğrenme metodudur. Veri kümesindeki verileri yorumlayarak ortak noktaları bulmak ve bunları kümeleştirme işlemi yapılarak anlamlı bir veri elde edebilmektir. Sistem, öğreten olmadan öğrenmeye çalışır. Ham verileri organize verilere dönüştüren bir makine öğrenimi türüdür.

Denetimsiz öğrenmede sadece sınıflandırılmamış veriler vardır bu verilerden sonuçlar çıkarılmaya çalışılır. Veriyi değişkenler arasındaki ilişkilere dayalı olarak kümeleyerek çeşitli modeller, yapılar oluşturulur.

Örneğin, bir alışveriş sitesinde alınan bir ürünün yanında kullanıcıların alabileceği diğer ürünlerin tavsiye olarak belirlenmesi. Ya da bir hizmet satın aldığında, o hizmetle etkileşimli diğer hizmetlerin müşterinin ilgi alanına girmesi.



Kümeleme

Sınıflandırma

Kümeleme algoritmaları, veri kümesindeki bileşenleri kendi aralarında benzerliklerinden gruplamaya çalışır. Burada kaç grup olacağı veya en uygun küme sayısını algoritmanın kendisi belirler.

Örnek: Bir sandık dolusu meyve içerisinde bir tane elmayı belirlesin. Ardından tüm elmaları bulan bir senaryo sizce nasıl olmalıdır? En sonunda sandıkta bir tane elma kalmaması yazdığın senaryonun hangi özelliği sağlamış olur. Performans artmış, deneyim kazanmış olur.

Alışveriş yapılan bir markette kasiyerin bir robot olduğunu düşünün ve tüm ürünler birbirine karışmış olsun. Elma bulup, onu tanıyıp diğer elemanları yığın içerisinde topladığını düşünün. Seçme işlemi devam ederken yetenek kazanarak performansını atırabilir; hatalı seçtiği

ürünler var ise ayıklayabilir. Böylece ürünlerin birbirlerine benzeme yakınlığı uzaklaşarak, ayırım yapma yeteneği artırılmış olur. Böylece sınılandırma da yapılmış olur.

Elmalar da kendi aralarında kümeleme yapılabilir mi? Aynı tipte verilerin değişik segmentlere bölünmüş halidir. Elinde örnekler var ama hangi veya kaç sınıfa ait olduğunu bilmiyor. Sınıfları(kümeleri) kendisi inşaa ediyor. Örneğin elimizde sadece domatesler varsa, ve bunları kalitelerine göre ayırılırsa bu kümeleme işlemidir.

Kümelemenin uygulama alanları:

Tıp'da elde edilen görüntülemeler üzerindeki farklıları analiz edilerek değişik nitelikler çıkartılabilir.

Suç Yerlerinin Belirlenmesi: Bir şehirdeki belirli bölgelerde mevcut olan suçlarla ilgili veriler, suç kategorisi, suç alanı ve ikisi arasındaki ilişki, bir şehirdeki ya da bölgedeki suça eğilimli alanlara ilişkin kaliteli bilgiler verebilir.

Oyuncu istatistiklerini analiz etmek: Oyuncu istatistiklerini analiz etmek, spor dünyasının her zaman kritik bir unsuru olmuştur ve artan rekabetle birlikte, makine öğrenmenin burada oynayacağı kritik bir rol vardır.

Çağrı Kaydı Detay Analizi: Bir çağrı detay kaydı (CDR), telekom şirketleri tarafından bir müşterinin araması, SMS ve internet etkinliği sırasında elde edilen bilgilerdir. Bu bilgiler, müşteri demografisiyle birlikte kullanıldığında, müşterinin ihtiyaçları hakkında daha fazla bilgi sağlar.

Müşteri Segmentasyonu:

Collaborative Filtering: Davranışları birbirine benzeyen insanların yaptıklarına bakılarak kümedeki birinin ne yapacağını kestirmek.

Tehdit ve Sahtekarlık Yakalama: Sınıflandırmada tehditlerin özellikleri makineye öğretilir. Kümelemede ise kümelerin dışında kalan , herhangi bir kümeye girmeyen örnek bir tehdit unsuru olarak tanımlanabilir.

Eksik Verilerin Tanımlanması: Örneğin birinin maaş bilgisi eksikse segmentte benzer kişilerin maaş ortalamasına göre bir maaş hesaplanabilir.

Pazar Segmentasyonu:

Davranışsal Segmentasyon: Örneğin ücretsiz uygulamaları yükleyenler neyi seçiyorlar?

Demografik Segmentasyon: Müşterilerin yaşı, cinsiyeti, eğitim düzeyi ile davranışlarının entegre edilmesi.

Psikolojik Segmentasyon: Müşterilerin hayal-beklentilerine göre farklı ürünler sunulabilir.

Coğrafi Segmentasyon: Ülkelere şehirlere göre vs.

Kümeleme Çeşitleri:

- Hiyerarşik Kümeleme
- Gürültülü Uygulamaların Yoğunluğa Dayalı Konumsal Kümelenmesi (DBSCAN)
- K-means Kümeleme
- Ağırlık Ortalama Kaydırma Kümelemesi
- Gauss Karışım Modelleri (GMM) kullanarak Beklenti-Maksimizasyon (EM) Kümeleme

Ağırlık Ortalama Kaydırma Kümelemesi:

Ortalama kaydırma kümeleme, veri noktalarının yoğun alanlarını bulmaya çalışan kayan pencere tabanlı bir algoritmadır. Centroid tabanlı bir algoritmadır, yani amacın her bir grubun / sınıfın merkez noktalarını bulmaktır, bu da kayan pencere içindeki noktaların ortalaması olacak merkez noktaları için adayları güncelleyerek çalışır. Bu aday pencereler daha sonra, neredeyse kopyaları ortadan kaldırmak için bir işlem sonrası aşamasında filtrelendirir ve nihai merkez noktaları ve bunlara karşılık gelen grupları oluşturur.

Sürgülü pencerelerin tümü ile uçtan uca tüm sürecin bir örneği aşağıda gösterilmiştir. Her siyah nokta, kayan bir pencerenin merkezini temsil eder ve her gri nokta bir veri noktasıdır.

Sınav sorusu: En son ağırlık ortalaması burasıdır denmesi için nasıl bir senaryo hazırlanmalı?



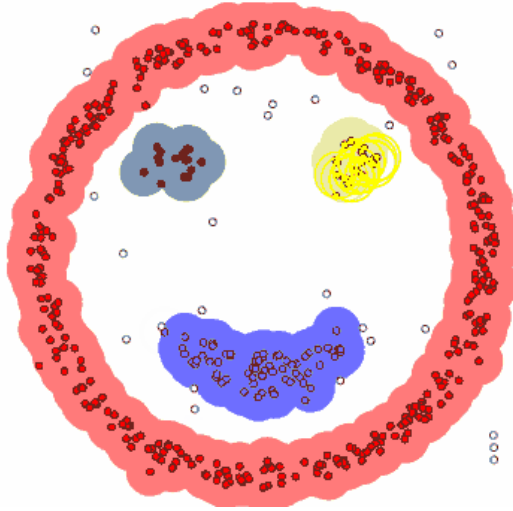
- 1) Ortalama-kaymayı açıklamak için, yukarıdaki resimde olduğu gibi iki boyutlu uzayda bir dizi noktayı ele alacağız. Bir C noktasında ortalanmış (rastgele seçilmiş) ve çekirdek olarak r yarıçapına sahip dairesel bir kayan pencere ile başlıyoruz. Ortalama kayma, bu çekirdeği yakınsamaya kadar her adımda yinelemeli olarak daha yüksek yoğunluklu bir bölgeye kaydırmayı içeren bir tepe tırmanma algoritmasıdır.
- 2) Her yinelemede, kayan pencere, merkez noktası pencere içindeki noktaların ortalamasına kaydırılarak (dolayısıyla adı) daha yüksek yoğunluklu bölgelere kaydırılır. Sürgülü pencere içindeki yoğunluk, içindeki noktaların sayısı ile orantılıdır.

Doğal olarak, penceredeki noktaların ortalamasına geçerek, yavaş yavaş daha yüksek nokta yoğunluğuna sahip alanlara doğru hareket edecektir.

- 3) Bir kaymanın çekirdek içinde daha fazla noktayı barındırabileceği bir yön olmayana kadar ortalamaya göre kayan pencereyi kaydırmaya devam ediyoruz. Yukarıdaki grafiğe bakın; Artık yoğunluğu artırmayana kadar (yani penceredeki nokta sayısı) daireyi hareket ettirmeye devam ediyoruz.
- 4) 1'den 3'e kadar olan bu adım süreci, tüm noktalar bir pencerenin içinde kalana kadar birçok sürgülü pencerede yapılır. Birden çok sürgülü pencere örtüştüğünde, en çok noktayı içeren pencere korunur. Veri noktaları daha sonra bulundukları kayan pencereye göre kümelenir.

Gürültülü Uygulamaların Yoğunluğa Dayalı Konumsal Kümelenmesi (DBSCAN):

DBSCAN, ortalama kaymaya ekseninde, benzer ve yoğunluklu bölgeleri kümeleyen bir algoritmadır, ancak birkaç önemli avantajı vardır. Minimum bölge sayısında ve uzaklıkta maksimum yoğunluk bölgesi oluşturulması hedeflenir.



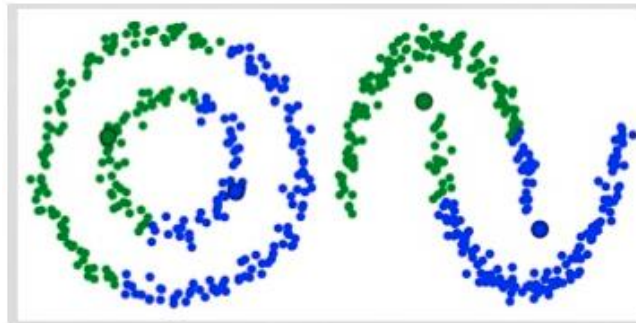
- DBSCAN, keyfi bir başlangıç veri noktasıyla başlar. Bu noktanın komşuluğu bir epsilon ϵ uzaklığı kullanılarak çıkarılır (ϵ mesafesi içindeki tüm noktalar komşuluk noktalarıdır).
- Bu mahalle içinde yeterli sayıda nokta (minPoints'e göre) varsa, kümeleme işlemi başlar ve mevcut veri noktası yeni kümedeki ilk nokta olur. Aksi takdirde, nokta gürültü olarak etiketlenecektir (daha sonra bu gürültülü nokta kümenin parçası haline gelebilir). Her iki durumda da bu nokta "ziyaret edildi" olarak işaretlenir.

- Yeni kümedeki bu ilk nokta için, ϵ mesafesi komşuluğundaki noktalar da aynı kümenin parçası olur. ϵ mahallesindeki tüm noktaları aynı kümeye ait yapma prosedürü, küme grubuna yeni eklenen tüm yeni noktalar için tekrarlanır.
- 2. ve 3. adımlardan oluşan bu süreç, kümedeki tüm noktalar belirlenene kadar, yani kümenin ϵ mahallesindeki tüm noktalar ziyaret edilip etiketlenene kadar tekrar edilir. Mevcut kümeyle işimiz bittiğinde, yeni bir ziyaret edilmemiş nokta alınır ve işlenir, bu da başka bir küme veya gürültü keşfine yol açar. Bu işlem, tüm noktalar ziyaret edildi olarak işaretlenene kadar tekrar eder.
- Bunun sonunda tüm noktalar ziyaret edildiğinden, her nokta bir kümeye ait veya gürültü olarak işaretlenecektir.

DBSCAN, diğer kümeleme algoritmalarına göre bazı büyük avantajlar sunar. İlk olarak, hiç bir küme gerektirmez. Ayrıca, veri noktası çok farklı olsa bile, aykırı değerleri basitçe bir kümeye atan ortalama kaymanın aksine, gürültü olarak tanımlar. Ek olarak, keyfi olarak boyutlandırılmış ve keyfi olarak şekillendirilmiş kümeleri oldukça iyi bulabilir. DBSCAN'ın ana dezavantajı, kümeler farklı yoğunlukta olduğunda diğerleri kadar iyi performans göstermemesidir. Bunun nedeni, komşu noktaların belirlenmesi için mesafe eşiği ϵ ve minPoints'in, yoğunluk değiştiğinde kümeden kümeye değişmesidir. Bu dezavantaj, çok yüksek boyutlu verilerde de ortaya çıkar, çünkü yine mesafe eşiğini ϵ tahmin etmek zorlaşır.

Gauss Karışım Modelleri (GMM) kullanarak Beklenti-Maksimizasyon (EM) Kümeleme:

K-Ortalamlarının en büyük dezavantajlarından biri, küme merkezi için ortalama değerin naif kullanımıdır. Aşağıdaki resme bakarak bunun bir şeyleri yapmanın en iyi yolu olmadığını anlayabiliriz. Sol tarafta, aynı ortalamaya merkezlenmiş farklı yarıçaplara sahip iki dairesel küme olduğu insan gözüne oldukça açık görünüyor. K-Ortalamlar bunun üstesinden gelemmez çünkü kümelerin ortalama değerleri birbirine çok yakındır. K-Ortalamlar, yine ortalamanın küme merkezi olarak kullanılması sonucunda kümelerin dairesel olmadığı durumlarda başarısız olur.

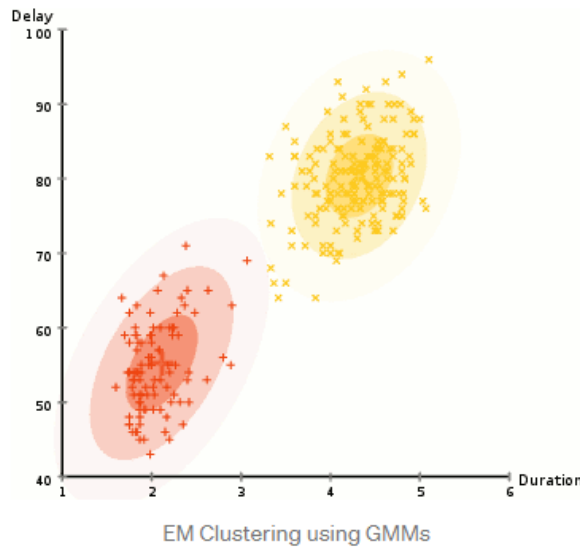


Two failure cases for K-Means

Gauss Karışım Modelleri (GMM'ler) bize K-Ortalamlarından daha fazla esneklik sağlar. GMM'ler ile veri noktalarının Gauss olarak dağıtıldığını varsayıyoruz; bu, ortalamayı

kullanarak döngüsel olduklarını söylemekten daha az kısıtlayıcı bir varsayımdır. Bu şekilde, kümelerin şeklini açıklamak için iki parametremiz var: ortalama ve standart sapma! İki boyutlu bir örnek alırsak, bu, kümelerin her türlü eliptik şekli alabileceği anlamına gelir (çünkü hem x hem de y yönlerinde standart bir sapmaya sahibiz). Böylece, her Gauss dağılımı tek bir kümeye atanır.

Her küme için Gauss'un parametrelerini bulmak için (örneğin ortalama ve standart sapma), Beklenti-Maksimizasyon (EM) adı verilen bir optimizasyon algoritması kullanacağız. Kümelere uydurulan Gaussian'ların bir örneği olarak aşağıdaki grafiğe bir göz atın. Daha sonra GMM'leri kullanarak Beklenti-Maksimizasyon kümeleme sürecine geçebiliriz.



- 1) Küme sayısını seçerek (K-Means'in yaptığı gibi) ve her küme için Gauss dağılım parametrelerini rastgele başlatarak başlıyoruz. Verilere de hızlıca göz atarak ilk parametreler için iyi bir tahmin sağlamaya çalışılabilir. Yine de, yukarıdaki grafikte de görülebileceği gibi, bu% 100 gerekli değildir çünkü Gauss bize çok zayıf olarak başlarlar, ancak hızla optimize edilirler.
- 2) Her küme için bu Gauss dağılımları göz önüne alındığında, her veri noktasının belirli bir kümeye ait olma olasılığı hesaplanır. Bir nokta Gauss'un merkezine ne kadar yakınsa, o kümeye ait olma olasılığı o kadar artar. Gauss dağılımında verilerin çoğunun kümenin merkezine daha yakın olduğunu varsaydığımız için, bu sezgisel bir anlam ifade etmelidir.
- 3) Bu olasılıklara dayanarak, kümeler içindeki veri noktalarının olasılıklarını maksimize edecek şekilde Gauss dağılımları için yeni bir dizi parametre hesaplıyoruz. Bu yeni parametreleri, veri noktası konumlarının ağırlıklı toplamını kullanarak hesaplıyoruz; burada ağırlıklar, o belirli kümeye ait veri noktasının olasılıklarıdır. Bunu görsel olarak açıklamak için yukarıdaki grafiğe, özellikle örnek olarak sarı kümeye bakabiliriz. Dağıtım ilk yinelemede rastgele başlar, ancak sarı noktaların çoğunun bu dağılımın

sağında olduğunu görebiliriz. Olasılıklara göre ağırlıklandırılmış bir toplamı hesapladığımızda, merkeze yakın bazı noktalar olmasına rağmen çoğu sağdadır. Dolayısıyla, doğal olarak dağılımın ortalaması bu noktalar kümesine daha da yakınlaşır. Ayrıca noktaların çoğunun “üstten-sağdan-aşağıya” olduğunu da görebiliriz. Bu nedenle standart sapma, olasılıkların ağırlıklandığı toplamı maksimize etmek için bu noktalara daha uygun bir elips oluşturmak üzere değişir.

- 4) Dağıtımların yinelemeden yinelemeye pek değişmediği yakınsamaya kadar 2. ve 3. adımlar yinelemeli olarak tekrarlanır.

GMM kullanmanın 2 önemli avantajı vardır. Birincisi, GMM'ler küme kovaryansı açısından K-Ortalamalarına göre çok daha esnektir; standart sapma parametresi nedeniyle, kümeler dairelerle sınırlı olmak yerine herhangi bir elips şeklini alabilir. K-Ortalamaları aslında her kümenin tüm boyutlardaki kovaryansının 0'a yaklaştığı özel bir GMM durumudur. İkinci olarak, GMM'ler olasılıkları kullandığından, veri noktası başına birden çok kümeye sahip olabilirler. Dolayısıyla, bir veri noktası üst üste binen iki kümenin ortasındaysa, sınıf 1'e yüzde X ve yüzde Y yüzde 2'ye ait olduğunu söyleyerek sınıfını tanımlayabiliriz. Yani GMM'ler karma üyeliği destekler.

Kümeleme Türleri:

Kümeleme, veri noktalarının benzerlik derecelerine göre farklı kümeler halinde gruplandırıldığı bir denetimsiz öğrenme türüdür.

Çeşitli kümeleme türleri şunlardır:

- Hiyerarşik kümeleme
- Bölümleme kümeleme

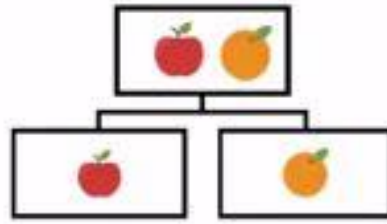
Hiyerarşik kümeleme ayrıca alt bölümlere ayrılır:

- Toplu kümeleme
- Bölücü kümeleme

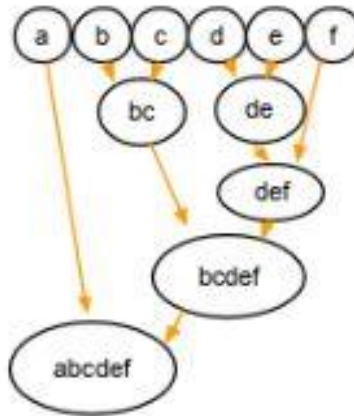
Bölümleme kümeleme ayrıca şu alt bölümlere ayrılır:

- K-Ortalamlar kümeleme
- Bulanık C-Ortalamlar kümeleme

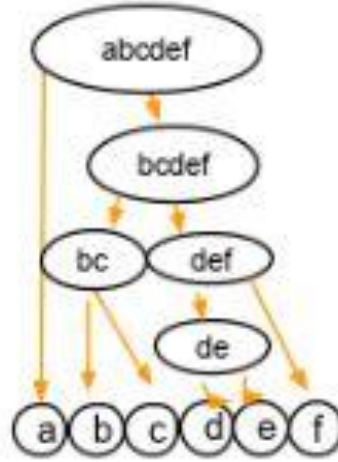
Hiyerarşik kümeleme, aşağıdaki gibi ağaç benzeri bir yapı kullanır:



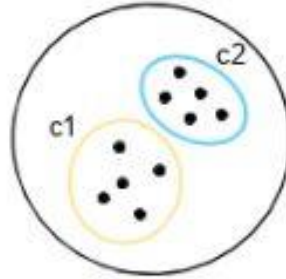
Aglomeratif kümelemede (agglomerative clustering) aşağıdan yukarıya bir yaklaşım vardır. Her bir öğeyle ayrı bir küme olarak başlıyoruz ve bunları aşağıda gösterildiği gibi art arda daha büyük kümeler halinde birleştiriyoruz:



Bölücü kümelem (Divisive clustering) e yukarıdan aşağıya bir yaklaşımdır. Tüm setle başlıyoruz ve aşağıda görebileceğiniz gibi, onu art arda daha küçük kümelere bölmeye devam ediyoruz:

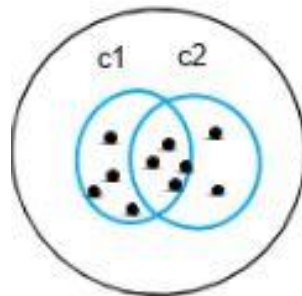


Bölümleme Kümeleme (Partitioning Clustering) iki alt türe ayrılır - K-Ortalamalar kümeleme ve Bulanık C-Ortalamalar. k-ortalama kümelemede, nesneler 'K' sayısı ile belirtilen birkaç kümeye ayrılır. Yani $K = 2$ dersek, nesneler gösterildiği gibi c1 ve c2 olmak üzere iki kümeye ayrılır:



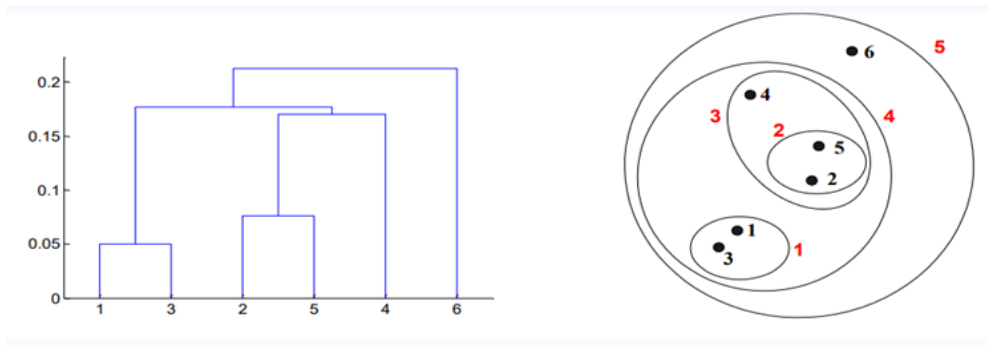
Burada özellikler veya özellikler karşılaştırılır ve benzer özelliklere sahip tüm nesneler birlikte kümelendir.

Bulanık c-ortalamalar (Fuzzy c-means), benzer özelliklere sahip nesneleri bir arada kümelemesi anlamında k-araçlarına çok benzer. k-ortalama kümelemede, tek bir nesne iki farklı kümeye ait olamaz. Ancak c-ortalamada nesneler gösterildiği gibi birden fazla kümeye ait olabilir.



2.1.1. Hiyerarşik Kümeleme

Hiyerarşik kümeleme algoritmaları 2 kategoriye ayrılır: yukarıdan aşağıya veya aşağıdan yukarıya. Aşağıdan yukarıya algoritmalar, her veri noktasını başlangıçta tek bir küme olarak ele alır ve ardından tüm kümeler tüm veri noktalarını içeren tek bir kümede birleştirilene kadar küme çiftlerini art arda birleştirir (veya toplar). Bu nedenle aşağıdan yukarıya hiyerarşik kümeleme, hiyerarşik kümelemeli kümeleme (hierarchical agglomerative clustering) veya HAC olarak adlandırılır. Bu küme hiyerarşisi bir ağaç (veya dendrogram) olarak temsil edilir. Ağacın kökü, tüm örnekleri toplayan benzersiz kümedir, yapraklar yalnızca bir örnek içeren kümelerdir. Algoritma adımlarına geçmeden önce bir örnek için aşağıdaki grafiğe bakın



- Her veri noktasını tek bir küme olarak ele alarak başlıyoruz, yani veri kümemizde X veri noktası varsa, o zaman X kümemiz var. Daha sonra iki küme arasındaki mesafeyi ölçen bir mesafe ölçüsü seçiyoruz. Örnek olarak, iki küme arasındaki mesafeyi, birinci kümedeki veri noktaları ile ikinci kümedeki veri noktaları arasındaki ortalama mesafe olarak tanımlayan ortalama bağlantıyı kullanacağız.
- Her yinelemede, iki kümeyi tek bir kümede birleştiriyoruz. Birleştirilecek iki küme, en küçük ortalama bağlantıya sahip olanlar olarak seçilir. Yani, seçtiğimiz uzaklık ölçütümüze göre, bu iki küme birbirleri arasındaki en küçük mesafeye sahiptir ve bu nedenle en benzer olanlardır ve birleştirilmeleri gerekir.
- Adım 2, ağacın köküne ulaşana kadar tekrar edilir, yani tüm veri noktalarını içeren tek bir kümeye sahibiz. Bu şekilde sonunda kaç tane küme istediğimizi seçebiliriz, sadece kümeleri birleştirmeyi ne zaman durduracağımızı seçerek, yani ağacı oluşturmayı bıraktığımızda!

Hiyerarşik kümeleme, küme sayısını belirlememizi gerektirmez ve hatta bir ağaç oluşturduğumuz için hangi küme sayısının en iyi görüneceğini seçebiliriz. Ek olarak, algoritma mesafe ölçüsü seçimine duyarlı değildir; hepsi eşit derecede iyi çalışma eğilimindeyken, diğer kümeleme algoritmalarında uzaklık ölçüsü seçimi kritiktir. Hiyerarşik kümeleme yöntemlerinin özellikle iyi bir kullanım durumu, temeldeki verilerin hiyerarşik bir yapıya sahip olması ve hiyerarşiyi kurtarmak istemenizdir; diğer kümeleme algoritmaları bunu yapamaz. Hiyerarşik kümelemenin bu avantajları, K-Ortalamalarının ve GMM'nin doğrusal

karmaşıklığından farklı olarak, $O(n^3)$ zaman karmaşıklığına sahip olduğundan, daha düşük verimlilik pahasına gelir.

Hiyerarşik kümeleme, veri noktalarını kümelemek için denetimsiz bir öğrenme yöntemidir. Algoritma, veriler arasındaki farklılıkları ölçerek kümeler oluşturur. Denetimsiz öğrenme, bir modelin eğitilmesi gerekmediği ve bir "hedef" değişkene ihtiyacımız olmadığı anlamına gelir. Bu yöntem, tek tek veri noktaları arasındaki ilişkiyi görselleştirmek ve yorumlamak için herhangi bir veri üzerinde kullanılabilir.

Burada veri noktalarını gruplamak ve hem bir dendrogram hem de dağılım grafiği kullanarak kümeleri görselleştirmek için hiyerarşik kümeleme kullanılacaktır.

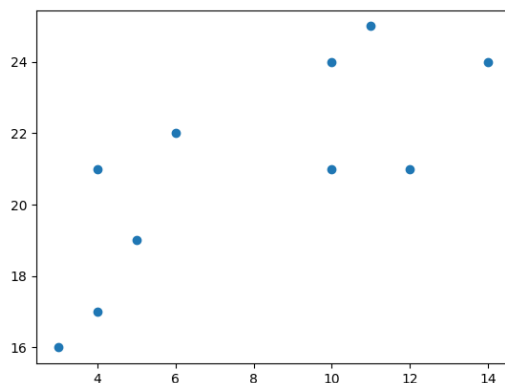
Nasıl çalışır?

Aşağıdan yukarıya bir yaklaşımı izleyen bir tür hiyerarşik kümeleme olan Aglomeratif Kümeleme kullanılacaktır. Her veri noktası kendi kümesi olarak ele alınarak başlanır. Ardından, daha büyük kümeler oluşturmak için aralarında en kısa mesafeye sahip kümeler birleştirilir. Bu adım, tüm veri noktalarını içeren büyük bir küme oluşana kadar tekrarlanır. Hiyerarşik kümeleme, hem mesafe hem de bağlantı yöntemine karar verilmesini gerektirir. Öklid mesafesini ve kümeler arasındaki varyansı en aza indirmeye çalışan Ward bağlantı yöntemi kullanılacaktır.

Örnek: Bazı veri noktaları görselleştirilerek başlanır.

Pyhon kod örneği:

```
import numpy as np
import matplotlib.pyplot as plt
x = [4, 5, 10, 4, 3, 11, 14, 6, 10, 12]
y = [21, 19, 24, 17, 16, 25, 24, 22, 21, 21]
plt.scatter(x, y)
plt.show()
```



Şimdi koğuş bağlantısı (ward linkage) öklid mesafesi kullanılarak hesaplanır ve bu bir dendrogram kullanılarak görselleştirilir.

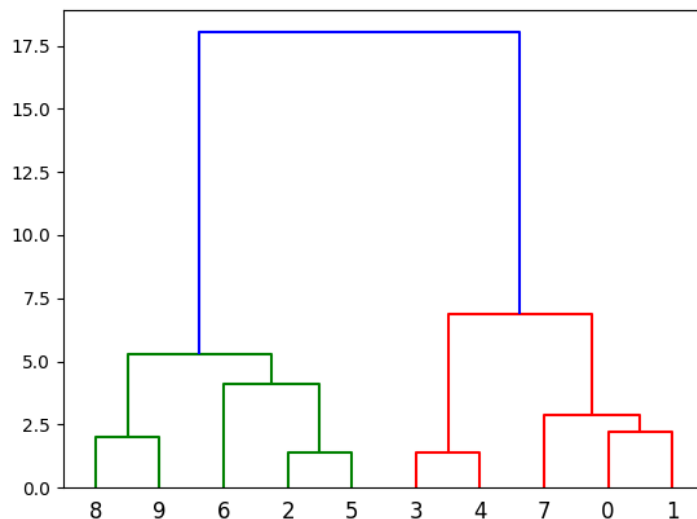
Python kod örneği:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram, linkage
x = [4, 5, 10, 4, 3, 11, 14, 6, 10, 12]
y = [21, 19, 24, 17, 16, 25, 24, 22, 21, 21]

data = list(zip(x, y))

linkage_data = linkage(data, method='ward', metric='euclidean')
dendrogram(linkage_data)

plt.show()
```



Burada aynı şeyi Python'un scikit-learn kütüphanesi ile yapılabilir. Ardından, 2 boyutlu bir alan üzerinde görselleştirilir.

Example

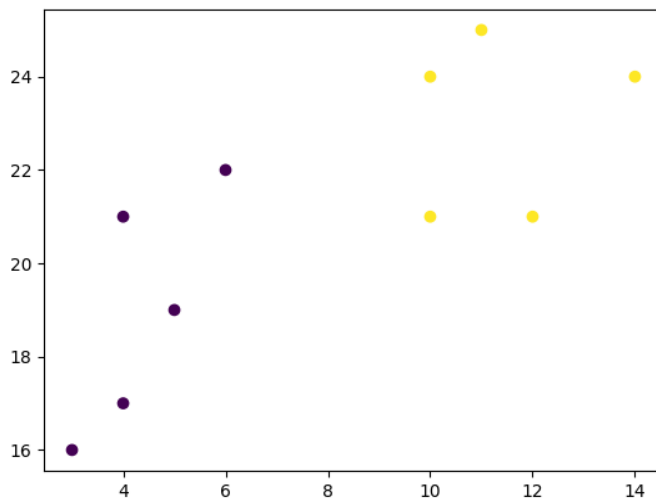
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import AgglomerativeClustering

x = [4, 5, 10, 4, 3, 11, 14, 6, 10, 12]
y = [21, 19, 24, 17, 16, 25, 24, 22, 21, 21]
```

```
data = list(zip(x, y))
```

```
hierarchical_cluster = AgglomerativeClustering(n_clusters=2,  
affinity='euclidean', linkage='ward')  
labels = hierarchical_cluster.fit_predict(data)
```

```
plt.scatter(x, y, c=labels)  
plt.show()
```



Örneğin açıklanması:

Import the modules you need.

```
import numpy as np
```

```
import matplotlib.pyplot as plt
```

```
from scipy.cluster.hierarchy import dendrogram, linkage
```

```
from sklearn.cluster import AgglomerativeClustering
```

NumPy, Python'da diziler ve matrislerle çalışmak için bir kütüphanedir.

scikit-learn, makine öğrenimi için popüler bir kütüphanedir.

Bir veri kümesinde iki değişkene benzeyen diziler oluşturulur. Burada sadece iki değişkenimiz olmasına rağmen, bu yöntemin herhangi bir sayıda değişkenle çalışacağı unutulmamalıdır:

```
x = [4, 5, 10, 4, 3, 11, 14, 6, 10, 12]
```

```
y = [21, 19, 24, 17, 16, 25, 24, 22, 21, 21]
```

Turn the data into a set of points:

```
data = list(zip(x, y))
```

```
print(data)
```

Sonuç:

```
[(4, 21), (5, 19), (10, 24), (4, 17), (3, 16), (11, 25), (14, 24), (6, 22), (10, 21), (12, 21)]
```

Tüm farklı noktalar arasındaki bağlantı hesaplanır. Burada basit bir öklid uzaklık ölçüsü ve kümeler arasındaki varyansı en aza indirmeye çalışan Ward'ın bağlantısı kullanılır.

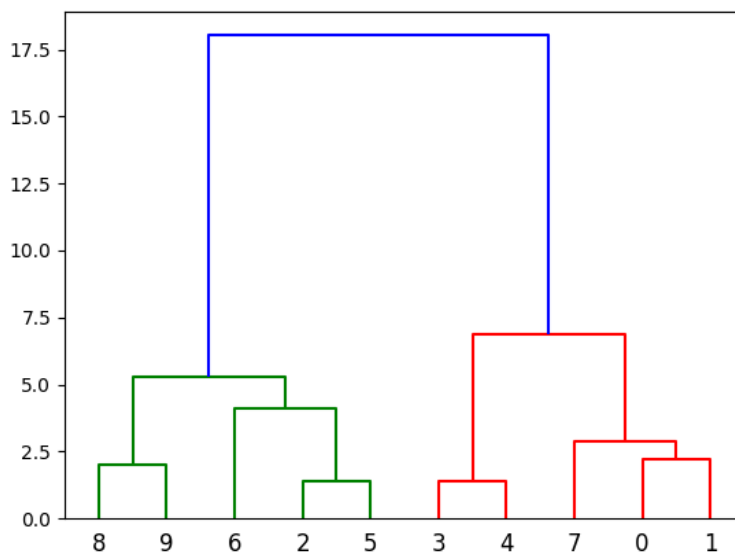
```
linkage_data = linkage(data, method='ward', metric='euclidean')
```

Son olarak, sonuçlar bir dendrogramda çizilir. Bu çizim bize alttan (bireysel noktalar) yukarıya (tüm veri noktalarından oluşan tek bir küme) kümelerin hiyerarşisini gösterecektir. `plt.show()` sadece ham bağlantı verisi yerine dendrogramı görselleştirmemizi sağlar.

```
dendrogram(linkage_data)
```

```
plt.show()
```

Result:



scikit-learn kitaplığı, hiyerarşik kümelemenin farklı bir şekilde kullanılmasına izin verir. İlk olarak, aynı öklid mesafesi ve Ward bağlantısı kullanılarak `AgglomerativeClustering` sınıfını 2 kümeyle başlatılır.

```
hierarchical_cluster = AgglomerativeClustering(n_clusters=2, affinity='euclidean', linkage='ward')
```

`.fit_predict` yöntemi, seçilen küme sayımızda tanımlanan parametreleri kullanarak kümeleri hesaplamak için verilerimiz üzerinde çağrılabilir.

scikit-learn kitaplığı, hiyerarşik kümelemeyi farklı bir şekilde kullanmamıza izin verir. İlk olarak, aynı öklid mesafesi ve Ward bağlantısı kullanılarak `AgglomerativeClustering` sınıfı 2 kümeyle başlatılır.

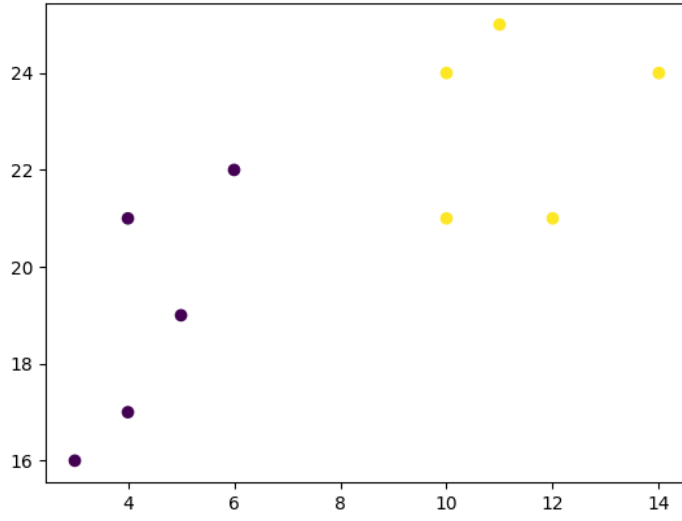
```
labels = hierarchical_cluster.fit_predict(data) print(labels)
```

Sonuç: [0 0 1 0 0 1 1 0 1 1]

Son olarak, hiyerarşik kümeleme yöntemiyle her bir dizine atanan etiketler kullanılarak aynı veriler çizilir ve noktalar renklendirilirse, her noktanın atandığı küme görülebilir:

```
plt.scatter(x, y, c=labels)
```

```
plt.show()
```



2.1.2. K-ortalama

K-Means Algoritması:

K-means algoritmasında kullanılan örneklem, k adet kümeye bölünür. Algoritmanın özü birbirlerine benzerlik gösteren verilerin aynı küme içerisine alınmasına dayanır. Algoritmadaki benzerlik terimi, veriler arasındaki uzaklığa göre belirlenmektedir. Uzaklığın az olması benzerliğin yüksek, çok olması ise düşük olduğu anlamına gelmektedir.

K-means algoritmasının yapısı aşağıdaki gibidir;

- 1) K adet rastgele küme oluştur
- 2) Kare hata oranını hesapla
- 3) Verilerin kümelerin orta noktalarına olan uzaklıklarını bul
- 4) Her veri için en yakın kümeyi, o verinin kümesi olarak belirle
- 5) Yeni yerleşim düzenine göre hata oranını hesapla
- 6) Eğer önceki hata oranı ile şimdiki hata oranı eşit değilse 2,3,4,5 ve 6. adımları tekrarla
- 7) Eğer önceki hata oranı ile şimdiki hata oranı eşitse kümeleme işlemini sonlandır

Dirsek yöntemi, kümelere nasıl ihtiyaç duyacağımız konusunda kullanışlı olur. Dirsek noktasının belirlenmesi gerekir.

K-means, veri noktalarını kümelemek için denetimsiz bir öğrenme yöntemidir. Algoritma, her kümedeki varyansı en aza indirerek veri noktalarını yinelemeli olarak K kümelerine böler.

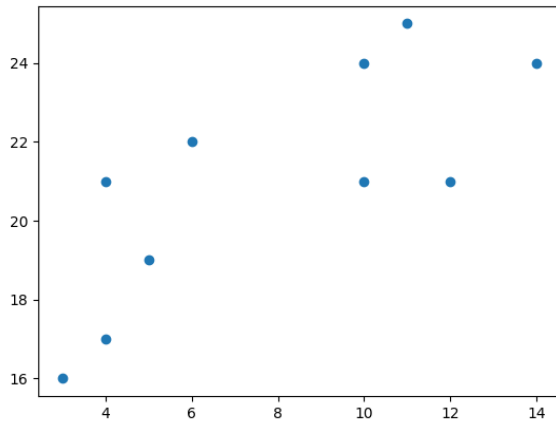
Nasıl çalışır?

İlk olarak, herhangi veri noktası rastgele olarak K kümelerinden birine atanır. Ardından, her kümenin ağırlık merkezini (işlevsel olarak merkez) hesaplarız ve her veri noktasını en yakın ağırlık merkezine sahip kümeye yeniden atarız. Her veri noktası için küme atamaları artık değişmeyene kadar bu işlemi tekrarlıyoruz.

K-ortalama kümeleme, verileri gruplandırmak istediğimiz küme sayısı olan K'yi seçmemizi gerektirir. Dirsek yöntemi, eylemsizliği (mesafeye dayalı bir ölçüm) grafiğini çıkarmamıza ve doğrusal olarak azalmaya başladığı noktayı görselleştirmemize olanak tanır. Bu nokta "elbow" olarak adlandırılır ve verilerimize dayalı olarak K için en iyi değer için iyi bir tahmindir.

Örnek: Bazı veri noktalarını görselleştirerek başlayın.

```
import matplotlib.pyplot as plt
x = [4, 5, 10, 4, 3, 11, 14, 6, 10, 12, 13, 7, 15, 17, 7, 5, 8]
y = [21, 19, 24, 17, 16, 25, 24, 22, 21, 21, 18, 20, 29, 21, 22, 15, 14]
plt.scatter(x, y)
plt.show()
```



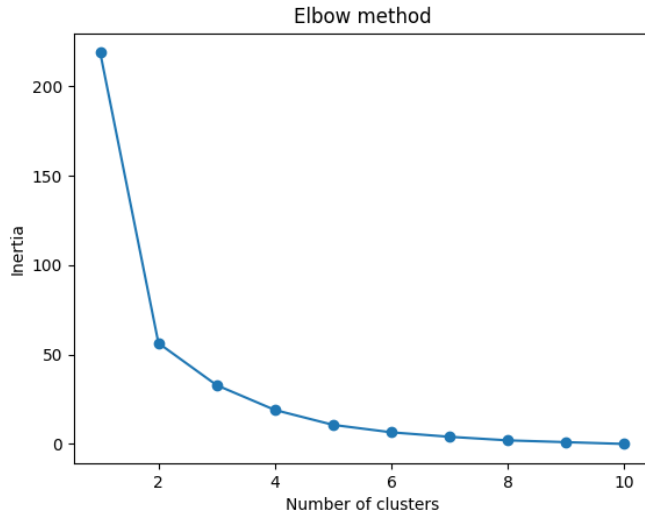
Şimdi farklı K değerleri için ataleti görselleştirmek için dirsek yöntemini (elbow method) kullanıyoruz.

Kod örneği:

```
from sklearn.cluster import KMeans
data = list(zip(x, y))
inertias = []
```

```
for i in range(1,11):
    kmeans = KMeans(n_clusters=i)
    kmeans.fit(data)
    inertias.append(kmeans.inertia_)
```

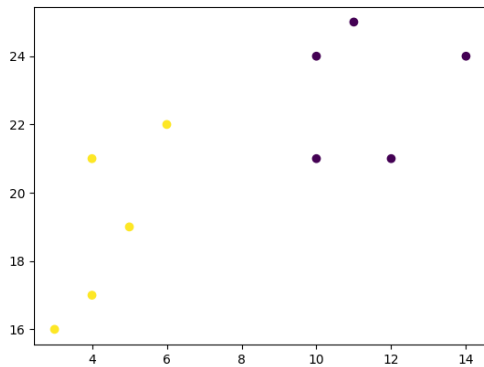
```
plt.plot(range(1,11), inertias, marker='o')
plt.title('Elbow method')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
plt.show()
```



Dirsek yöntemi, 2'nin K için iyi bir değer olduğunu gösterir, bu nedenle sonucu yeniden eğitir ve görselleştiririz.

Kod Örneği:

```
kmeans = KMeans(n_clusters=2)
kmeans.fit(data)
plt.scatter(x, y, c=kmeans.labels_)
plt.show()
```



Örneğin açıklanması:

İhtiyaç duyulan modüller içe aktarılır.

```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
```

from sklearn.cluster import Kmeans, K-Ortalama algoritmasına ait yazılımı kütüphaneden transfer eder.

scikit-learn, makine öğrenimi için popüler bir kütüphanedir. Bir veri kümesinde iki değişkene benzeyen diziler oluşturulur. Burada yalnızca iki değişken kullanırken, bu yöntemin herhangi bir sayıda değişkenle çalışacağını unutmayın:

```
x = [4, 5, 10, 4, 3, 11, 14, 6, 10, 12]
```

```
y = [21, 19, 24, 17, 16, 25, 24, 22, 21, 21]
```

Veriler bir dizi noktaya dönüştürülür:

```
data = list(zip(x, y))
```

```
print(data)
```

Sonuç:

```
[(4, 21), (5, 19), (10, 24), (4, 17), (3, 16), (11, 25), (14, 24), (6, 22), (10, 21), (12, 21)]
```

K için en iyi değeri bulmak için, bir dizi olası değer için verilerimizde K-araçlarını çalıştırmamız gerekir. Yalnızca 10 veri noktamız var, dolayısıyla maksimum küme sayısı 10'dur. Böylece, (1,11) aralığındaki her K değeri için bir K-ortalama modeli eğitiriz ve bu küme sayısındaki inertia'yı çizeriz:

```
inertias = []
```

```
for i in range(1,11):
```

```
    kmeans = KMeans(n_clusters=i)
```

```
    kmeans.fit(data)
```

```
    inertias.append(kmeans.inertia_)
```

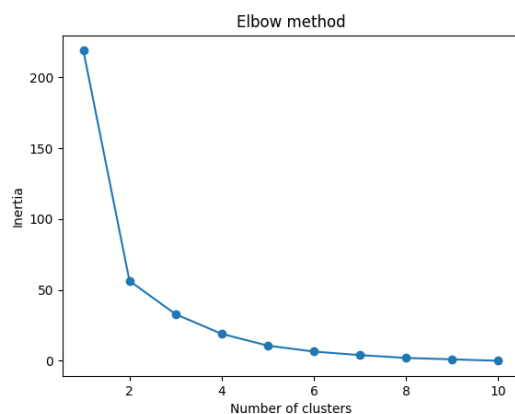
```
plt.plot(range(1,11), inertias, marker='o')
```

```
plt.title('Elbow method')
```

```
plt.xlabel('Number of clusters')
```

```
plt.ylabel('Inertia')
```

```
plt.show()
```



Yukarıdaki grafikte (interia'nın daha lineer hale geldiği) "dirsek" in K=2'de olduğunu görebiliriz. Daha sonra K-araç algoritmamızı bir kez daha uydurabilir ve verilere atanan farklı kümeleri çizebiliriz:

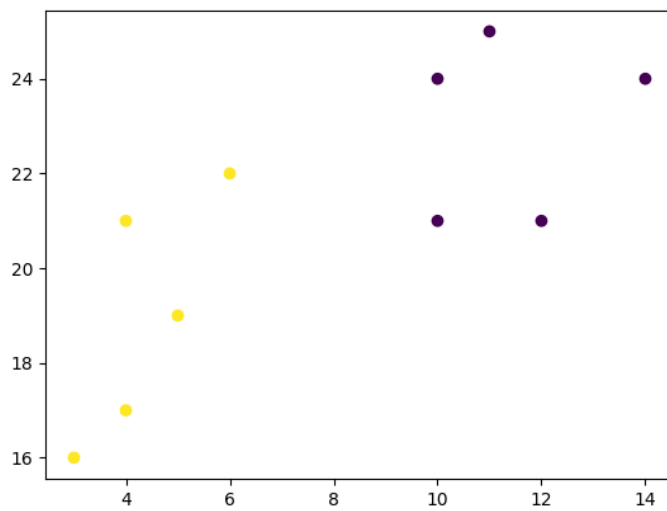
```
kmeans = KMeans(n_clusters=2)
```

```
kmeans.fit(data)
```

```
plt.scatter(x, y, c=kmeans.labels_)
```

```
plt.show()
```

Result:



2.1.3. Temel Bileşen Analizi (Principal Component Analysis - PCA)

PCA (Principal Component Analysis) Temel Bileşenler Analizi

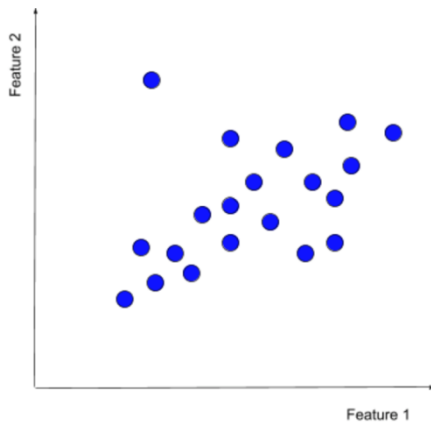
Türkçesi “Temel Bileşenler Analizi” olan PCA tanıma, sınıflandırma, görüntü sıkıştırma alanlarında kullanılan yararlı bir istatistiksel tekniktir. Temel amacı yüksek boyutlu verilerde en yüksek varyans ile veri setini tutmak ancak bunu yaparken boyut indirgemeyi sağlamak olan bir tekniktir. Fazla boyutlu verilerdeki genel özellikleri bularak boyut sayısının azaltılmasını, verinin sıkıştırılmasını sağlar. **Boyut azalmasıyla bazı özelliklerin kaybedileceği kesindir; fakat amaçlanan, bu kaybolan özelliklerin veri yığını hakkında çok az bilgi içeriyor olmasıdır.** Bu yöntem, yüksek korelasyonlu değişkenleri bir araya getirerek, verilerdeki en çok varyasyonu oluşturan “temel bileşenler” olarak adlandırılan daha az sayıda yapay değişken kümesi oluşturulur.

PCA verideki gerekli bilgileri ortaya çıkarmada oldukça etkili bir yöntemdir. **PCA’ın arkasında yatan temel mantık çok boyutlu bir veriyi, verideki temel özellikleri yakalayarak daha az sayıda değişkenle göstermektir.**

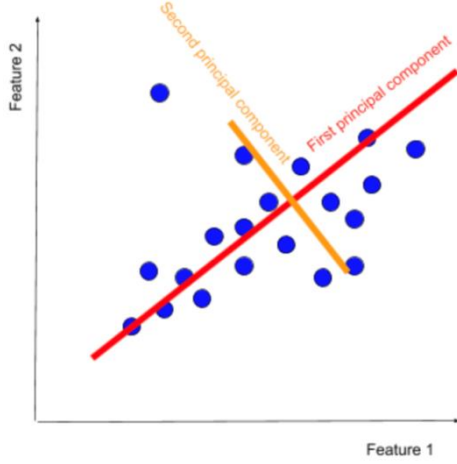
PCA'yı daha iyi anlayabilmek için 2 boyutlu bir veri setimiz olduğunu hayal edin. Her boyut bir özellik sütunu olarak temsil edilebilir:

Feature 1	Feature 2
4	2
6	3
13	6
...	...

Aynı veri kümesini bir dağılım grafiği olarak temsil edebiliriz:



PCA'nın temel amacı, veri noktalarını bir dizi temel bileşenle tanımlayabilen bu tür temel bileşenleri bulmaktır.



PCA'da Temel bileşenler vektörlerdir, ancak rastgele seçilmezler. İlk temel bileşen, orijinal özelliklerdeki en büyük varyansı (En küçük kareler metodu) açıklayacak şekilde hesaplanır. İkinci bileşen birinci bileşene diktir ve birinci ana bileşenden sonra kalan en büyük varyansı açıklar.

Orijinal veriler, özellik vektörleri olarak temsil edilebilir. PCA, bir adım daha ileri gitmemize ve verileri temel bileşenlerin doğrusal kombinasyonları olarak temsil etmemize olanak tanır. Temel bileşenlerin alınması, verilerin özellik1 x özellik2 ekseninden PCA1 x PCA2 eksenine doğrusal bir şekilde dönüştürülmesine eşdeğerdir.

Bu neden yararlıdır?

Yukarıdaki küçük 2 boyutlu örnekte, formun bir özellik vektörü (özellik1, özellik2), formun bir vektörüne (birinci ana bileşen (PCA1), ikinci asıl bileşen) çok benzer olacağından, PCA kullanarak fazla bir şey elde etmiyoruz. bileşen (PCA2)). Ancak çok büyük veri kümelerinde (boyut sayısının 100 farklı değişkeni geçebildiği), ana bileşenler çok sayıda özelliği yalnızca birkaç temel bileşene indirgeyerek gürültüyü ortadan kaldırır. Temel bileşenler, verilerin düşük boyutlu uzaya dikey projeksiyonlarıdır.

PCA ne için kullanılır?

Algoritma kendi başına kullanılabilir veya başka bir makine öğrenme algoritmasından önce kullanılan bir veri temizleme veya veri ön işleme tekniği olarak hizmet edebilir.

Kendi başına PCA, çeşitli kullanım durumlarında kullanılır:

1. Çok boyutlu verileri görselleştirir. Veri görselleştirmeleri, çok boyutlu verileri 2 veya 3 boyutlu grafikler olarak iletmek için harika bir araçtır.
2. Bilgileri sıkıştırır. Temel Bileşen Analizi, verileri daha verimli bir şekilde depolamak ve iletmek için bilgileri sıkıştırmak için kullanılır. Örneğin çok fazla kalite kaybetmeden görüntüleri sıkıştırmak için veya sinyal işlemede kullanılabilir. Teknik, örüntü tanıma

(özellikle yüz tanıma), görüntü tanıma ve daha fazlasında çok çeşitli sıkıştırma problemlerinde başarıyla uygulanmıştır.

3. **Karmaşık iş kararlarını basitleştirir.** PCA, geleneksel olarak karmaşık iş kararlarını basitleştirmek için kullanılmıştır. Örneğin, tüccarlar portföyleri yönetmek için 300'den fazla finansal araç kullanır. Algoritma, faiz oranı türev portföylerinin risk yönetiminde başarılı olduğunu kanıtladı ve finansal araçların sayısını 300'den fazla olanlardan 3-4 ana bileşene indirdi.

4. **Kıvrımlı bilimsel süreçleri netleştirir.** Algoritma, sinirsel toplulukların aksiyon potansiyellerini tetikleme olasılığını artıran dolambaçlı ve çok yönlü faktörlerin anlaşılmasında kapsamlı bir şekilde uygulanmıştır.

Ön işlemenin bir parçası olarak PCA kullanıldığında, algoritma şunlara uygulanır:

1. Eğitim veri kümesindeki boyutların sayısını azaltır.
2. Verilerin gürültüsünü giderir. PCA, en büyük varyansı açıklayan bileşenleri bularak hesaplandığından, verilerdeki sinyali yakalar ve gürültüyü atlar.

PCA'yı hesaplamanın birden fazla yolu vardır:

1. Kovaryans matrisinin öz bileşimi
2. Veri matrisinin tekil değer ayrıştırması
3. Güç yinlemeli hesaplama yoluyla özdeğer yaklaşımı
4. Doğrusal olmayan yinlemeli kısmi en küçük kareler (NIPALS) hesaplaması
5. ... ve daha fazlası.

PCA'nın daha derin bir takdirini kazanmak için ilk yöntem - kovaryans matrisinin öz bileşimine - daha yakından bakalım. PCA'nın hesaplanmasında birkaç adım vardır:

1. Özellik standardizasyonu. Her özelliği ortalama 0 ve varyansı 1 olacak şekilde standartlaştırıyoruz. Daha sonra varsayımlarda ve sınırlamalarda açıklayacağımız gibi, farklı büyüklük sıralarında değerlere sahip özellikler PCA'nın en iyi temel bileşenleri hesaplamasını engeller.
2. Kovaryans matrisi hesaplamasını elde edin. Kovaryans matrisi, d'nin "boyut" (veya verilerimiz tablo halindeyse özellik veya sütun) anlamına geldiği $d \times d$ boyutlarında bir kare matristir. Her bir özellik arasındaki ikili özellik korelasyonunu gösterir.
3. Kovaryans matrisinin öz bileşimini hesaplayın. Kovaryans matrisinin özvektörlerini (birim vektörler) ve bunlarla ilişkili özdeğerlerini (özvektörü çarptığımız skalerler) hesaplarız. Lineer cebirinizi tazelemek istiyorsanız, bu öz bileşim bilginizi yenilemek için iyi bir kaynaktır.
4. Özvektörleri en yüksek özdeğerden en düşüğe doğru sıralayın. En yüksek özdeğere sahip özvektör, ilk temel bileşendir. Daha yüksek özdeğerler, açıklanan daha fazla paylaşılan varyansa karşılık gelir.
5. Ana bileşenlerin sayısını seçin. N ana bileşen olmak için en üstteki N özvektörleri (özdeğerlerine göre) seçin. Optimum temel bileşen sayısı hem öznel hem de soruna bağlıdır. Genellikle, temel bileşenlerin kombinasyonu tarafından açıklanan paylaşılan varyansın

kümülatif miktarına bakarız ve paylaşılan varyansı hala önemli ölçüde açıklayan bu sayıda bileşen seçeriz.

Veri bilimcilerinin çoğunun PCA'yı elle hesaplamayacağını, bunun yerine Python'da ScikitLearn ile uygulayacağını veya hesaplamak için R'yi kullanacağını unutmayın. Bu matematiksel temeller, PCA anlayışımızı zenginleştirir, ancak uygulanması için gerekli değildir. PCA'yı anlamak, avantajları ve dezavantajları hakkında daha iyi bir fikre sahip olmamızı sağlar.

PCA'nın avantajları ve dezavantajları nelerdir?

PCA birden fazla fayda sağlar, ancak aynı zamanda bazı eksikliklerden de muzdariptir.

PCA'nın Avantajları:

1. Hesaplaması kolaydır. PCA, bilgisayarlar tarafından hesaplanması kolay olan doğrusal cebire dayanmaktadır.
2. Diğer makine öğrenimi algoritmalarını hızlandırır. Makine öğrenimi algoritmaları, orijinal veri kümesi yerine temel bileşenler üzerinde eğitildiğinde daha hızlı birleşir.
3. Yüksek boyutlu veri sorunlarına karşı koyar. Yüksek boyutlu veriler, regresyona dayalı algoritmaların kolayca fazla uyum sağlamasına neden olur. Eğitim veri kümesinin boyutlarını azaltmak için önceden PCA'yı kullanarak, tahmine dayalı algoritmaların gereğinden fazla takılmasını önleriz.

PCA'nın Dezavantajları:

1. Temel bileşenlerin düşük yorumlanabilirliği. Temel bileşenler, orijinal verilerdeki özelliklerin doğrusal kombinasyonlarıdır, ancak yorumlanması o kadar kolay değildir. Örneğin, temel bileşenleri hesapladıktan sonra veri kümesindeki en önemli özelliklerin hangileri olduğunu söylemek zordur.
2. Bilgi kaybı ve boyutsallık azaltma arasındaki değiş tokuş. Boyut azaltma yararlı olsa da, bir bedeli vardır. Bilgi kaybı, PCA'nın gerekli bir parçasıdır. Boyut azaltma ve bilgi kaybı arasındaki dengeyi dengelemek, ne yazık ki PCA kullanırken yapmamız gereken gerekli bir uzlaşmadır.

4. PCA'nın varsayımları ve sınırlamaları nelerdir?

PCA, Pearson korelasyonundaki bir dizi işlemle ilgilidir, bu nedenle benzer varsayımları ve sınırlamaları devralır:

- 1) PCA, özellikler arasında bir korelasyon olduğunu varsayar. Özellikler (veya tablo verilerindeki boyutlar veya sütunlar) ilişkili değilse, PCA temel bileşenleri belirleyemez.
- 2) PCA, özelliklerin ölçeğine duyarlıdır. İki özelliğimiz olduğunu hayal edin - biri 0 ile 1000 arasında değerler alırken, diğeri 0 ile 1 arasında değerler alır. PCA, verilerdeki gerçek maksimum varyanstan bağımsız olarak birinci ilke bileşeni olan ilk özelliğe aşırı derecede yanlı olacaktır. Bu yüzden önce değerleri standartlaştırmak çok önemlidir.

- 3) PCA aykırı değerlere karşı dayanıklı değildir. Yukarıdaki noktaya benzer şekilde, algoritma güçlü aykırı değerlere sahip veri kümelerinde yanıltıcı olacaktır. Bu nedenle, PCA gerçekleştirmeden önce aykırı değerlerin kaldırılması önerilir.
- 4) PCA, özellikler arasında doğrusal bir ilişki olduğunu varsayar. Algoritma, doğrusal olmayan ilişkileri yakalamak için pek uygun değildir. Bu nedenle, doğrusal olmayan özelliklerin veya özellikler arasındaki ilişkilerin, günlük dönüşümleri gibi standart yöntemler kullanılarak doğrusal hale getirilmesi önerilir.
- 5) Teknik uygulamalar genellikle hiçbir eksik değer olmadığını varsayar. İstatistiksel yazılım araçlarını kullanarak PCA'yı hesaplarken, genellikle özellik kümesinin eksik değerleri (boş satırlar) olmadığını varsayarlar. Eksik değerlere sahip satırları ve/veya sütunları kaldırdığınızdan veya eksik değerleri yakın bir tahminle (örneğin, sütunun ortalaması) eklediğinizden emin olun.

PCA, verilerdeki makul olduğu kadar çok bilgiyi işleyen bir projeksiyon yapmaya çalışır. Bu bir boyut küçültme tekniğidir. En yüksek varyansa sahip yönleri bulur ve boyutları küçültmek için verileri bunlarla birlikte yansıtır.

Calculation steps of PCA:

Let there is an $N \times 1$ vector with values $x_1, x_2, \dots, x_m$.

- Calculate the sample mean:

$$\bar{x} = 1/M \sum_{i=1}^M x_i$$

- Subtract sample mean with vector value:

$$\Phi = x_i - \bar{x}$$

- Calculate the sample covariance matrix:

$$\Sigma_x = 1/M \sum_{i=1}^M \Phi_i \Phi_i^T = 1/M A A^T$$

$$\text{Where, } A = [\Phi_1 \ \Phi_2 \ \dots \ \Phi_n]$$

$n \times m$ matrix

- Calculate the eigenvalues and eigenvectors of the covariance matrix

$$\sum_x u_i = \lambda_i u_i$$

Assumptions,

$\lambda_1 > \lambda_2 > \dots > \lambda_n$ and $u_1, u_2, \dots, u_n$ are the corresponding eigenvectors

- **Dimensionality reduction:** approximate x using only the first k eigenvectors ($k < N$).

Python Implementation of the Principal Component Analysis (PCA)

The main goal of Python's implementation of the PCA:

- Implement the covariance matrix.
- Derive eigenvalues and eigenvectors.
- Understand the concept of dimensionality reduction from the PCA.

Loading the Iris data

```
import numpy as np
import pylab as pl
import pandas as pd
from sklearn import datasets
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler

load_iris = datasets.load_iris()
iris_df = pd.DataFrame(load_iris.data,
    columns=[load_iris.feature_names])
iris_df.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

Standardization

It is always good to standardize the data to keep all features of the data in the same scale.

```
standardized_x = StandardScaler().fit_transform(load_iris.data)
standardized_x[:2]
```

```
array([[ -0.90068117,  1.01900435, -1.34022653, -1.3154443 ],
       [-1.14301691, -0.13197948, -1.34022653, -1.3154443 ]])
```

Compute Covariance Matrix

```
covariance_matrix_x = np.cov(standardized_x.T)
covariance_matrix_x
```

```
array([[ 1.00671141, -0.11835884,  0.87760447,  0.82343066],
       [-0.11835884,  1.00671141, -0.43131554, -0.36858315],
       [ 0.87760447, -0.43131554,  1.00671141,  0.96932762],
       [ 0.82343066, -0.36858315,  0.96932762,  1.00671141]])
```

Compute Eigenvalues and Eigenvectors from Covariance Matrix

```
eigenvalues, eigenvectors = np.linalg.eig(covariance_matrix_x)eigenvalues
```

```
array([2.93808505, 0.9201649 , 0.14774182, 0.02085386])
```

eigenvectors

```
array([[ 0.52106591, -0.37741762, -0.71956635,  0.26128628],
       [-0.26934744, -0.92329566,  0.24438178, -0.12350962],
       [ 0.5804131 , -0.02449161,  0.14212637, -0.80144925],
       [ 0.56485654, -0.06694199,  0.63427274,  0.52359713]])
```

Check variance in Eigenvalues

```
total_of_eigenvalues = sum(eigenvalues)variance = [(i / total_of_eigenvalues)*100 for i in
sorted(eigenvalues, reverse=True)]variance
```

```
[72.96244541329989, 22.850761786701753, 3.668921889282865, 0.5178709107154905]
```

The values shown in figure indicate the variance giving the analysis below:

- 1st Component = 72.96%
- 2nd Component = 22.85%
- 3rd Component = 3.5%
- 4th Component = 0.5%

So, the **third** and **fourth** Components have very low variance respectively. These can be dropped. Because these components can't add any value.

Taking 1st and 2nd Components only and Reshaping

```
eigenpairs = [(np.abs(eigenvalues[i]), eigenvectors[:,i]) for i in range(len(eigenvalues))]
```

```
Sorting from Higher values to lower valueeigenpairs.sort(key=lambda x: x[0],
```

```
reverse=True)eigenpairs
```

```
[(2.938085050199995,
  array([ 0.52106591, -0.26934744,  0.5804131 ,  0.56485654])),
 (0.9201649041624864,
  array([-0.37741762, -0.92329566, -0.02449161, -0.06694199])),
 (0.1477418210449475,
  array([-0.71956635,  0.24438178,  0.14212637,  0.63427274])),
 (0.020853862176462696,
  array([ 0.26128628, -0.12350962, -0.80144925,  0.52359713]))]
```

Perform Matrix Weighing of Eigenpairs

```
matrix_weighing =  
np.hstack((eigenpairs[0][1].reshape(4,1),eigenpairs[1][1].reshape(4,1)))matrix_weighing  
array([[ 0.52106591, -0.37741762],  
       [-0.26934744, -0.92329566],  
       [ 0.5804131 , -0.02449161],  
       [ 0.56485654, -0.06694199]])
```

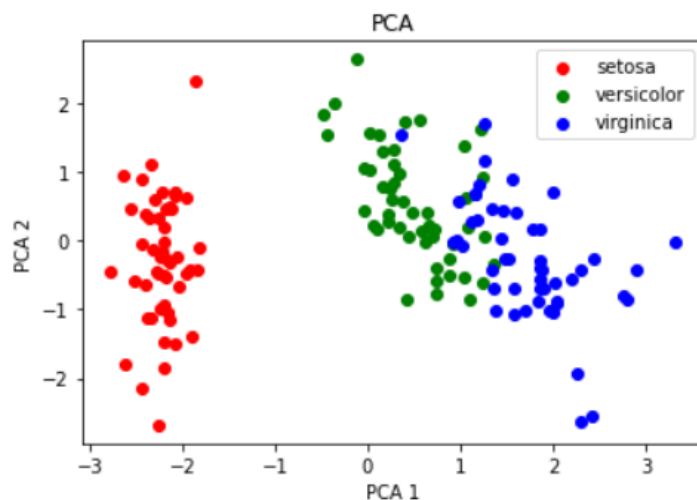
Multiply the standardized matrix with matrix weighing:

```
Y = standardized_x.dot(matrix_weighing)Y
```

```
array([[ -2.26470281, -0.4800266 ],  
       [ -2.08096115,  0.67413356],  
       [ -2.36422905,  0.34190802],  
       [ -2.29938422,  0.59739451],  
       [ -2.38984217, -0.64683538],  
       [ -2.07563095, -1.48917752],  
       [ -2.44402884, -0.0476442 ],  
       [ -2.23284716, -0.22314807],  
       [ -2.33464048,  1.11532768],  
       [ -2.18432817,  0.46901356],  
       [ -2.1663101 , -1.04369065],  
       [ -2.32613087, -0.13307834],  
       [ -2.2184509 ,  0.72867617],  
       [ -2.6331007 ,  0.96150673],
```

Plotting

```
plt.figure()target_names = load_iris.target_names  
y = load_iris.targetfor c, i, target_name in zip("rgb", [0, 1, 2], target_names):  
    plt.scatter(Y[y==i,0], Y[y==i,1], c=c, label=target_name)plt.xlabel('PCA 1')  
plt.ylabel('PCA 2')  
plt.legend()  
plt.title('PCA')  
plt.show()
```



Matrix Decomposition or Matrix Factorization

Matrix Decomposition or factorization is also an important part of linear algebra used in machine learning. Basically, it is a factorization of the matrix into a product of matrices. There are several techniques of matrix decomposition like LU decomposition, Singular value decomposition (SVD), etc.

Singular Value Decomposition (SVD)

It is a technique for the reduction of dimension. As per singular value decomposition matrix: Let M is a rectangular matrix and can be broken down into three products of matrix — (1) orthogonal matrix (U), (2) diagonal matrix (S), and (3) transpose of the orthogonal matrix (V).

$$M = U S V^T$$

The diagram illustrates the Singular Value Decomposition (SVD) equation $M = U S V^T$. Below the equation, three labels are connected to the matrices by blue lines: 'Orthogonal' is connected to U , 'Diagonal' is connected to S , and 'Orthogonal' is connected to V^T .

Conclusion

Machine learning and deep learning has been build upon the concept of mathematics. A vast area of mathematics is used to build algorithms and also for the computation of data. Linear algebra is the study of vectors [\[10\]](#) and several rules to manipulate vectors. It is a key infrastructure, and it covers many areas of machine learning like linear regression, one-hot encoding in the categorical variable, PCA (Principle component analysis) for the dimensionality reduction, matrix factorization for recommender systems.

Deep learning is completely based on linear algebra and calculus. It is also used in several optimization techniques like gradient descent, stochastic gradient descent, and others. Matrices are an essential part of linear algebra that we use to compactly represent systems of linear equations, linear mapping, and others. Also, vectors are unique objects that can be added together and multiplied by scalars that produce another object of similar kinds. Any suggestions or feedback is crucial to continue to improve. Please let us know in the comments if you have any.

Veri setimiz üzerinde PCA yapabilmek için sklearn kütüphanesinden PCA metotunu çağırmanız gereklidir. Öncelikle iris veri setimizi sklearn'den indirelim. iris veri setimizde 3 farklı sınıfa ait 4 öznitelik bulunmaktadır.

```
In [1]: ## import load_iris dataset
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import seaborn as sns
```

```
In [2]: iris = load_iris()
data = iris.data
feature_names = iris.feature_names
y = iris.target
```

```
In [3]: y
```

[illegible]

```
In [4]: df = pd.DataFrame(data, columns = feature_names)
df["sinif"] = y
x = data
df.head()
```

```
Out[4]:
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

Veri seti üzerinde yapmış olduğumuz PCA dönüşümü bize 2 bileşen seçmenin, verilerin toplam varyansının yaklaşık % 97.7'sini koruyabileceğimizi söylemektedir. Tüm bileşenleri kullanmak istemediğimiz sadece **ana bileşenleri** kullanmak istediğimiz için bu varyans oranı bizim için yeterlidir.

```
In [5]: ### PCA
from sklearn.decomposition import PCA
pca = PCA(n_components = 2, whiten= True ) # whiten = normalize
pca.fit(x)

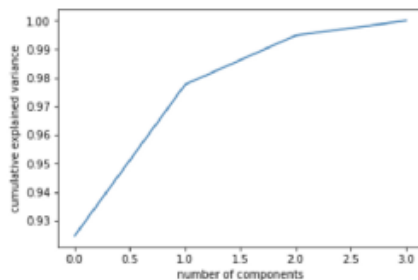
x_pca = pca.transform(x)

print("variance ratio: ", pca.explained_variance_ratio_)

print("sum: ",sum(pca.explained_variance_ratio_))

variance ratio: [ 0.92461621 0.05301557]
sum: 0.977631775025
```

```
In [6]: pca = PCA(whiten=True).fit(x)
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance');
plt.show()
```

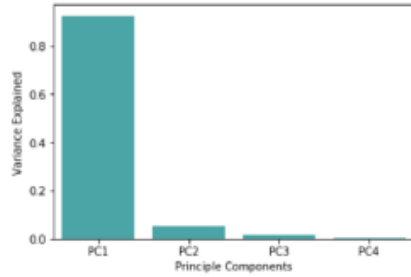


```
In [7]: df_sns = pd.DataFrame({'var':pca.explained_variance_ratio_,
                              'PC':['PC1','PC2','PC3','PC4']})
df_sns
```

```
Out[7]:
```

	PC	var
0	PC1	0.924616
1	PC2	0.053016
2	PC3	0.017185
3	PC4	0.005183

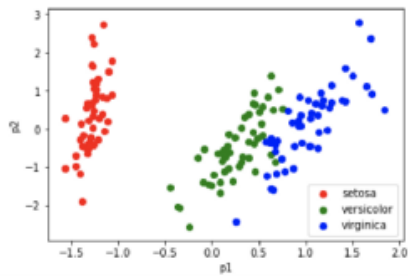
```
In [8]: sns.barplot(x='PC',y='var',data=df_sns, color="c")
plt.ylabel("Variance Explained")
plt.xlabel("Principle Components")
plt.show()
```



PCA dönüşümü işlemini tamamladıktan sonra sınıfların iki boyutlu vektörlerini aşağıdaki grafikte görebiliriz.

```
In [9]: df["p1"] = x_pca[:,0]
df["p2"] = x_pca[:,1]
color = ["red","green","blue"]
```

```
In [10]: for each in range(3):
plt.scatter(df.p1[df.sinif == each],df.p2[df.sinif == each],color = color[each],label = iris.target_names[each])
plt.legend()
plt.xlabel("p1")
plt.ylabel("p2")
plt.show()
```



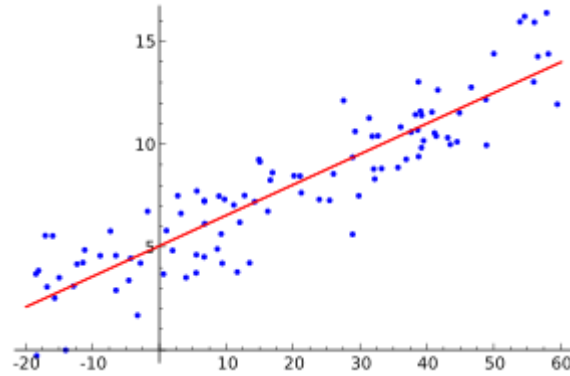
3. Regresyon Algoritmaları

Değişkenler arasındaki ilişkiyi bulmaya çalıştığınızda regresyon yöntemi kullanılır. Makine Öğreniminde ve istatistiksel modellemede bu ilişki gelecekteki olayların sonucunu tahmin etmek için kullanılır. Sürekli değerleri tahmin etmek için regresyon algoritmaları kullanılır.

Regresyon algoritmaları:

- Linear Regression
- Polynomial Regression
- Exponential Regression
- Logistic Regression
- Logarithmic Regression

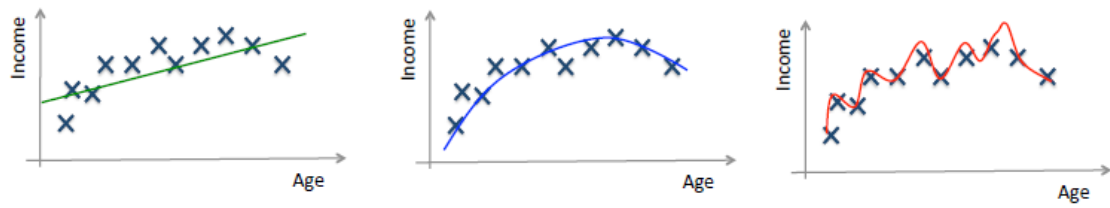
Regresyon analizi, girdi değişkenleri ile çıktı arasındaki ilişkiyi tahmin etmek için çok çeşitli istatistiksel yöntemleri kapsar. En yaygın şekli, sıradan en küçük kareler gibi bir matematiksel kritere göre verilen verilere en iyi uyacak şekilde tek bir çizginin çizildiği doğrusal regresyondur.



Makine öğrenmesinde bir modeli eğitmenin amacı, tüm örneklerde ortalama olarak düşük kayıplı bir eşik değer bulmaktır. Bu ortalama kareler hatası ile bulunur; farkların kareleri toplamının örnek sayısına bölümüdür. Regresyon algoritmaları, girdi özellikleri ile çıktı arasındaki ilişkiyi modelleyen istatistiksel bir yaklaşımdır. Regresyon algoritmasında veri kümesine ait kabul edilebilir birden fazla sayıda fonksiyon elde edilebilir. Bu durumda doğru fonksiyonu seçimde ortalama karelerin hatası göz önüne alınır.

Regresyon katsayıları bulunurken optimal kapsayıcı bir eğri fonksiyonu elde edilmelidir.

İpucu: Sınıflandırma yapılırken yüksek dereceli polinomlardan kaçınılmalıdır.



$$f(x) = \lambda_0 + \lambda_1 x \dots (1)$$

$$f(x) = \lambda_0 + \lambda_1 x + \lambda_2 x^2 \dots (2)$$

$$f(x) = \lambda_0 + \lambda_1 x + \lambda_2 x^2 + \lambda_3 x^3 + \lambda_4 x^4 \dots (3)$$

Yeterli veri olduğunda, aşırı uyum sorununu önlemek için "İzotonik Regresyon" kullanılır.

3.1.1. Doğrusal Regrasyon

Doğrusal Regresyon Nedir? Makine Öğreniminde Nasıl Kullanılır?

Çoğu makine öğrenimi (ML) algoritması, başta istatistik olmak üzere çeşitli alanlardan ödünç alınmıştır. Modellerin daha iyi tahmin etmesine yardımcı olabilen her şey, sonunda makine öğreniminin bir parçası haline gelmektedir. Bu nedenle, doğrusal regresyonun hem istatistiksel hem de makine öğrenimi algoritması olduğunu söylemek güvenlidir.

Doğrusal regresyon, veri bilimi ve makine öğreniminde kullanılan popüler ve karmaşık olmayan bir algoritmadır. Denetimli bir öğrenme algoritmasıdır ve değişkenler arasındaki matematiksel ilişkiyi incelemek için kullanılan en basit regresyon biçimidir.

Doğrusal regresyon nedir?

Doğrusal regresyon, değişkenler arasındaki ilişkiyi göstermeye çalışan istatistiksel bir yöntemdir. Farklı veri noktalarına bakar ve bir trend çizgisi çizer. Doğrusal regresyonun basit bir örneği, bir makine parçasını tamir etme maliyetinin zamanla arttığını bulmaktır.

Daha kesin olarak, bir bağımlı değişken ile bir dizi diğer bağımsız değişken arasındaki ilişkinin karakterini ve gücünü belirlemek için doğrusal regresyon kullanılır. Bir şirketin hisse senedi fiyatını tahmin etmek gibi tahminler yapmak için modeller oluşturmaya yardımcı olur.

Gözlemlenen veri kümesine doğrusal bir model uydurmaya çalışmadan önce, değişkenler arasında bir ilişki olup olmadığı değerlendirilmelidir. Elbette bu, bir değişkenin diğerine neden olduğu anlamına gelmez, ancak aralarında gözle görülür bir korelasyon olmalıdır.

Örneğin, daha yüksek üniversite notları mutlaka daha yüksek bir maaş paketi anlamına gelmez. Ancak iki değişken arasında bir ilişki olabilir.

Doğrusal terimi, bir çizgiye benzeyen veya çizgilerle ilgili anlamına gelir. Bir dağılım grafiği oluşturmak, açıklayıcı (bağımsız) ve bağımlı değişkenler arasındaki ilişkinin gücünü belirlemek için idealdir. Dağılım grafiği herhangi bir artan veya azalan eğilim göstermiyorsa, gözlemlenen değerlere doğrusal bir regresyon modeli uygulamak faydalı olmayabilir.

Korelasyon katsayıları, iki değişken arasındaki ilişkinin ne kadar güçlü olduğunu hesaplamak için kullanılır. Genellikle r ile gösterilir ve -1 ile 1 arasında bir değere sahiptir. Pozitif bir korelasyon katsayısı değeri, değişkenler arasında pozitif bir ilişkiyi gösterir. Aynı şekilde, negatif bir değer, değişkenler arasında negatif bir ilişkiyi gösterir.

İpucu: Regresyon analizini yalnızca korelasyon katsayısı pozitif veya negatif 0,50 veya üzerindeyse gerçekleştirilmelidir.

Çalışma süresi ile notlar arasındaki ilişkiye bakıyor olsaydınız, muhtemelen pozitif bir ilişki görürdünüz. Öte yandan, sosyal medyada geçirilen süre ile notlar arasındaki ilişkiye bakarsanız, büyük olasılıkla negatif bir ilişki görürsünüz.

Burada “notlar” bağımlı değişken, ders çalışmak veya sosyal medyada geçirilen süre ise bağımsız değişkendir. Bunun nedeni, notların çalışmak için ne kadar zaman harcadığınıza bağlı olmasıdır.

Hem dağılım grafiği hem de korelasyon katsayısı yoluyla değişkenler arasında (en azından) orta düzeyde bir korelasyon kurabilirsiniz, söz konusu değişkenler bir tür doğrusal ilişkiye sahiptir.

Kısacası, doğrusal regresyon, gözlemlenen verilere doğrusal bir denklem uygulayarak iki değişken arasındaki ilişkiyi modellemeye çalışır. Doğrusal bir regresyon çizgisi, düz bir çizginin denklemi kullanılarak temsil edilebilir:

$$y = mx + b$$

Bu basit doğrusal regresyon denkleminde:

- y tahmini bağımlı değişkendir (veya çıktıdır)
- m , regresyon katsayısıdır (veya eğimdir)
- x , bağımsız değişkendir (veya girdidir)
- b sabittir (veya y -keseni üzerinde bir nokta)

Değişkenler arasındaki ilişkiyi bulmak, değerleri veya sonuçları tahmin etmeyi mümkün kılar. Başka bir deyişle, doğrusal regresyon, mevcut verilere dayalı olarak yeni değerlerin tahmin edilmesini mümkün kılar.

Bir örnek, alınan yağışa dayalı olarak mahsul verimini tahmin etmek olabilir. Bu durumda, yağış bağımsız değişkendir ve mahsul verimi (öngörülen değerler) bağımlı değişkendir.

Bağımsız değişkenler aynı zamanda yordayıcı değişkenler olarak da adlandırılır. Aynı şekilde, bağımlı değişkenler de yanıt değişkenleri olarak bilinir.

Lineer regresyonda anahtar terminolojiler

Doğrusal regresyon analizini anlamak, bir dizi yeni terime aşina olmak anlamına da gelebilir. İstatistik veya makine öğrenimi dünyasına yeni adım attıysanız, bu terminolojileri adil bir şekilde anlamanız önemli olacaktır.

- **Değişken (Variable):** Sayılabilen veya ölçülebilen herhangi bir sayı, nicelik veya özelliktir. Veri ögesi olarak da adlandırılır. Gelir, yaş, hız ve cinsiyet örnek olarak verilebilir.
- **Katsayı (Coefficient):** Yanındaki değişkenle çarpılan bir sayıdır (genellikle bir tam sayıdır). Örneğin, $7x$ 'te 7 sayısı katsayıdır. $y = a x + b$, a ve b katsayıdır, x: bağımsız değişken, y ise bağımlı değişkendir.
- **Aykırı Değerler (Outliers):** Bunlar, diğerlerinden önemli ölçüde farklı olan veri noktalarıdır.
- **Kovaryans (Covariance):** İki değişken arasındaki doğrusal ilişkinin yönü. Başka bir deyişle, iki değişkenin doğrusal olarak ilişkili olma derecesini hesaplar.
- **Çok değişkenli (Multivariate):** Tek bir sonuçla sonuçlanan iki veya daha fazla bağımsız değişkenlerin dahil edilmesi anlamına gelir.
- **Artıklar (Residuals):** Bağımlı değişkenin gözlemlenen ve tahmin edilen değerleri arasındaki fark.
- **Değişkenlik (Variability):** Tutarlılığın olmaması veya bir dağılımın ne ölçüde sıkıştırıldığı veya esnetildiği.
- **Doğrusallık (Linearity):** Orantılılıkla yakından ilgili olan ve grafiksel olarak düz bir çizgi olarak gösterilebilen bir matematiksel ilişkinin özelliği.
- **Doğrusal fonksiyon (Linear function):** Grafiği düz bir çizgi olan fonksiyondur.
- **Doğrusallık (Collinearity):** Bir regresyon modelinde doğrusal bir ilişki sergileyecek şekilde bağımsız değişkenler arasındaki korelasyon.
- **Standart sapma (Standard deviation - SD):** Bir veri kümesinin ortalamasına göre dağılımının bir ölçüsüdür. Başka bir deyişle, sayıların ne kadar dağıldığının bir ölçüsüdür.
- **Standart hata (Standard error - SE):** İstatistiksel bir örneklem popülasyonunun yaklaşık SD'si. Değişkenliği ölçmek için kullanılır.

Doğrusal regresyon türleri

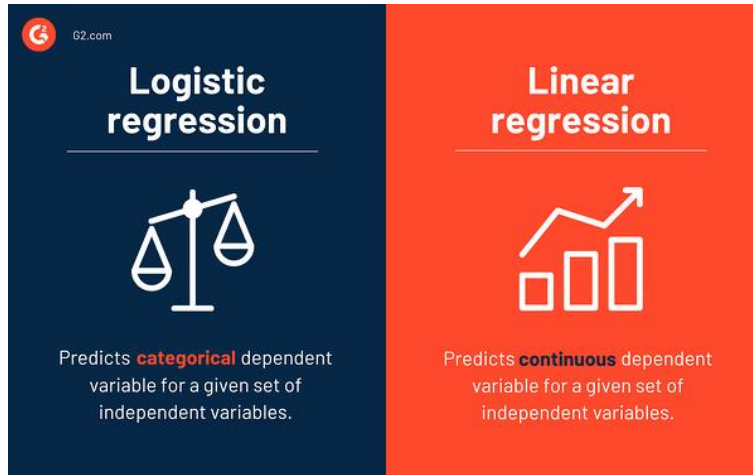
İki tür doğrusal regresyon vardır: basit doğrusal regresyon ve çoklu doğrusal regresyon.

Basit doğrusal regresyon yöntemi, tek bir bağımsız değişken ile buna karşılık gelen bir bağımlı değişken arasındaki ilişkiyi bulmaya çalışır. Bağımsız değişken girdidir ve karşılık gelen bağımlı değişken çıktıdır. İpucu: Lineer regresyonu Python, R, MATLAB ve Excel gibi çeşitli programlama dillerinde ve ortamlarında uygulayabilirsiniz.

Çoklu doğrusal regresyon yöntemi, iki veya daha fazla bağımsız değişken ile bunlara karşılık gelen bağımlı değişken arasındaki ilişkiyi bulmaya çalışır. Polinom regresyon adı verilen özel bir çoklu doğrusal regresyon durumu da vardır.

Basit bir şekilde ifade etmek gerekirse, basit bir doğrusal regresyon modelinde yalnızca tek bir bağımsız değişken bulunurken, çoklu bir doğrusal regresyon modelinde iki veya daha fazla bağımsız değişken olacaktır. Ve evet, oldukça karmaşık veri analizi için kullanılan başka doğrusal olmayan regresyon yöntemleri de var.

Doğrusal regresyon, belirli bir bağımsız değişken seti için sürekli bağımlı değişkeni tahmin ederken, lojistik regresyon kategorik bağımlı değişkeni tahmin eder. Her ikisi de denetimli öğrenme yöntemleridir. Ancak regresyon problemlerini çözmek için lineer regresyon kullanılırken, sınıflandırma problemlerini çözmek için lojistik regresyon kullanılır.



Elbette lojistik regresyon, regresyon problemlerini çözebilir, ancak esas olarak sınıflandırma problemlerinde kullanılır. Çıktısı yalnızca 0 veya 1 olabilir. İki sınıf arasındaki olasılıkları belirlemeniz veya başka bir deyişle bir olayın olasılığını hesaplamanız gereken durumlarda değerlidir. Örneğin, bugün yağmur yağıp yağmayacağını tahmin etmek için lojistik regresyon kullanılabilir.

Doğrusal regresyon varsayımları:

Değişkenler arasındaki ilişkiyi modellemek için doğrusal regresyon kullanırken birkaç varsayımda bulunuyoruz. Varsayımlar, tahminlerde bulunmak için bir model kullanmadan önce karşılanması gereken gerekli koşullardır.

Doğrusal regresyon modelleriyle ilgili genellikle dört varsayım vardır:

- Doğrusal ilişki: Bağımsız değişken x ile bağımlı değişken y arasında doğrusal bir ilişki vardır.
- Bağımsızlık: Artıklar bağımsızdır. Zaman serisi verilerinde ardışık artıklar arasında bir ilişki yoktur.
- Eş varyans: Artıklar tüm seviyelerde eşit varyansa sahiptir.
- Normallik: Artıklar normal dağılır.

Doğrusal regresyon modellerini çözme yöntemleri:

Makine öğreniminde veya istatistik dilinde, doğrusal bir regresyon modeli öğrenmek, mevcut verileri kullanarak katsayıların değerlerini tahmin etmek anlamına gelir. Doğrusal bir regresyon modelini daha verimli hale getirmek için çeşitli yöntemler uygulanabilir.

İpucu: Tekdüze görevleri ortadan kaldırmak ve doğru tahminler yapmak için makine öğrenimi yazılımını kullanılır. Farklılıklarını ve takaslarını anlamak için doğrusal regresyon modellerini çözmek için kullanılan farklı tekniklere bakılmalıdır.

Basit doğrusal regresyon:

Daha önce de belirtildiği gibi, basit doğrusal regresyonda tek bir girdi veya bir bağımsız değişken ve bir bağımlı değişken vardır. Sürekli doğada oldukları göz önüne alındığında, iki değişken arasındaki en iyi ilişkiyi bulmak için kullanılır. Örneğin, tüketilen kalorilere göre kazanılan kilo miktarını tahmin etmek için kullanılabilir.

Sıradan en küçük kareler regresyonu, birden fazla bağımsız değişken veya girdi olduğunda katsayıların değerini tahmin etmenin başka bir yöntemidir. Doğrusal regresyonu çözmek için en yaygın yaklaşımlardan biridir ve normal denklem olarak da bilinir.

Bu prosedür kare artıkların toplamını en aza indirmeye çalışır. Verileri bir matris olarak ele alır ve her katsayı için en uygun değerleri belirlemek için doğrusal cebir işlemlerini kullanır. Tabii ki, bu yöntem ancak tüm verilere erişimimiz varsa uygulanabilir ve ayrıca verileri sıfırlamak için yeterli bellek olmalıdır.

Gradyan iniş, doğrusal regresyon problemlerini çözmek için en kolay ve yaygın olarak kullanılan yöntemlerden biridir. Bir veya daha fazla girdi olduğunda kullanışlıdır ve modelin hatasını yinelemeli olarak en aza indirerek katsayıların değerini optimize etmeyi içerir.

Gradyan iniş, her katsayı için rastgele değerlerle başlar. Her giriş ve çıkış değeri çifti için, kareleri alınmış hataların toplamı hesaplanır. Öğrenme oranı olarak bir ölçek faktörü kullanır ve her katsayı hatayı en aza indirecek yönde güncellenir.

İşlem, daha fazla iyileştirme mümkün olmayana veya minimum kareler toplamına ulaşılan kadar tekrarlanır. Gradyan iniş, belleğe sığmayan çok sayıda satır ve sütun içeren büyük bir veri kümesi olduğunda yardımcı olur.

Düzenleştirme (Regularization), bir modelin hata karelerinin toplamını en aza indirmeye çalışan ve aynı zamanda modelin karmaşıklığını azaltan bir yöntemdir. Sıradan en küçük kareler yöntemini kullanarak kareleri alınmış hataların toplamını azaltır.

Kement regresyonu ve sırt regresyonu (Lasso regression and ridge regression), lineer regresyonda düzenleştirmenin iki ünlü örneğidir. Bu yöntemler, bağımsız değişkenlerde eşdoğrusallık olduğunda değerlidir.

Adam'ın yöntemi (Adam's method):

Uyarlanabilir moment tahmini veya ADAM, derin öğrenmede kullanılan bir optimizasyon algoritmasıdır. Gürültülü veriler üzerinde iyi performans gösteren yinelemeli bir algoritmadır. Uygulaması kolaydır, hesaplama açısından verimlidir ve minimum bellek gereksinimlerine sahiptir.

ADAM, iki gradyan iniş algoritmasını birleştirir - kök ortalama kare yayılma (**root mean square propagation - RMSprop**) ve uyarlanabilir gradyan iniş (adaptive gradient descent). ADAM, gradyanı hesaplamak için tüm veri kümesini kullanmak yerine, stokastik bir yaklaşım yapmak için rasgele seçilmiş alt kümeleri kullanır.

ADAM, çok sayıda parametre veya veri içeren problemler için uygundur. Ayrıca, bu optimizasyon yönteminde, hiperparametreler genellikle minimum ayar gerektirir ve sezgisel yorumlamaya sahiptir.

Tekil değer ayrışımı

Tekil değer ayrışımı veya SVD (Singular value decomposition), doğrusal regresyonda yaygın olarak kullanılan bir boyut indirgeme tekniğidir. Öğrenme algoritması için boyut sayısını azaltan bir ön işleme adıımıdır.

SVD, bir matrisi diğer üç matrisin bir ürünü olarak parçalamayı içerir. Yüksek boyutlu veriler için uygundur ve küçük veri kümeleri için verimli ve karardır. Karardılığı nedeniyle, lineer regresyon için lineer denklemleri çözmek için en çok tercih edilen yaklaşımlardan biridir. Ancak, aykırı değerlere karşı hassastır ve büyük bir veri kümesiyle kararsız hale gelebilir.

Verileri doğrusal regresyon için hazırlama

Çoğu durumda gerçek dünya verileri eksiktir. Diğer herhangi bir makine öğrenimi modelinde olduğu gibi, veri hazırlama ve ön işleme, doğrusal regresyonda çok önemli bir süreçtir. Eksik değerler, hatalar, aykırı değerler, tutarsızlıklar ve öznitelik değerleri eksikliği olacaktır.

Eksik verileri hesaba katmanın ve daha güvenilir bir tahmin modeli oluşturmanın bazı yolları aşağıda verilmiştir.

- Doğrusal regresyon, yordayıcı (Tahmin edici) ve yanıt değişkenlerinin gürültülü olmadığını düşünür. Bu nedenle, çeşitli veri temizleme işlemleriyle gürültünün giderilmesi çok önemlidir. Mümkünse çıktı değişkenindeki aykırı değerler kaldırılmalıdır.
- Girdi ve çıktı değişkenleri Gauss dağılımına sahipse, lineer regresyon daha iyi tahminler yapacaktır.
- Girdi değişkenlerini normalleştirme veya standardizasyon kullanarak yeniden ölçeklendirilirse, doğrusal regresyon genellikle daha iyi tahminler yapar.
- Çok sayıda öznitelik varsa, verileri doğrusal bir ilişkiye sahip olacak şekilde dönüştürülmesi gerekir.
- Girdi değişkenleri yüksek oranda ilişkiliyse, doğrusal regresyon verileri aşacaktır. Bu gibi durumlarda, eşdoğrusallık kaldırılmalıdır.

Doğrusal regresyonun avantajları ve dezavantajları:

Doğrusal regresyon, anlaşılması en basit ve uygulaması en basit algoritmalarından biridir. Değişkenler arasındaki ilişkileri analiz etmek için harika bir araçtır.

Lineer regresyonun bazı dikkate değer avantajları şunlardır:

- Basitliği nedeniyle bir devam et algoritması.
- Fazla uydurmaya yatkın olmasına rağmen, boyut küçültme tekniklerinin yardımıyla önlenebilir.
- İyi yorumlanabilirliğe sahiptir.
- Doğrusal olarak ayrılabilir veri kümelerinde iyi performans gösterir.
- Alan karmaşıklığı düşüktür; bu nedenle, yüksek gecikmeli bir algoritmadır.

Bununla birlikte, pratik uygulamaların çoğu için genellikle doğrusal regresyon önerilmez. Bunun nedeni, değişkenler arasında doğrusal bir ilişki olduğunu varsayarak gerçek dünya sorunlarını aşırı basitleştirmesidir.

Lineer regresyonun bazı dezavantajları şunlardır:

- Aykırı değerlerin regresyon üzerinde olumsuz etkileri olabilir.
- Doğrusal bir modele uyması için değişkenler arasında doğrusal bir ilişki olması gerektiğinden, değişkenler arasında doğrusal bir ilişki olduğu varsayılır.
- Verilerin normal dağıldığını algılar

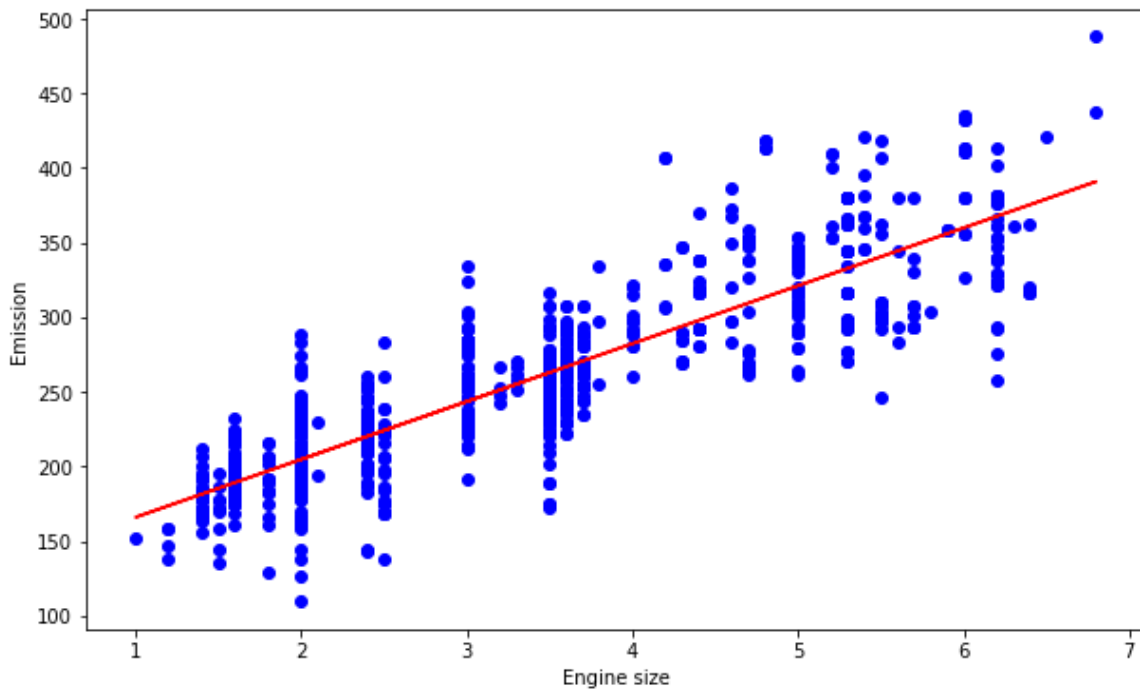
- Bağımsız ve bağımlı değişkenlerin ortalamaları arasındaki ilişkiye de bakar.
- Doğrusal regresyon, değişkenler arasındaki ilişkilerin tam bir açıklaması değildir.
- Değişkenler arasında yüksek bir korelasyonun varlığı, lineer bir modelin performansını önemli ölçüde etkileyebilir.

Önce gözlemler, sonra tahmin et

Doğrusal regresyonda, değişkenlerin doğrusal bir ilişkisi olup olmadığını değerlendirmek çok önemlidir. Bazı insanlar eğilime bakmadan tahminde bulunmaya çalışsa da, değişkenler arasında orta derecede güçlü bir korelasyon olduğundan emin olmak en iyisidir.

Daha önce bahsedildiği gibi, dağılım grafiğine ve korelasyon katsayısına bakmak mükemmel yöntemlerdir. Ve evet, korelasyon yüksek olsa bile dağılım grafiğine bakmak yine de daha iyidir. Kısacası, veriler görsel olarak doğrusalsa, doğrusal regresyon analizi yapılabilir.

Doğrusal regresyon, bağımlı bir değişkenin değerini tahmin etmenizi sağlarken, yeni veri noktalarını sınıflandıran veya komşularına bakarak değerlerini tahmin eden bir algoritma vardır. Buna k-en yakın komşu algoritması denir ve tembel bir öğrenme algoritmasıdır.



Grafik: Doğrusal Regresyon algoritması

Doğrusal regresyon, girdi özellikleri ile çıktı arasındaki ilişkiyi modelleyen istatistiksel bir yaklaşımdır. Girdi özelliklerine bağımsız değişkenler denir ve çıktıya bağımlı değişken adı verilir. Buradaki amacımız, girdi özelliklerine göre çıktının değerini, optimal katsayıları ile çarparak tahmin etmektir.

Doğrusal regresyonun gerçek hayattan bazı örnekleri:

- (1) Ürün satışlarını tahmin etmek.
- (2) Ekonomik büyümeyi tahmin etmek.
- (3) Petrol fiyatlarını tahmin etmek.
- (4) Yeni bir arabanın emisyonunu tahmin etmek.
- (5) Not ortalamasının üniversiteye girişler üzerindeki etkisi.

İki tür doğrusal regresyon vardır:

1. Basit Doğrusal Regresyon
2. Çok Değişkenli Doğrusal Regresyon

Basit Doğrusal Regresyon:

Basit doğrusal regresyonda, $y=ax+b$ ifadesinde çıkış ya da bağımlı değişken yalnızca bir giriş özelliğine göre tahmin edilir.

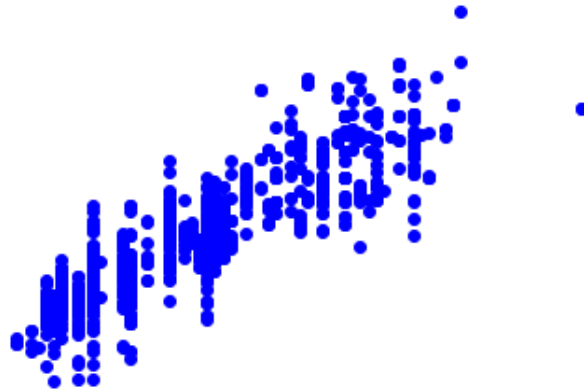
Burada

a: eğimi, girdi özelliğinin katsayısı

y: bağımlı çıkış değişkeni,

x: bağımsız giriş değişkeni, çıktının esas alındığı giriş özelliği

b: doğrunun x-eksenindeki kaymasını gösterir.

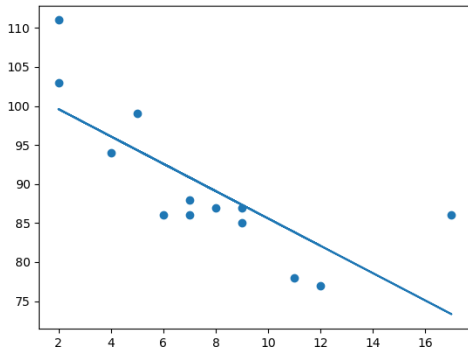


Doğrusal regresyon algoritması için veri grafiği

Adım adım uygulama:

- Gerekli kütüphaneler aktarılır: Hesaplamalar için çeşitli kütüphaneler kullanacağımızdan, bunları aktarmamız gerekiyor.
- Veritabanı dosyası okunur.
- Değerler tahmin edilirken dikkate almak istenilen özellikleri seçilir.
- Etiketli verilerin istatistiksel özellikleri belirlenir ve yorumlanır.
- Verilerin grafiği çizilir.
- Bir modelin doğruluğunu kontrol etmek için, veriler eğitim ve test veri setlerine ayrılır.
- Model eğitilir: Modelin eğitimi uygun regresyon çizgimi için katsayıların nasıl bulabileceğidir.
- En uygun çizgi çizilir.
- Test veri seti için bir tahmin fonksiyonu bulunur.
- Test verilerinin doğruluğu kontrol edilir: Gerçek değerleri veri setindeki tahmin edilen değerlerle karşılaştırılarak bir modelin doğruluğu kontrol edilebilir.
- Performans artırma süreci gerçekleştirilir.

Doğrusal regresyon, veri noktaları arasındaki ilişkiyi, tüm bunların arasından düz bir çizgi çizmek için kullanır. Bu çizgi gelecekteki değerleri tahmin etmek için kullanılabilir.



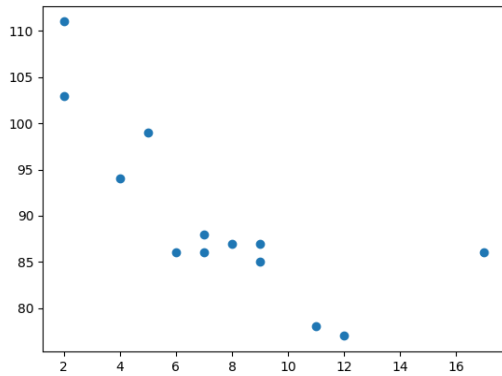
Makine Öğreniminde geleceği tahmin etmek çok önemlidir.

Python, veri noktaları arasında bir ilişki bulmak ve bir doğrusal regresyon çizgisi çizmek için yöntemlere sahiptir. Matematik formülü üzerinden gitmek yerine bu yöntemleri nasıl kullanacağınızı göstereceğiz.

Aşağıdaki örnekte, x eksenini arabanın yaşı, y eksenini ise hızı temsil etmektedir. Bir gışeden geçen 13 arabanın yaşını ve hızını kaydettik. Topladığımız verilerin doğrusal bir regresyonda kullanılıp kullanılamayacağını görelim:

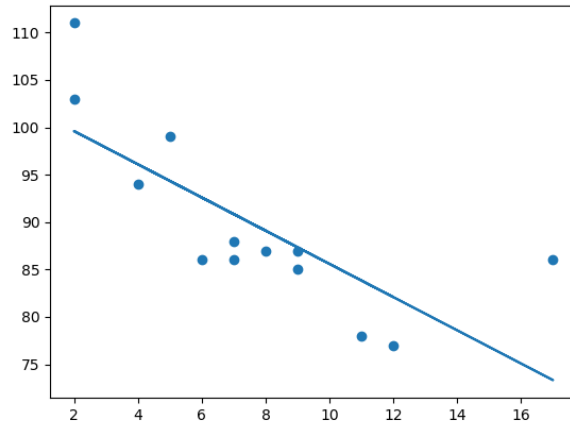
Örnek: Bir dağılımın grafiği çizilerek başlanılır.

```
import matplotlib.pyplot as plt
x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]
plt.scatter(x, y)
plt.show()
```



Example: Import scipy and draw the line of Linear Regression.

```
import matplotlib.pyplot as plt
from scipy import stats
x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]
slope, intercept, r, p, std_err = stats.linregress(x, y)
def myfunc(x):
    return slope * x + intercept
mymodel = list(map(myfunc, x))
plt.scatter(x, y)
plt.plot(x, mymodel)
plt.show()
```



Örneğin açıklanması:

İhtiyaç olan modüller içe aktarılır.

matplotlib.pyplot, plt olarak içe aktarılır.

scipy istatistiklerinden transfer edilir.

x ve y ekseninin değerlerini temsil eden diziler oluşturulur:

x = [5,7,8,7,2,17,2,9,4,11,12,9,6]

y = [99,86,87,88,111,86,103,87,94,78,77,85,86]

Doğrusal Regresyonun bazı önemli anahtar değerlerini elde eden bir komut yürütülür:

slop(eğim), intercept(kesişme), r, p, std_err = stats.linregress(x, y)

Yeni bir değer elde etmek için eğim ve kesişme değerlerini kullanan bir işlev oluşturulur. Bu yeni değer, x değerine karşılık gelen y değerinin y-ekseninde nereye yerleştirileceğini temsil edilir:

```
def myfunc(x):
```

```
    return slope * x + intercept
```

x dizisinin her değeri işlev aracılığıyla çalıştırılır. Bu, y eksenini için yeni değerlere sahip yeni bir diziyle sonuçlanacaktır:

```
mymodel = list(map(myfunc, x))
```

Orijinal dağılım grafiğini çizin: plt.scatter(x, y)

Doğrusal regresyon çizgisini çizin: plt.plot(x, mymodel)

Diyagramı görüntüleyin: plt.show()

X ekseninin değerleri ile y ekseninin değerleri arasındaki ilişkinin nasıl olduğunu bilmek önemlidir, eğer ilişki yoksa doğrusal regresyon hiçbir şeyi tahmin etmek için kullanılamaz. Bu ilişki - korelasyon katsayısı - r olarak adlandırılır. r değeri -1 ile 1 arasındadır, burada 0 hiçbir ilişki olmadığını ve 1 (ve -1) %100 ilişkili anlamına gelir. Python ve Scipy modülü sizin için bu değeri hesaplayacaktır, tek yapmanız gereken onu x ve y değerleri ile beslemek.

Örnek: Verilerim doğrusal bir regresyona ne kadar uyuyor?

```
from scipy import stats
x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]
slope, intercept, r, p, std_err = stats.linregress(x, y)
print(r)
```

Not: r korelasyon değeridir. Bağımlı ve bağımsız iki değişken arasında bir ilişki olup olmadığını belirler. $r=-0.76$ sonucu, mükemmel olmayan bir ilişki olduğunu gösterir, ancak gelecekteki tahminlerde doğrusal regresyon kullanabileceğimizi gösterir. Mükemmel ilişki, -1 ya da +1 değeridir. Mükemmel olmayan ilişki 0 değeridir.

Gelecekteki Değerleri Tahmin Edilmesi

Artık gelecekteki değerleri tahmin etmek için topladığımız bilgileri kullanabiliriz. Örnek: 10 yaşında bir arabanın hızını tahmin etmeye çalışalım. Bunu yapmak için, yukarıdaki örnekteki aynı myfunc() işlevine ihtiyacımız var:

```
def myfunc(x):
    return slope * x + intercept
```

Örnek:

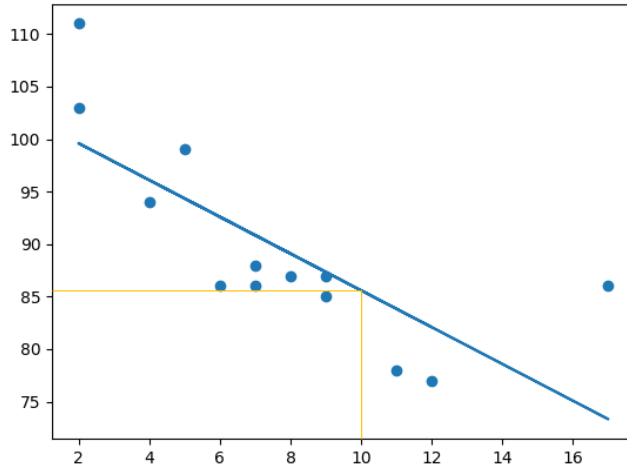
```
from scipy import stats
x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]
slope, intercept, r, p, std_err = stats.linregress(x, y)
def myfunc(x):
    return slope * x + intercept
speed = myfunc(10)
print("speed=", speed)
print("r=", r)
print("slope=", slope)
print("intercept=", intercept)
```

```
speed= 85.59308314937454
r= -0.7585915243761551
slope= -1.751287711552612
intercept= 103.10596026490066
```

Soru: aşağıdaki ifadede komutlar neden farklı hizalarda? Python'da koşul, döngüsel ve alt fonksiyonlar ifadelerinde farklı hizalarda olmazlar.

```
def myfunc(x):  
    return slope * x + intercept
```

Örnek, şemadan da okuyabileceğimiz 85.6'da bir hız öngördü:



Kötü Uyum

Doğrusal regresyonun gelecekteki değerleri tahmin etmek için en iyi yöntem olmayacağı bir örnek oluşturalım.

Example: x ve y eksenini için bu değerler, doğrusal regresyon için çok kötü bir uyumla sonuçlanmalıdır.

```
import matplotlib.pyplot as plt  
from scipy import stats  
x = [89,43,36,36,95,10,66,34,38,20,26,29,48,64,6,5,36,66,72,40]  
y = [21,46,3,35,67,95,53,72,58,10,26,34,90,33,38,20,56,2,47,15]  
slope, intercept, r, p, std_err = stats.linregress(x, y)  
def myfunc(x):  
    return slope * x + intercept  
mymodel = list(map(myfunc, x))  
plt.scatter(x, y)  
plt.plot(x, mymodel)  
plt.show()
```

speed= 40.59144864292098

r= 0.013318141542974908

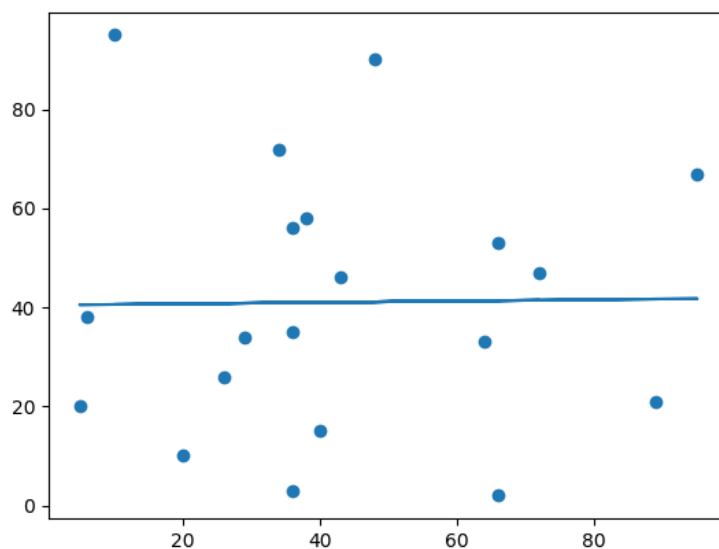
slope= 0.013916581398452628

intercept= 40.452282828936454

Not: $y = ax + b$ ifadesinde slope=a, intercept=b

Not: r değeri 0'a yakın bir değer olduğu için x ile y değerleri arasında bir ilişki yoktur.

```
import matplotlib.pyplot as plt
from scipy import stats
x = [89,43,36,36,95,10,66,34,38,20,26,29,48,64,6,5,36,66,72,40]
y = [21,46,3,35,67,95,53,72,58,10,26,34,90,33,38,20,56,2,47,15]
slope, intercept, r, p, std_err = stats.linregress(x, y)
def myfunc(x):
    return slope * x + intercept
mymodel = list(map(myfunc, x))
plt.scatter(x, y)
plt.plot(x, mymodel)
plt.show()
```



3.1.2. Çok Değişkenli Doğrusal Regresyon

Basit doğrusal regresyonda, çıktı özelliğinin değerinin tahmin edilmesi için yalnızca bir giriş özelliği dikkate alınır. Bununla birlikte, Çok Değişkenli Doğrusal Regresyonda, çıktı birden fazla giriş özelliğine göre tahmin edilebilir.

Burada $b_0, b_1, \dots, b_n$: Sabit katsayılardır.

$x_1, x_2, \dots, x_n$: Giriş değerleridir; bağımsız değişkenlerdir.

Y : Çıkış eğeri; bağımlı değişkenler; x 'değerleri ve katsayılarla bağımlıdır.

ML, hangi değerler elde edilir? Parametreler yani, $b_0, b_1, \dots, b_n$

Katsayılar çok büyükse ne yaparsınız? Katsayılar elenir. Önemsiz olanlar yani değerleri sıfıra yakın olanlar. Bunlar neden elersiniz? Gürültü...

$$Y = b_0 + b_1 * X_1 + b_2 * X_2 + \dots + b_n * X_n$$

Where,

b_0 = constant or y intercept of line

b_1, b_2, b_n = coefficient of input feature

X_1, X_2, X_n = input features

Y = output

g. Predict the values:

h. Accuracy of the model:

Now notice that here we used the same dataset for simple and multivariable linear regression. We can notice that the accuracy of multivariable linear regression is far better than the accuracy of simple linear regression.

Multiple linear regression formula

Multiple regression is like linear regression, but with more than one independent value, meaning that we try to predict a value based on two or more variables.

The formula for a multiple linear regression is:

$$y = \beta_0 + \beta_1 X_1 + \dots + \beta_n X_n + \epsilon$$

- y = the predicted value of the dependent variable
- β_0 = the y-intercept (value of y when all other parameters are set to 0)
- $\beta_1 X_1$ = the regression coefficient (β_1) of the first independent variable (X_1) (a.k.a. the effect that increasing the value of the independent variable has on the predicted y value)
- ... = do the same for however many independent variables you are testing

- $B_n X_n$ = the regression coefficient of the last independent variable
- ϵ = model error (a.k.a. how much variation there is in our estimate of y)

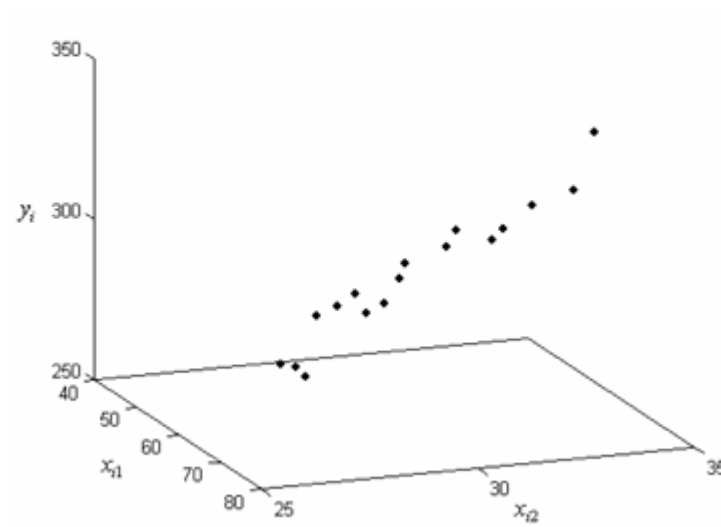
Multiple Regression

$$\begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} 1 & x_{11} & x_{12} & \dots & x_{1q} \\ 1 & x_{21} & x_{22} & \dots & x_{2q} \\ 1 & x_{31} & x_{32} & \dots & x_{3q} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & x_{n2} & \dots & x_{nq} \end{pmatrix} \begin{pmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \vdots \\ \beta_q \end{pmatrix} + \begin{pmatrix} \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \\ \vdots \\ \epsilon_n \end{pmatrix}$$

Örnek: An analyst studying a chemical process expects the yield to be affected by the levels of two factors, x_1 and x_2 . Observations recorded for various levels of the two factors are shown in the following table. The analyst wants to fit a first order regression model to the data. Interaction between x_1 and x_2 is not expected based on knowledge of similar processes. Units of the factor levels and the yield are ignored for the analysis.

Observation Number	Factor 1 (x_{i1})	Factor 2 (x_{i2})	Yield (y_i)
1	41.9	29.1	251.3
2	43.4	29.3	251.3
3	43.9	29.5	248.3
4	44.5	29.7	267.5
5	47.3	29.9	273.0
6	47.5	30.3	276.5
7	47.9	30.5	270.3
8	50.2	30.7	274.9
9	52.8	30.8	285.0
10	53.2	30.9	290.0
11	56.7	31.5	297.0
12	57.0	31.7	302.5
13	63.5	31.9	304.5
14	65.3	32.0	309.3
15	71.1	32.1	321.7
16	77.0	32.5	330.7
17	77.8	32.9	349.0

A scatter plot for the data is shown next.



The first order regression model applicable to this data set having two predictor variables is:

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \epsilon$$

where the dependent variable, Y , represents the yield and the predictor variables, x_1 and x_2 , represent the two factors respectively. The X and y matrices for the data can be obtained as:

$$X = \begin{bmatrix} 1 & 41.9 & 29.1 \\ 1 & 43.4 & 29.3 \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ 1 & 77.8 & 32.9 \end{bmatrix} \quad y = \begin{bmatrix} 251.3 \\ 251.3 \\ \cdot \\ \cdot \\ \cdot \\ 349.0 \end{bmatrix}$$

The least square estimates, $\hat{\beta}$, can now be obtained:

$$\begin{aligned}\hat{\beta} &= (X'X)^{-1}X'y \\ &= \begin{bmatrix} 17 & 941 & 525.3 \\ 941 & 54270 & 29286 \\ 525.3 & 29286 & 16254 \end{bmatrix}^{-1} \begin{bmatrix} 4902.8 \\ 276610 \\ 152020 \end{bmatrix} \\ &= \begin{bmatrix} -153.51 \\ 1.24 \\ 12.08 \end{bmatrix}\end{aligned}$$

Thus:

$$\hat{\beta} = \begin{bmatrix} \hat{\beta}_0 \\ \hat{\beta}_1 \\ \hat{\beta}_2 \end{bmatrix} = \begin{bmatrix} -153.51 \\ 1.24 \\ 12.08 \end{bmatrix}$$

and the estimated regression coefficients are $\hat{\beta}_0 = -153.51$, $\hat{\beta}_1 = 1.24$ and $\hat{\beta}_2 = 12.08$. The fitted regression model is:

$$\begin{aligned}\hat{y} &= \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2 \\ &= -153.5 + 1.24x_1 + 12.08x_2\end{aligned}$$

Adım adım uygulama:

- Gerekli kütüphaneler aktarılır.
- Veri tabanı dosyası okunur.
- X ve Y tanımlanır: X, istediğimiz bir girdi, Y ise çıktının değeri.
- Veriler bir test ve eğitim veri kümesine bölünür.
- Model eğitilir.
- Giriş özelliklerinin katsayılarını bulunur: Şimdi hangi özelliğin çıktı değişkeni üzerinde daha önemli bir etkiye sahip olduğunu bilmemiz gerekiyor. Bunun için katsayı değerleri bulunur. Negatif katsayının çıktı üzerinde ters etkisi olduğu anlamına geldiği unutulmamalıdır. Özelliklerin değeri artarsa, çıktı değeri düşer.
- Değerler tahmin edilir.
- Modelin doğruluğu test edilir: Şimdi, burada basit ve çok değişkenli doğrusal regresyon için aynı veri seti kullanıldığına dikkat edilir. Çok değişkenli doğrusal regresyonun doğruluğunun, basit doğrusal regresyonun doğruluğundan çok daha iyi olduğu fark edilebilir.

Örnek:

Çoklu regresyon, doğrusal regresyon gibidir, ancak birden fazla bağımsız değere sahiptir, yani iki veya daha fazla değişkene dayalı bir değeri tahmin etmeye çalışıyoruz.

Aşağıdaki veri setine bir göz atın, arabalar hakkında bazı bilgiler içeriyor.

Car	Model	Volume	Weight	CO2
Toyota	Aygo	1000	790	99
Mitsubishi	Space Star	1200	1160	95
Skoda	Citigo	1000	929	95
Fiat	500	900	865	90
Mini	Cooper	1500	1140	105
VW	Up!	1000	929	105
Skoda	Fabia	1400	1109	90
Mercedes	A-Class	1500	1365	92
Ford	Fiesta	1500	1112	98
Audi	A1	1600	1150	99
Hyundai	I20	1100	980	99
Suzuki	Swift	1300	990	101
Ford	Fiesta	1000	1112	99
Honda	Civic	1600	1252	94
Hundai	I30	1600	1326	97
Opel	Astra	1600	1330	97
BMW	1	1600	1365	99
Mazda	3	2200	1280	104
Skoda	Rapid	1600	1119	104
Ford	Focus	2000	1328	105
Ford	Mondeo	1600	1584	94
Opel	Insignia	2000	1428	99
Mercedes	C-Class	2100	1365	99
Skoda	Octavia	1600	1415	99
Volvo	S60	2000	1415	99
Mercedes	CLA	1500	1465	102
Audi	A4	2000	1490	104
Audi	A6	2000	1725	114
Volvo	V70	1600	1523	109
BMW	5	2000	1705	114
Mercedes	E-Class	2100	1605	115
Volvo	XC70	2000	1746	117
Ford	B-Max	1600	1235	104
BMW	2	1600	1390	108
Opel	Zafira	1600	1405	109
Mercedes	SLK	2500	1395	120

Motorun boyutuna bağlı olarak bir arabanın CO2 emisyonunu tahmin edebiliriz, ancak çoklu regresyonla, tahmini daha doğru hale getirmek için arabanın ağırlığı gibi daha fazla değişken ekleyebiliriz. Python'da işi bizim için yapacak modüllerimiz var.

Pandalar modülü içe aktararak başlanır.

```
import pandas
```

Pandas modülü, csv dosyalarını okumamıza ve bir DataFrame nesnesi döndürmemize izin verir.

```
df = pandas.read_csv("cars.csv")
```

Ardından bağımsız değerlerin bir listesi yapılır ve bu değişkene X adını verilir. Aynı şekilde bağımlı değerler de y adlı bir değişkene konur.

```
X = df[['Weight', 'Volume']]
```

```
y = df['CO2']
```

İpucu: Bağımsız değerler listesini büyük harf X ile ve bağımlı değerler listesini küçük harf y ile adlandırmak yaygındır. Sklearn modülünden bazı yöntemler kullanacağız, bu yüzden o modülü de içe aktarmamız gerekecek.

```
from sklearn import linear_model
```

Sklearn modülünden, doğrusal bir regresyon nesnesi oluşturmak için LinearRegression() yöntemini kullanacağız. Bu nesnenin, bağımsız ve bağımlı değerleri parametre olarak alan ve regresyon nesnesini ilişkiyi tanımlayan verilerle dolduran fit() adlı bir yöntemi vardır.

```
regr = linear_model.LinearRegression()
```

```
regr.fit(X, y)
```

Artık bir otomobilin ağırlığına ve hacmine dayalı olarak CO2 değerlerini tahmin etmeye hazır bir regresyon nesnemiz var.

```
#predict the CO2 emission of a car where the weight is 2300kg, and the volume is 1300cm3:  
predictedCO2 = regr.predict([[2300, 1300]])
```

Örnek: Tüm örneği çalışırken görün.

```
import pandas
```

```
from sklearn import linear_model
```

```
df = pandas.read_csv("cars.csv")
```

```
X = df[['Weight', 'Volume']]
```

```
y = df['CO2']
```

```
regr = linear_model.LinearRegression()
regr.fit(X, y)
#predict the CO2 emission of a car where the weight is 2300kg, and the volume is 1300cm3:
predictedCO2 = regr.predict([[2300, 1300]])
print(predictedCO2)
```

Result: [107.2087328]

1,3 litrelik motora ve 2300 kg ağırlığa sahip bir otomobilin kat ettiği her kilometrede yaklaşık 107 gram CO2 salacağı tahmin edilmiştir.

Katsayı:

Katsayı, bilinmeyen bir değişkenle olan ilişkiyi tanımlayan bir faktördür.

Örnek: x bir değişken ise, o zaman 2x, x'in iki katıdır. x bilinmeyen değişkendir ve 2 sayısı katsayıdır.

Bu durumda, CO2'ye karşı ağırlığın ve CO2'ye karşı hacmin katsayı değerini isteyebiliriz.

Aldığımız cevap(lar), bağımsız değerlerden birini arttırsak veya azaltırsak ne olacağını söyler.

Örnek: Regresyon nesnesinin katsayı değerlerinin bulunması.

```
import pandas
from sklearn import linear_model
df = pandas.read_csv("cars.csv")
X = df[['Weight', 'Volume']]
y = df['CO2']
regr = linear_model.LinearRegression()
regr.fit(X, y)
print(regr.coef_)
```

Result: [0.00755095 0.00780526]

Sonuç Açıklaması:

Sonuç dizisi, ağırlık ve hacmin katsayı değerlerini temsil eder.

Ağırlık: 0.00755095

Hacim: 0.00780526

Bu değerler bize ağırlık 1 kg artarsa CO2 emisyonunun 0.00755095g arttığını söyler.

Motor hacmi (Hacim) 1 cm³ artarsa CO2 emisyonu 0,00780526 g artar.

Bence bu adil bir tahmin, ama test edelim!

1300cm³ motorlu bir araba 2300kg ağırlığındaysa CO₂ emisyonunun yaklaşık 107g olacağını zaten tahmin etmiştik.

Ağırlığı 1000kg ile arttırırsak ne olur?

Örnek: Önceki örneği kopyalayın, ancak ağırlığı 2300'den 3300'e değiştirin.

```
import pandas
from sklearn import linear_model
df = pandas.read_csv("cars.csv")
X = df[['Weight', 'Volume']]
y = df['CO2']
regr = linear_model.LinearRegression()
regr.fit(X, y)
predictedCO2 = regr.predict([[3300, 1300]])
print(predictedCO2)
```

Result:

[114.75968007]

1,3 litrelik motora ve 3300 kg ağırlığa sahip bir otomobilin kat ettiği her kilometrede yaklaşık 115 gram CO₂ salacağını tahmin etmiştik.

Bu, 0.00755095 katsayısının doğru olduğunu gösterir:

$$107.2087328 + (1000 * 0.00755095) = 114.75968$$

Ölçeklendirme Özellikleri:

Verileriniz farklı değerlere ve hatta farklı ölçüm birimlerine sahip olduğunda bunları karşılaştırmak zor olabilir. Metre ile karşılaştırdığınızda kilogram nedir? Ya da zamana göre yükseklik? Bu sorunun cevabı ölçeklendirmedir. Verileri, karşılaştırması daha kolay yeni değerlere ölçekleyebiliriz. Aşağıdaki tabloya bakın, aynı veri seti ama bu sefer hacim sütunu cm^3 (1000 yerine 1.0) yerine litre cinsinden değerler içeriyor.

Car	Model	Volume	Weight	CO2
Toyota	Aygo	1	790	99
Mitsubishi	Space Star	1.2	1160	95
Skoda	Citigo	1	929	95
Fiat	500	0.9	865	90
Mini	Cooper	1.5	1140	105
VW	Up!	1	929	105
Skoda	Fabia	1.4	1109	90
Mercedes	A-Class	1.5	1365	92
Ford	Fiesta	1.5	1112	98
Audi	A1	1.6	1150	99
Hyundai	I20	1.1	980	99
Suzuki	Swift	1.3	990	101
Ford	Fiesta	1	1112	99
Honda	Civic	1.6	1252	94
Hundai	I30	1.6	1326	97
Opel	Astra	1.6	1330	97
BMW	1	1.6	1365	99
Mazda	3	2.2	1280	104
Skoda	Rapid	1.6	1119	104
Ford	Focus	2	1328	105
Ford	Mondeo	1.6	1584	94
Opel	Insignia	2	1428	99
Mercedes	C-Class	2.1	1365	99
Skoda	Octavia	1.6	1415	99
Volvo	S60	2	1415	99
Mercedes	CLA	1.5	1465	102
Audi	A4	2	1490	104
Audi	A6	2	1725	114
Volvo	V70	1.6	1523	109
BMW	5	2	1705	114
Mercedes	E-Class	2.1	1605	115
Volvo	XC70	2	1746	117

Ford	B-Max	1.6	1235	104
BMW	2	1.6	1390	108
Opel	Zafira	1.6	1405	109
Mercedes	SLK	2.5	1395	120

1.0 hacmini 790 ağırlığı ile karşılaştırmak zor olabilir, ancak her ikisini de karşılaştırılabilir değerlere ölçeklendirirsek, bir değer diğerine kıyasla ne kadar olduğunu kolayca görebiliriz. Verileri ölçeklendirmek için farklı yöntemler vardır, bu derste standardizasyon adı verilen bir yöntem kullanacağız. Standardizasyon yöntemi şu formülü kullanır:

$$z = (x - u) / s$$

z'nin yeni değer olduğu yerde, x orijinal değerdir, u ortalamadır ve s standart sapmadır. Ağırlık sütununu yukarıdaki veri kümesinden alırsanız, ilk değer 790'dır ve ölçeklenen değer şöyle olacaktır: $(790 - 1292.23) / 238.74 = -2.1$

Hacim sütununu yukarıdaki veri kümesinden alırsanız, ilk değer 1.0'dır ve ölçeklenen değer şöyle olacaktır: $(1.0 - 1.61) / 0.38 = -1.59$

Artık 790'ı 1.0 ile karşılaştırmak yerine -2.1 ile -1.59'u karşılaştırabilirsiniz. Bunu manuel olarak yapmanız gerekmez, Python sklearn modülü, veri kümelerini dönüştürmek için yöntemlerle bir Scaler nesnesi döndüren StandardScaler() adlı bir yonteme sahiptir.

Örnek: Ağırlık ve Hacim sütunlarındaki tüm değerleri ölçeklendirin.

```
import pandas
from sklearn import linear_model
from sklearn.preprocessing import StandardScaler
scale = StandardScaler()
df = pandas.read_csv("cars1.csv")
X = df[['Weight', 'Volume']]
scaledX = scale.fit_transform(X)
print(scaledX)
```

Result: İlk iki değer, hesaplamalarımıza karşılık gelen -2.1 ve -1.59 olduğuna dikkat edin..

```
[[-2.10389253 -1.59336644]
 [-0.55407235 -1.07190106]
 [-1.52166278 -1.59336644]
 [-1.78973979 -1.85409913]
 [-0.63784641 -0.28970299]
 [-1.52166278 -1.59336644]
 [-0.76769621 -0.55043568]
 [ 0.3046118 -0.28970299]]
```

```
[-0.7551301 -0.28970299]
[-0.59595938 -0.0289703 ]
[-1.30803892 -1.33263375]
[-1.26615189 -0.81116837]
[-0.7551301 -1.59336644]
[-0.16871166 -0.0289703 ]
[ 0.14125238 -0.0289703 ]
[ 0.15800719 -0.0289703 ]
[ 0.3046118 -0.0289703 ]
[-0.05142797  1.53542584]
[-0.72580918 -0.0289703 ]
[ 0.14962979  1.01396046]
[ 1.2219378 -0.0289703 ]
[ 0.5685001  1.01396046]
[ 0.3046118  1.27469315]
[ 0.51404696 -0.0289703 ]
[ 0.51404696  1.01396046]
[ 0.72348212 -0.28970299]
[ 0.8281997  1.01396046]
[ 1.81254495  1.01396046]
[ 0.96642691 -0.0289703 ]
[ 1.72877089  1.01396046]
[ 1.30990057  1.27469315]
[ 1.90050772  1.01396046]
[-0.23991961 -0.0289703 ]
[ 0.40932938 -0.0289703 ]
[ 0.47215993 -0.0289703 ]
[ 0.4302729  2.31762392]]
```

CO2 Değerlerinin Tahmin Edilmesi:

Görev, yalnızca ağırlığını ve hacmini bildiğiniz bir arabadan kaynaklanan CO2 emisyonunu tahmin etmektir. Veri seti ölçeklendiğinde, değerleri tahmin ederken ölçeği kullanmanız gerekecektir.

Örnek: 2300 kilogram ağırlığındaki 1,3 litrelik bir arabanın CO2 emisyonunu tahmin edin.

```
import pandas
from sklearn import linear_model
from sklearn.preprocessing import StandardScaler
scale = StandardScaler()
df = pandas.read_csv("cars1.csv")
X = df[['Weight', 'Volume']]
```

```
y = df['CO2']
scaledX = scale.fit_transform(X)
regr = linear_model.LinearRegression()
regr.fit(scaledX, y)
scaled = scale.transform([[2300, 1.3]])
predictedCO2 = regr.predict([scaled[0]])
print(predictedCO2)
```

Result: [107.2087328]

Modelin Değerlendirilmesi:

Makine Öğreniminde, ağırlığı ve motor boyutunu bildiğimizde bir arabanın CO2 emisyonunu tahmin ettiğimiz önceki bölümde olduğu gibi, belirli olayların sonucunu tahmin etmek için modeller oluşturulur. Modelin yeterince iyi olup olmadığını ölçmek için Eğitim/Test adlı bir yöntem kullanılır.

Eğitim / Test

Eğitim / Test, modelin doğruluğunu ölçmek için kullanılan bir yöntemdir. Eğitim/Test olarak adlandırılır çünkü veri kümesi iki kümeye bölünür: bir eğitim kümesi ve bir test kümesi. Söz gelimi, eğitim için %80 ve test için %20.

Model eğitim seti kullanılarak eğitilir. Test seti kullanılarak model test edilir. Modeli eğitmek, modeli oluşturmak anlamına gelir. Modeli test etmek, modelin doğruluğunu test etmek anlamına gelir.

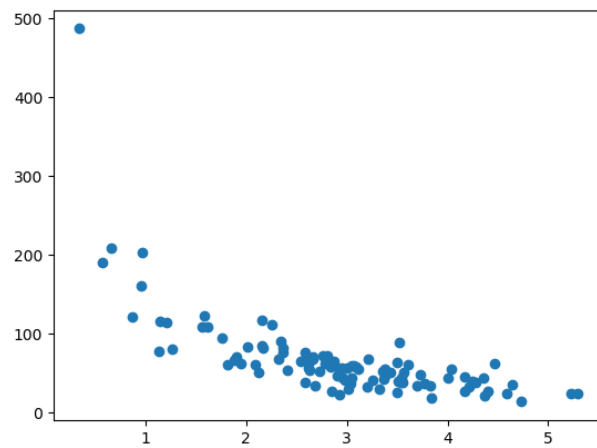
Örnek: Bir veri kümesiyle ve Test etmek istenilen bir veri seti ile başlanır. Veri seti bir mağazadaki 100 müşteriyi ve onların alışveriş alışkanlıklarını gösterebilir.

```
import numpy
import matplotlib.pyplot as plt
numpy.random.seed(2)
x = numpy.random.normal(3, 1, 100)
y = numpy.random.normal(150, 40, 100) / x
plt.scatter(x, y)
plt.show()
```

Sonuç:

X eksenini, satın alma yapmadan önce gezinme süresini dakika olarak temsil etsin.

Y eksenini, satın alma işlemine harcanan para miktarını temsil etsin.



Eğitim/Test olarak bölme:

Eğitim seti, orijinal verilerin %80'inden rastgele bir seçim olmalıdır. Test seti kalan %20 olmalıdır.

```
train_x = x[:80]
```

```
train_y = y[:80]
```

```
test_x = x[80:]
```

```
test_y = y[80:]
```

Eğitim Setini Görüntülenmesi:

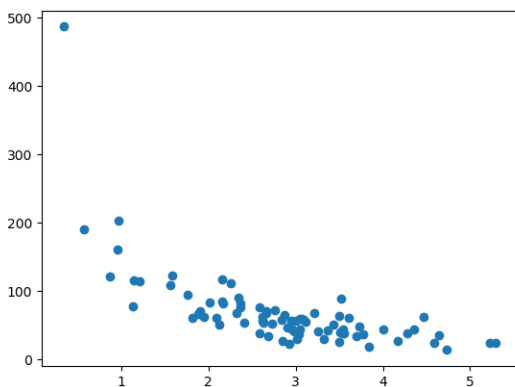
Eğitim seti ile aynı dağılım grafiğinde görüntülenirç

Example:

```
plt.scatter(train_x, train_y)
```

```
plt.show()
```

Result: It looks like the original data set, so it seems to be a fair selection.

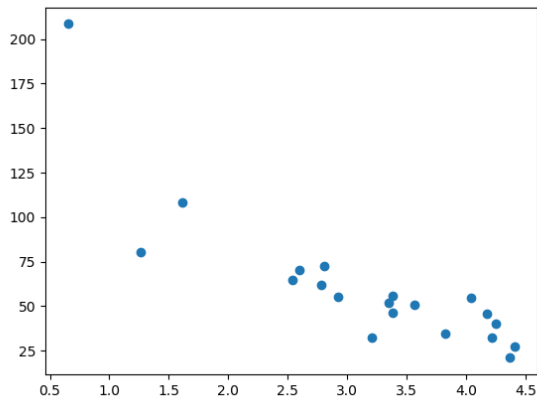


Test Setini Göster: Test setinin tamamen farklı olmadığından emin olmak için test setine de bir göz atacağız.

Example:

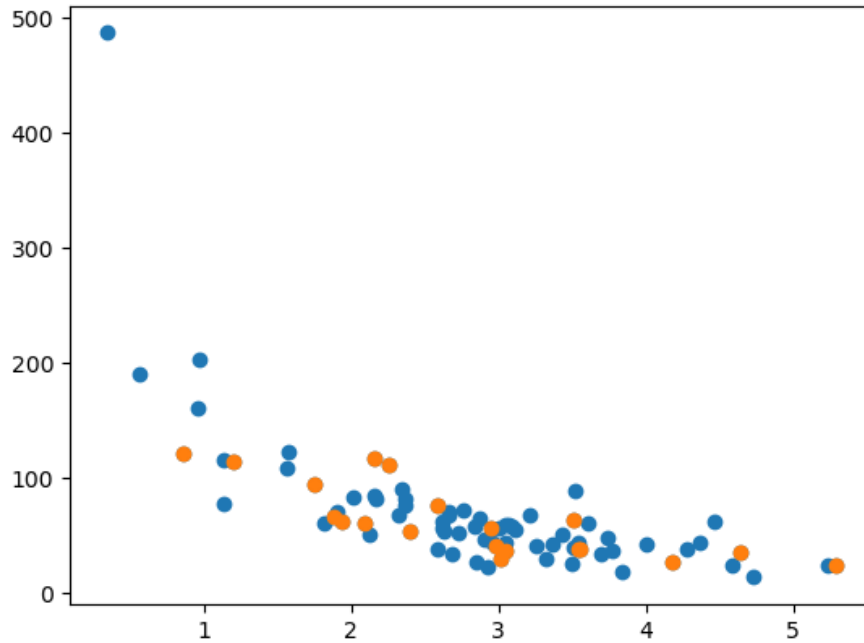
```
plt.scatter(test_x, test_y)  
plt.show()
```

Sonuç: Test seti aynı zamanda orijinal veri setine benziyor.



Eğitim ve test verilerinin birlikte çizilmesi:

```
import numpy  
import matplotlib.pyplot as plt  
numpy.random.seed(2)  
x = numpy.random.normal(3, 1, 100)  
y = numpy.random.normal(150, 40, 100) / x  
  
train_x = x[:80]  
train_y = y[:80]  
plt.scatter(train_x, train_y)  
  
test_x = x[80:]  
test_y = y[80:]  
plt.scatter(test_x, test_y)  
  
plt.show()
```



Veri Kümesini Matematiksel Modele Uydurulması:

Veri seti neye benziyor? Bence en uygun olanı bir polinom regresyonu olur, bu yüzden bir polinom regresyon eğrisi çizelim. Veri noktaları arasında bir çizgi çizmek için matplotlib modülünün plot() yöntemini kullanırız.

Örnek: Veri noktalarından bir polinom regresyon çizgisi çizilmesi.

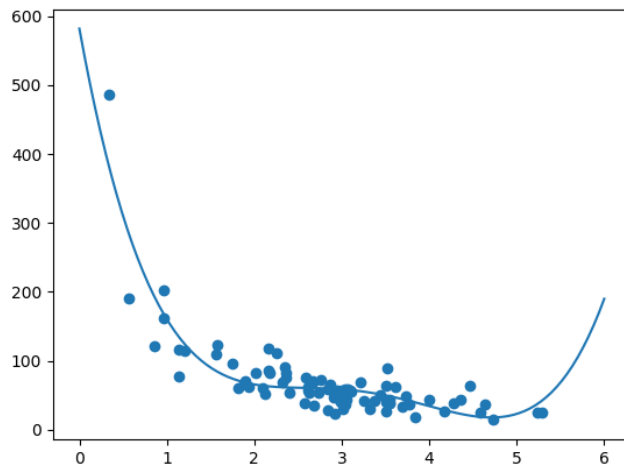
```
import numpy
import matplotlib.pyplot as plt
numpy.random.seed(2)

x = numpy.random.normal(3, 1, 100)
y = numpy.random.normal(150, 40, 100) / x

train_x = x[:80]
train_y = y[:80]

test_x = x[80:]
test_y = y[80:]

mymodel = numpy.poly1d(numpy.polyfit(train_x, train_y, 4))
myline = numpy.linspace(0, 6, 100)
plt.scatter(train_x, train_y)
plt.plot(myline, mymodel(myline))
plt.show()
```



Sonuç, veri kümesinin dışındaki değerleri tahmin etmeye çalışırsak bize bazı garip sonuçlar verecek olsa da, bir polinom regresyonuna uyan veri kümesi önerimi destekleyebilir. Örnek: satır, mağazada 6 dakika geçiren bir müşterinin 200 değerinde bir satın alma yapacağını gösterir. Bu muhtemelen fazla uydurmanın bir işaretidir. Peki ya R-kare puanı? R-kare puanı, veri setimin modele ne kadar iyi uyduğunun iyi bir göstergesidir.

R^2

R-kare olarak da bilinen R^2 'yi hatırlıyor musunuz? X eksenini ile y eksenini arasındaki ilişkiyi ölçer ve 0 ile 1 arasında değişir; burada 0, ilişki yok ve 1 tamamen ilişkili anlamına gelir.

sklearn modülünde bu ilişkiyi bulmamıza yardımcı olacak `r2_score()` adında bir metod vardır. Bu durumda müşterinin mağazada kaldığı dakikalar ile ne kadar para harcadıkları arasındaki ilişkiyi ölçmek istiyoruz.

Örnek: Eğitim verileri bir polinom regresyonuna ne kadar uyuyor?

```
import numpy
from sklearn.metrics import r2_score
numpy.random.seed(2)

x = numpy.random.normal(3, 1, 100)
y = numpy.random.normal(150, 40, 100) / x

train_x = x[:80]
train_y = y[:80]

test_x = x[80:]
test_y = y[80:]
```

```
mymodel = numpy.poly1d(numpy.polyfit(train_x, train_y, 4))
r2 = r2_score(train_y, mymodel(train_x))
print(r2)
```

Not: 0.799 sonucu, bir olumlu ilişki olduğunu gösterir.

Test Setini getirilmesi:

Şimdi, en azından eğitim verileri söz konusu olduğunda, tamam olan bir model yaptık. Şimdi aynı sonucu verip vermediğini görmek için modeli test verileriyle de test etmek istiyoruz.

Örnek: Test verilerini kullanırken R2 puanını bulalım.

```
import numpy
from sklearn.metrics import r2_score
numpy.random.seed(2)

x = numpy.random.normal(3, 1, 100)
y = numpy.random.normal(150, 40, 100) / x

train_x = x[:80]
train_y = y[:80]

test_x = x[80:]
test_y = y[80:]

mymodel = numpy.poly1d(numpy.polyfit(train_x, train_y, 4))
r2 = r2_score(test_y, mymodel(test_x))
print(r2)
```

Not: 0.809 sonucu, modelin test kümesine de uyduğunu gösterir ve modeli gelecekteki değerleri tahmin etmek için kullanabileceğimizden eminiz.

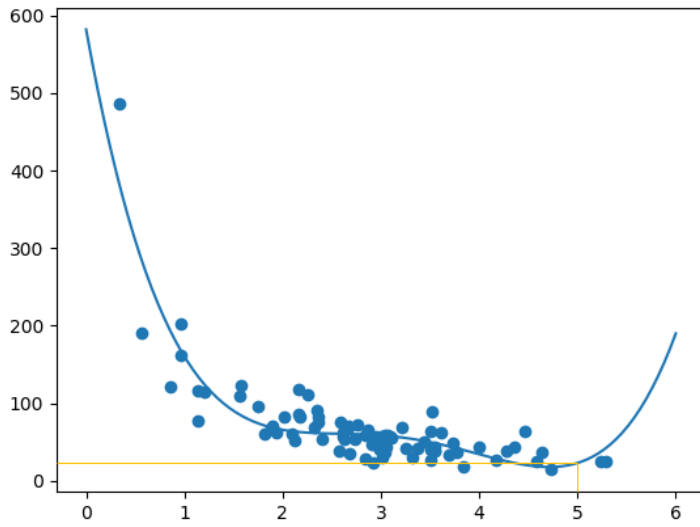
Tahmin Değerleri:

Artık modelimizin tamam olduğunu belirlediğimize göre, yeni değerleri tahmin etmeye başlayabiliriz.

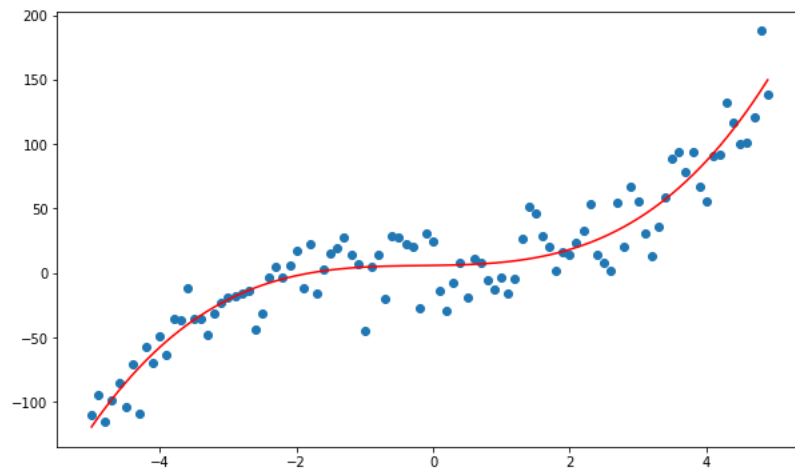
Örnek: Satın alan bir müşteri, dükkanda 5 dakika kalırsa ne kadar para harcar?

```
print(mymodel(5))
```

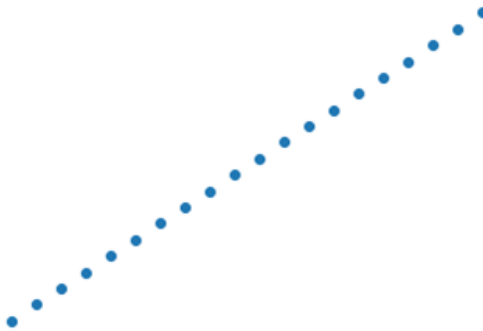
Örnek, şemaya karşılık geldiği gibi, müşterinin 22.88 dolar harcamasını öngördü.



3.1.3. Polynomial Regression



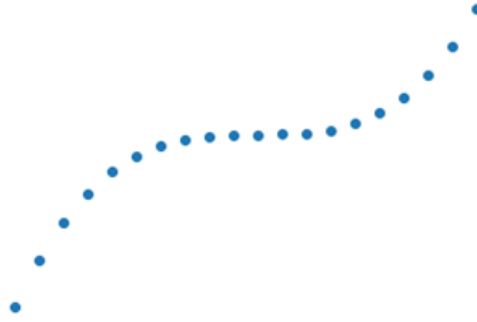
Graph for $Y=X$



Graph for $Y = X^2$



Graph for $Y = X^3$



Graph with more than one polynomials: $Y = X^3 + X^2 + X$

Below is the formula for polynomial regression:

$$Y = b_0 + b_1X + b_2X^2 + \dots + b_nX^n$$

Where,

Y = output

X = input feature

b_0, b_1, b_n = coefficients

$$\theta = (X^T X)^{-1} \cdot (X^T y)$$

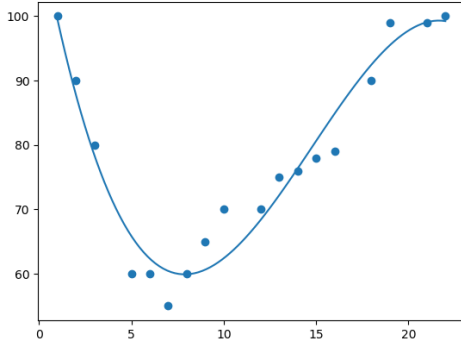
Yukarıdaki denklemde:

θ : onu en iyi tanımlayan hipotez parametreleri.

X : her örneğin giriş özelliği değeri.

Y : Her örneğin çıktı değeri.

Veri noktalarınız açıkça doğrusal bir regresyona uymuyorsa (tüm veri noktalarından geçen düz bir çizgi), polinom regresyon için ideal olabilir. Polinom regresyon, lineer regresyon gibi, veri noktalarından bir çizgi çizmenin en iyi yolunu bulmak için x ve y değişkenleri arasındaki ilişkiyi kullanır.

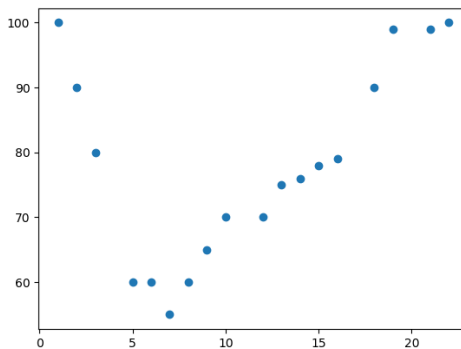


Python, veri noktaları arasında bir ilişki bulmak ve bir polinom regresyon çizgisi çizmek için yöntemlere sahiptir. Matematik formülü üzerinden gitmek yerine bu yöntemleri nasıl kullanacağınızı göstereceğiz. Aşağıdaki örnekte, belirli bir gışeden geçerken 18 araba kaydettik. Aracın hızını ve geçişin gerçekleştiği günün saatini (saati) kaydettik. X eksenini günün saatlerini ve y eksenini hızı temsil eder.

Örnek: Bir dağılım grafiği çizerek başlayın.

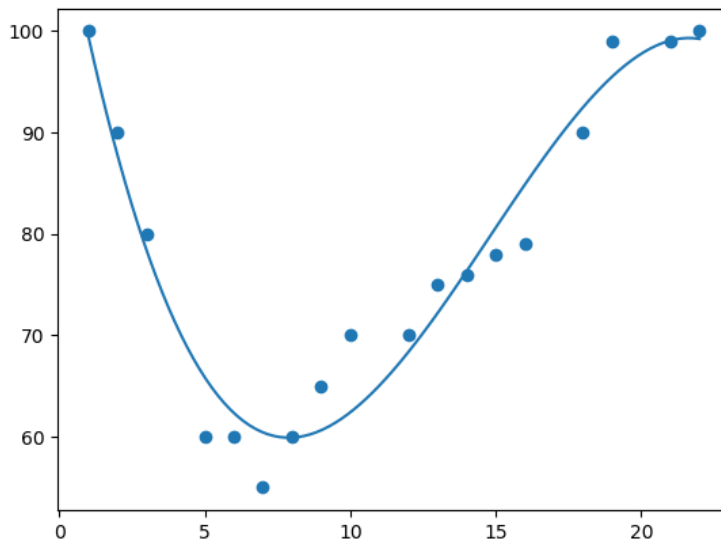
```
import matplotlib.pyplot as plt
x = [1,2,3,5,6,7,8,9,10,12,13,14,15,16,18,19,21,22]
y = [100,90,80,60,60,55,60,65,70,70,75,76,78,79,90,99,99,100]
plt.scatter(x, y)
plt.show()
```

Result:



Example: Numpy ve matplotlib'i içe aktarın, ardından Polinom Regresyon çizgisini çizin

```
import numpy
import matplotlib.pyplot as plt
x = [1,2,3,5,6,7,8,9,10,12,13,14,15,16,18,19,21,22]
y = [100,90,80,60,60,55,60,65,70,70,75,76,78,79,90,99,99,100]
mymodel = numpy.poly1d(numpy.polyfit(x, y, 3))
myline = numpy.linspace(1, 22, 100)
plt.scatter(x, y)
plt.plot(myline, mymodel(myline))
plt.show()
```



Örneğin açıklanması

İhtiyacınız olan modülleri içe aktarılır.

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

x ve y ekseninin değerlerini temsil eden dizileri oluşturulur:

```
x = [1,2,3,5,6,7,8,9,10,12,13,14,15,16,18,19,21,22]
y = [100,90,80,60,60,55,60,65,70,70,75,76,78,79,90,99,99,100]
```

NumPy, bir polinom modeli yapmamızı sağlayan bir metoda sahiptir:

```
mymodel = numpy.poly1d(numpy.polyfit(x, y, 3))
```

Ardından satırın nasıl görüneceğini belirtilir, 1. pozisyondan başlayıp 22. pozisyonda bitirilirsin:

```
myline = numpy.linspace(1, 22, 100)
```

Orijinal dağılım grafiğini çizilir:

```
plt.scatter(x, y)
```

Polinom regresyon çizgisini çizdirilir:

```
plt.plot(myline, mymodel(myline))
```

Diyagramı görüntülenir:

```
plt.show()
```

X ve y eksenı deęerleri arasındaki iliřkinin ne kadar iyi olduęunu bilmek önemlidir, eęer iliřki yoksa polinom regresyonu hiębir řeyi tahmin etmek ięin kullanılamaz.

İliřki, r-kare adı verilen bir deęerle ölçölür.

r-kare deęeri 0 ile 1 arasındadır, burada 0 hiębir iliřki olmadıęı ve 1 %100 iliřkili anlamına gelir.

Python ve Sklearn modölü bu deęeri sizin ięin hesaplayacaktır, tek yapmanız gereken onu x ve y dizileriyle beslemek:

Örnek: Verilerim bir polinom regresyonuna ne kadar uyuyor?

```
import numpy
```

```
from sklearn.metrics import r2_score
```

```
x = [1,2,3,5,6,7,8,9,10,12,13,14,15,16,18,19,21,22]
```

```
y = [100,90,80,60,60,55,60,65,70,70,75,76,78,79,90,99,99,100]
```

```
mymodel = numpy.poly1d(numpy.polyfit(x, y, 3))
```

```
print(r2_score(y, mymodel(x)))
```

Not: 0.94 sonucu, ęok iyi bir iliřkinin olduęunu ve gelecek tahminlerinde polinom regresyonunu kullanabileceęimizi göstermektedir.

Gelecekteki Deęerlerin Tahmin Edilmesi

Artık gelecekteki deęerleri tahmin etmek ięin topladıęımız bilgileri kullanabiliriz. Örnek: Saat 17.00 civarında giředen geęen bir arabanın hızını tahmin etmeye ęalıřalım: Bunu yapmak ięin, yukarıdaki örnekteki aynı mymodel dizisine ihtiyacımız var:

```
mymodel = numpy.poly1d(numpy.polyfit(x, y, 3))
```

Örnek: Saat 17:00'de geęen bir arabanın hızını tahmin edin.

```
import numpy
```

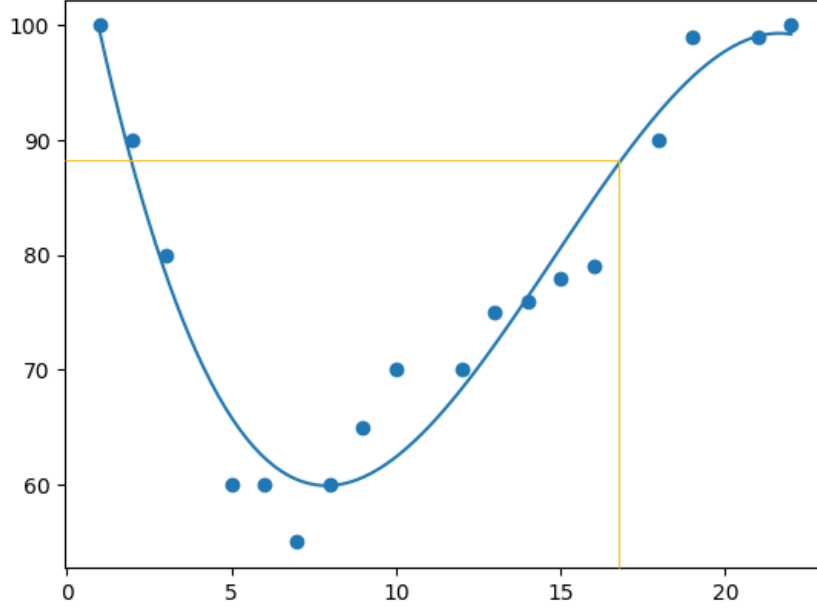
```
from sklearn.metrics import r2_score
```

```
x = [1,2,3,5,6,7,8,9,10,12,13,14,15,16,18,19,21,22]
```

```
y = [100,90,80,60,60,55,60,65,70,70,75,76,78,79,90,99,99,100]
```

```
mymodel = numpy.poly1d(numpy.polyfit(x, y, 3))  
speed = mymodel(17)  
print(speed)
```

Örnek, şemadan da okuyabileceğimiz bir hızın 88.87 olacağını öngörüldü.

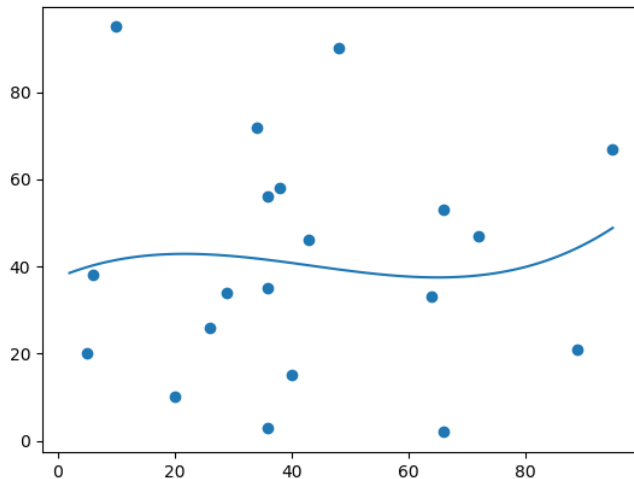


Kötü Uyum

Polinom regresyonunun gelecekteki değerleri tahmin etmek için en iyi yöntem olmayacağı bir örnek oluşturalım. Örnek: x ve y eksenini için bu değerler, polinom regresyonu için çok kötü bir uyumla sonuçlanmalıdır.

```
import numpy
import matplotlib.pyplot as plt
x = [89,43,36,36,95,10,66,34,38,20,26,29,48,64,6,5,36,66,72,40]
y = [21,46,3,35,67,95,53,72,58,10,26,34,90,33,38,20,56,2,47,15]
mymodel = numpy.poly1d(numpy.polyfit(x, y, 3))
myline = numpy.linspace(2, 95, 100)
plt.scatter(x, y)
plt.plot(myline, mymodel(myline))
plt.show()
```

Result:



Çok düşük bir r-kare değeri almalısınız.

```
import numpy
from sklearn.metrics import r2_score
x = [89,43,36,36,95,10,66,34,38,20,26,29,48,64,6,5,36,66,72,40]
y = [21,46,3,35,67,95,53,72,58,10,26,34,90,33,38,20,56,2,47,15]
mymodel = numpy.poly1d(numpy.polyfit(x, y, 3))
print(r2_score(y, mymodel(x)))
```

Sonuç: 0,00995 çok kötü bir ilişkiyi gösterir ve bize bu veri setinin polinom regresyonu için uygun olmadığını söyler.

3.1.4. Hypothesis Function for Polynomial Regression

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \dots + \beta_n x^{n-1}$$

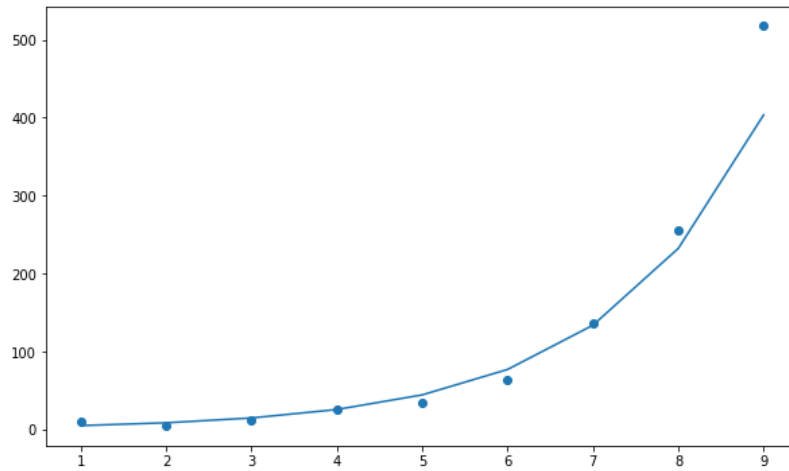
$$X = \begin{bmatrix} 1 & x_0 & x_0^2 & \dots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \dots & x_1^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \dots & x_n^{n-1} \end{bmatrix}$$

Hiperbolik fonksiyonlar ML uygulamaları nelerdir? İz sürmede ve yörende tahmin etmede.

Step by step implementation:

- Import the required libraries:
- Generate the data points: We are going to generate a dataset for implementing our polynomial regression.
- Initialize x, x^2, x^3 vectors: We are taking the maximum power of x as 3. So our X matrix will have X, X^2, X^3 .
- Column-1 of X matrix: The 1st column of the main matrix X will always be 1 because it holds the coefficient of β_0 .
- Form the complete x matrix: Look at the matrix X at the start of this implementation. We are going to create it by appending vectors.
- Transpose of the matrix:
We are going to calculate the value of θ step-by-step. First, we need to find the transpose of the matrix.
- Matrix multiplication: After finding the transpose, we need to multiply it with the original matrix. Keep in mind that we are going to implement it with a normal equation, so we have to follow its rules.
- The inverse of a matrix:
- Matrix multiplication: Finding the multiplication of transposed X and the Y vector and storing it in the temp2 variable.
- Coefficient values: To find the coefficient values, we need to multiply temp1 and temp2 .
- Store the coefficients in variables: Storing those coefficient values in different variables.
- Plot the data with curve:
- Prediction function: Now we are going to predict the output using the regression curve.
- Error function: Calculate the error using mean squared error function.
- Calculate the error:

3.1.5. Üssel Regrasyon



Üstel büyümenin bazı gerçek hayat örnekleri:

1. Kültürlerdeki mikroorganizmalar
2. Yiyeceklerin bozulması
3. İnsan Nüfusu
4. Bileşik Faiz
5. Pandemiler (Covid-19 gibi)
6. Ebola Salgını
7. İstilacı Türler
8. Ateş
9. Kansere Hücreleri
10. Akıllı Telefon Alımı ve Satışı

The formula for exponential regression is as follow:

$$Y = a + b * c^X$$

Where,

Y = output

X = input feature

a = shift value

b = y – intercept

c = base

Korelasyon katsayısı: $-1 < r < +1$

$r=-1$: kuvvetli negatif

$r=0$: zayıf

$r=+1$: kuvvetli positif

linear, $y=mx + a$

Üssel, $y=e^x$

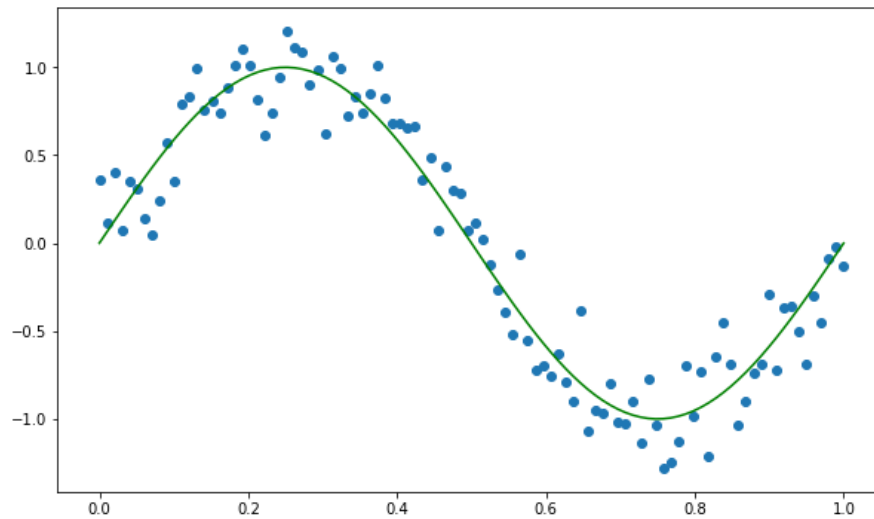
$dy/dx = e^x = y$, eğim, fonksiyonun kendisine eşittir. $x=0$, da $y=1$ olur. Eğim de 1'e eşittir. Yükseliyor. Pozitif yükseliyor. x büyüdükçe y 'de çok hızlı büyümektedir.

$$y = a + bc^x$$

Step by step implementation:

- a. Import the required libraries:
- b. Insert the data points:
- c. Implement the exponential function algorithm:
- d. Apply optimal parameters and covariance:
- e. Plot the data: Plotting the data with the coefficients found.
- f. Check the accuracy of the model:

3.1.6. Sinusoidal Regression



Some real-life examples of sinusoidal regression:

1. Müzik dalgalarının oluşturulması.
2. Ses, dalgalar halinde hareket eder.
3. Yapılarda trigonometrik fonksiyonlar.
4. Uzay uçuşlarında kullanılır.
5. GPS konum hesaplamaları.
6. Mimari.
7. Elektrik akımı.
8. Radyo yayını.
9. Okyanusun alçak ve yüksek gelgitler.
10. Binalar.

$$Y = A * \sin(B(X + C)) + D$$

Where,

A = amplitude

$Period = 2 * \pi / B$

$Period = \text{length of one cycle}$

C = phase shift (In radian)

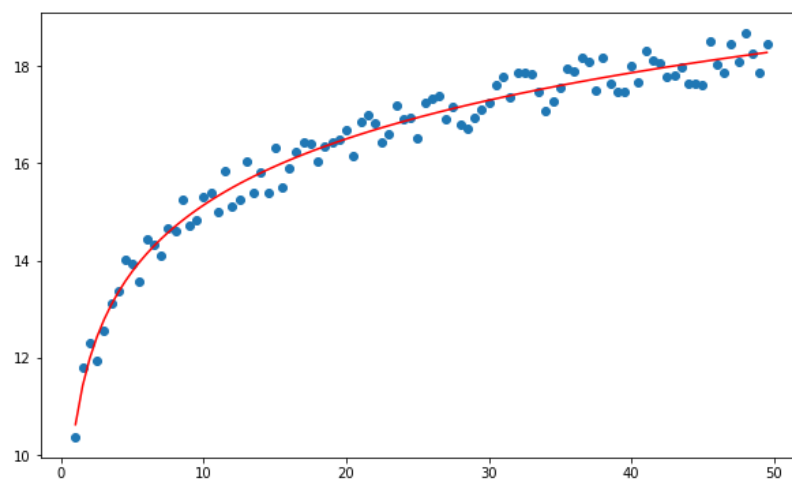
D = vertical shift

Step by step implementation:

- a. Generating the dataset
- b. Applying a sine function
- c. Why does a sinusoidal regression perform better than linear regression?

If we check the accuracy of the model after fitting our data with a straight line, we can see that the accuracy in prediction is less than that of sine wave regression. That is why we use sinusoidal regression.

3.1.7. Logarithmic Regression



Graph for a logarithmic regression

Some real-life examples of logarithmic growth:

1. The magnitude of earthquakes.
2. The intensity of sound.
3. The acidity of a solution.
4. The pH level of solutions.
5. Yields of chemical reactions.
6. Production of goods.
7. Growth of infants.

Sometimes we have data that grows exponentially in the statement, but after a certain point, it goes flat. In such a case, we can use a logarithmic regression.

$$Y = a + b * \ln(X)$$

Where,

Y = output

X = input feature

a = the line/curve always passes through (1, a)

b = controls the rate of growth or decay

Step by step implementation:

- a. Import required libraries:
- b. Generating the dataset:
- c. The first column of our matrix X :
- d. Reshaping X:
- e. Going with the Normal Equation formula:
- f. Forming the main matrix X:
- g. Finding the transpose matrix:
- h. Performing matrix multiplication:
- i. Finding the inverse:
- j. Matrix multiplication:
- k. Finding the coefficient values:
- l. Plot the data with the regression curve:

4. Lojistik Regresyon

Bir veya birden fazla bağımsız değişkeni bulunan ve bir sonucu belirlemek için kullanılan istatistik yöntemidir. Var olan bir veri kümesinin analizi sonucu iki olası sonuçtan seçim yapılması imkanı verir. Sınıflandırma problemlerinde kullanılır. Lojistik regresyon, ikili(binary) 1 veya 0 olarak kodlanmış verileri içerir.

Lojistik regresyon, iki veri faktörü arasındaki ilişkileri bulmak için matematikten yararlanan bir veri analizi tekniğidir. Lojistik regresyon, daha sonra bağımsız değişken bağımlı değişkene dayalı faktörlerden birinin değerini tahmin etmek için bu ilişkiyi kullanır. Bağımsız değişken değişkenleri bağımlı değişkenler var. Bunları ikil ya da çoklu gruplandırılır. Yeni bir bağımsız değişken geldiğinde bunun hangi gruba ait olduğu belirlenir. Tahminin genellikle evet ya da hayır gibi sınırlı sayıda sonucu vardır.

Örneğin, web sitesi ziyaretçinizin alışveriş sepetindeki ödeme düğmesine tıklayıp tıklamayacağını tahmin etmek istediğinizi varsayalım. Lojistik regresyon analizi, web sitesinde harcanan zaman ve sepetteki ürün sayısı gibi geçmiş ziyaretçi davranışlarına bakar. Geçmişte, ziyaretçiler sitede beş dakikadan fazla zaman geçirdiyse ve sepete üçten fazla ürün eklediyse ödeme düğmesine tıkladıklarını belirler. Lojistik regresyon işlevi bu bilgiyi kullanarak daha sonra yeni bir web sitesi ziyaretçisinin davranışını tahmin edebilir.

Lojistik regresyon neden önemlidir?

Lojistik regresyon, yapay zeka ve makine öğrenimi (AI/ML) alanında önemli bir tekniktir. **ML modelleri (Algoritmaları), insan müdahalesi olmadan karmaşık veri işleme görevlerini gerçekleştirmek için eğitebileceğiniz yazılım programlarıdır.** Lojistik regresyon kullanılarak oluşturulan ML modelleri, kuruluşların iş verilerinden eyleme dönüştürülebilir öngörüler elde etmelerine yardımcı olur. Bu bilgileri operasyonel maliyetleri azaltmak, verimliliği artırmak ve daha hızlı ölçeklendirmek amacıyla tahmine dayalı analiz için kullanabilirler. Örneğin, işletmeler, çalışanların elde tutulmasını artıran veya daha kârlı ürün tasarımına yol açan kalıpları ortaya çıkarabilir.

Diğer ML tekniklerine göre lojistik regresyon kullanmanın bazı avantajları aşağıda verilmiştir:

- Basitlik: Lojistik regresyon modelleri matematiksel olarak diğer ML yöntemlerine göre daha az karmaşıktır. Bu nedenle, ekibinizdeki hiç kimsenin derinlemesine ML uzmanlığı olmasa bile bunları uygulayabilirsiniz.
- Hız: Lojistik regresyon modelleri, bellek ve işlem gücü gibi daha az hesaplama kapasitesine ihtiyaç duydukları için büyük hacimli verileri yüksek hızda işleyebilir. Bu

da onları, ML projelerine başlayan kuruluşların hızlı kazançlar elde etmesi için ideal kılar.

- **Esneklik:** Lojistik regresyon, iki veya daha fazla sınırlı sonucu olan soruların yanıtlarını bulmak için kullanılabilir. Ayrıca verileri önceden işlemek için de kullanılabilir. Örneğin, banka işlemleri gibi çok çeşitli değerlere sahip verileri lojistik regresyon kullanarak daha küçük, sınırlı bir değer aralığında sıralayabilirsiniz. Daha sonra daha doğru analiz için diğer ML tekniklerini kullanarak bu küçük veri kümesini işleyebilirsiniz.
- **Görünürlük:** Lojistik regresyon analizi, geliştiricilere dahili yazılım süreçlerinde diğer veri analizi tekniklerinden daha fazla görünürlük sağlar. Hesaplamalar daha az karmaşık olduğundan sorun giderme ve hata düzeltme de daha kolaydır.

Lojistik regresyon uygulamaları nelerdir?

Lojistik regresyon birçok farklı sektörde birkaç gerçek dünya uygulamasına sahiptir.

Üretim: İmalat şirketleri, makinelerde parça arızası olasılığını tahmin etmek için lojistik regresyon analizini kullanır. Daha sonra gelecekteki arızaları en aza indirmek için bu tahmine dayalı olarak bakım programları planlarlar.

Sağlık hizmetleri: Tıbbi araştırmacılar, **hastalarda hastalık olasılığını tahmin ederek önleyici bakım ve tedaviyi planlar. Aile öyküsünün veya genlerin hastalıklar üzerindeki etkisini karşılaştırmak için lojistik regresyon modelleri kullanırlar.**

Finans: Finansal şirketlerin dolandırıcılık için finansal işlemleri analiz etmesi ve kredi başvurularını ve sigorta uygulamalarını risk açısından değerlendirmesi gerekir. Lojistik regresyon modellerinin **yüksek riskli veya düşük riskli ve dolandırıcılık olan ya da olmayan gibi ayrı sonuçları olduğundan bu sorunlar lojistik regresyon modeli için uygundur.**

Pazarlama: Çevrimiçi reklamcılık araçları, **kullanıcıların bir reklama tıklayıp tıklamayacağını tahmin etmek için lojistik regresyon modelini kullanır.** Sonuç olarak pazarlamacılar, farklı kelimelere ve resimlere verilen kullanıcı yanıtlarını analiz edebilir ve müşterilerin etkileşimde bulunacağı yüksek performanslı reklamlar oluşturabilir.

Regresyon analizi nasıl çalışır?

Lojistik regresyon, veri bilimcilerin makine öğreniminde (ML) yaygın olarak kullandığı birkaç farklı regresyon analizi tekniğinden biridir. Lojistik regresyonu anlamak için öncelikle temel regresyon analizini anlamalıyız. Aşağıda regresyon analizinin nasıl çalıştığını göstermek için bir doğrusal regresyon analizi örneği verilmiştir.

Soruyu tanımlanır: Herhangi bir veri analizi, bir iş sorusuyla başlar. Lojistik regresyon için belirli sonuçları elde etmek üzere soruyu çerçevelemelisiniz:

- Yağmurlu günler aylık satışlarımızı etkiler mi? (evet ya da hayır)
- Müşteri ne tür bir kredi kartı etkinliği gerçekleştiriyor? (yetkili, dolandırıcı veya potansiyel olarak dolandırıcı)

Geçmiş verileri toplanır: Soruyu tanımladıktan sonra, dahil olan veri faktörlerini belirlemeniz gerekir. Daha sonra tüm faktörler için geçmiş verileri toplarsınız. Örneğin, yukarıda gösterilen ilk soruyu cevaplamak üzere, son üç yılda her ay için yağmurlu günlerin sayısını ve aylık satış verilerinizi toplayabilirsiniz.

Regresyon analiz modelini eğitilir: Geçmiş verileri regresyon yazılımını kullanarak işlersiniz. Yazılım, farklı veri noktalarını işler ve denklemleri kullanıp bunları matematiksel olarak bağlar. Örneğin, üç aylık yağmurlu gün sayısı 3, 5 ve 8 ise ve o aylardaki satış sayısı 8, 12 ve 18 ise, regresyon algoritması faktörleri denklemlerle birleştirecektir:

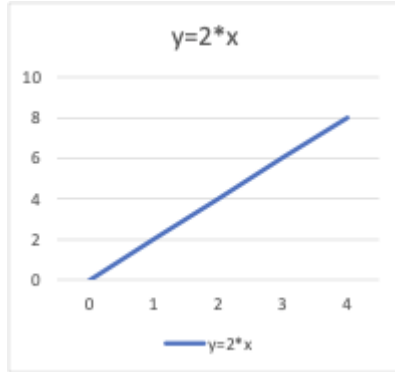
$$\text{Satış Sayısı} = 2 * (\text{Yağmurlu Gün Sayısı}) + 2$$

Bilinmeyen değerler için tahminlerde bulunur: Bilinmeyen değerler söz konusu olduğunda, yazılım bir tahmin yapmak için denklemleri kullanır. Temmuz ayında altı gün yağmur yağacağını biliyorsanız yazılım, temmuz ayının satış değerini 14 olarak tahmin edecektir.

Lojistik regresyon modeli nasıl çalışır?

Lojistik regresyon modelini anlamak için önce denklemleri ve değişkenleri anlayalım.

Denklemler: Matematikte denklemler iki değişken arasındaki ilişkiyi verir: x ve y . Bu denklemleri veya fonksiyonları, x eksenini ve y eksenini boyunca bir grafiği çizmek için farklı x ve y değerleri koyarak kullanabilirsiniz. Örneğin, $y = 2 \cdot x$ fonksiyonunun grafiğini çizerseniz aşağıda gösterildiği gibi düz bir çizgi elde edersiniz. Dolayısıyla bu fonksiyona doğrusal fonksiyon da denir.



Değişkenler : İstatistikte değişkenler, veri faktörleri veya değerleri değişen özniteliklerdir. Herhangi bir analiz için, belirli değişkenler bağımsız veya açıklayıcı değişkenlerdir. Bu öznitelikler bir sonucun sebebidir. Diğer değişkenler; bağımlı değişkenler veya yanıt değişkenleridir ve değerleri bağımsız değişkenlere bağlıdır. Genel anlamda lojistik regresyon, her iki değişkenin geçmiş veri değerlerine bakarak bağımsız değişkenlerin bir bağımlı değişkeni nasıl etkilediğini araştırır.

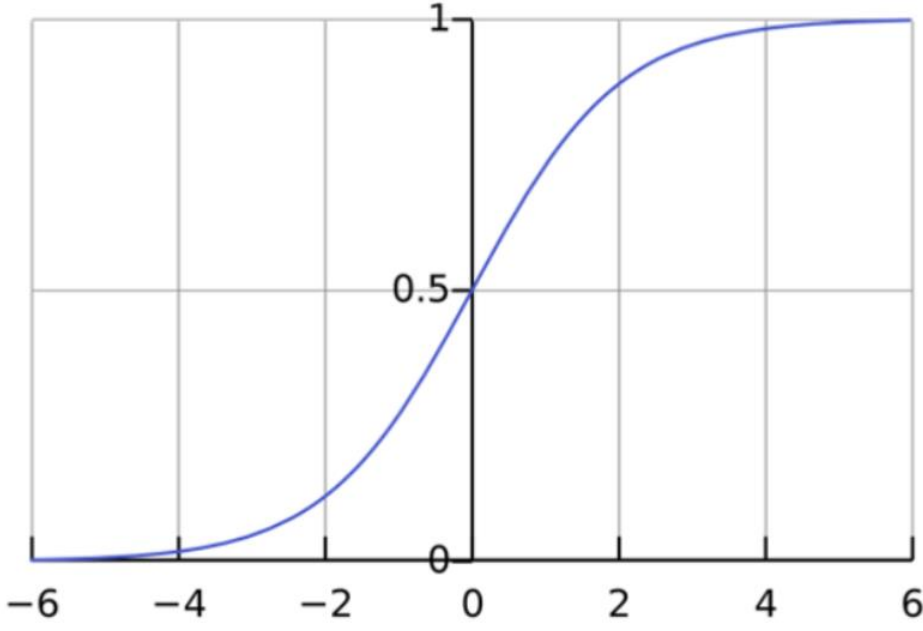
Yukarıdaki örneğimizde x , bilinen bir değere sahip olduğundan bağımsız değişken, tahmin değişkeni veya açıklayıcı değişken olarak adlandırılır. y 'nin değeri bilinmediğinden buna bağımlı değişken, sonuç değişkeni veya yanıt değişkeni denir.

Lojistik regresyon fonksiyonu

Lojistik regresyon, matematikte x ve y arasındaki denklem olarak lojistik fonksiyonu veya logit fonksiyonu kullanan istatistiksel bir modeldir. Logit fonksiyonu, y 'yi x 'in sigmoid fonksiyonu olarak eşler.

$$f(x) = \frac{1}{1 + e^{-x}}$$

Bu lojistik regresyon denklemini çizerseniz aşağıda gösterildiği gibi bir S eğrisi elde edersiniz.



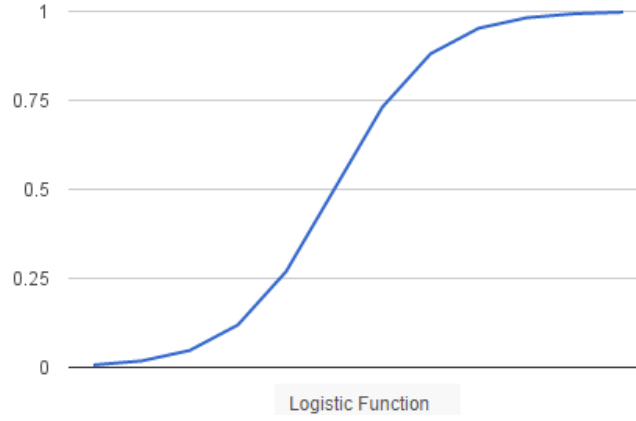
Gördüğünüz gibi, logit fonksiyonu bağımsız değişkenin değerlerinden bağımsız olarak bağımlı değişken için yalnızca 0 ile 1 arasındaki değerleri döndürür. Lojistik regresyon, bağımlı değişkenin değerini bu şekilde tahmin eder. Lojistik regresyon yöntemleri aynı zamanda çoklu bağımsız değişken ile bir bağımlı değişken arasındaki denklemleri de modellemektedir.

Sigmoid işlevi olarak da adlandırılan lojistik işlev, istatistikçiler tarafından geliştirildi, hızla yükseldi ve kapasitesini maksimuma çıkardı. 0 ile 1 arasındaki bir değerle eşleştirebilen, gerçek değerli herhangi bir sayı S şeklinde bir eğri üzerindeki değere eşleştirilir.

Giriş değerleri (x), bir çıktı değerini (y) tahmin etmek için ağırlıklar veya katsayı değerleri kullanılır. Doğrusal regresyondan önemli bir fark, modellenen çıktı değerinin sayısal bir değerden ziyade ikili değerler (0 veya 1) olmasıdır. Aşağıda örnek bir lojistik regresyon denklemi verilmiştir:

$$y = \frac{e^{b_0 + b_1 x}}{1 + e^{b_0 + b_1 x}}$$

e, doğal logaritmaların tabanıdır (Euler'in sayısı veya EXP () işlevi) ve değer, dönüştürmek istediğiniz gerçek sayısal değerdir. Aşağıda, lojistik fonksiyon kullanılarak 0 ve 1 aralığına dönüştürülmüş 0 ile 1 arasındaki sayıların bir grafiği bulunmaktadır.



y'nin tahmin edilen çıktı olduğu durumlarda, b_0 öngörü veya kesme terimidir ve b_1 , tek giriş değeri (x) için katsayıdır. Giriş verilerinizdeki her sütunun, eğitim verilerinizden öğrenilmesi gereken ilişkili bir b katsayısı (sabit bir gerçek değer) vardır. Bellekte veya bir dosyada depolayacağınız modelin gerçek temsili, denklemdeki katsayılardır.

Lojistik regresyon, varsayılan sınıfın olasılığını modeller:

Örneğin, eğer insanların cinsiyetini boylarından erkek veya kadın olarak modelliyorsak, lojistik regresyon modeli, bir kişinin boyuna göre erkek olma olasılığı olarak yazılabilir:

$$P(\text{cinsiyet} = \text{erkek} \mid \text{boy})$$

Başka bir şekilde yazıldığında, bir girdinin (X) varsayılan sınıfa ($Y = 1$) ait olma olasılığını modelliyoruz, bu şu şekilde yazılır:

$$P(X) = P(Y = 1 \mid X)$$

Olasılıklar mı tahmin ediliyor? Oysa Lojistik regresyonun bir sınıflandırma algoritmasıdır. Gerçekte bir olasılık tahmini yapmak için olasılık tahmininin ikili değerlere (0 veya 1) dönüştürülmesi gerekir. Lojistik regresyon doğrusal bir yöntemdir, ancak tahminler lojistik fonksiyon kullanılarak dönüştürülür. Bunun etkisi, tahminleri artık doğrusal regresyonla yapabildiğimiz gibi girdilerin doğrusal bir kombinasyonu olarak anlayamayacağımızdır, örneğin yukarıdan devam edersek, model şu şekilde ifade edilebilir:

$$p(X) = \frac{e^{(b_0 + b_1 * X)}}{1 + e^{(b_0 + b_1 * X)}}$$

Matematiğe çok fazla dalmak istemiyorum, ancak yukarıdaki denklemi aşağıdaki gibi döndürebiliriz (e 'yi bir taraftan diğerine doğal bir logaritma ($\ln$) ekleyerek kaldırabileceğimizi unutmayın):

$$\ln(p(X) / 1 - p(X)) = b_0 + b_1 * X$$

Soldak orana varsayılan sınıfın olasılıkları denir. Oranlar, olayın olasılığının olay olmama olasılığına bölünmesiyle hesaplanır, ör. $0.8 / (1-0.8)$ olan 4'lük olasılık var. Yani bunun yerine şunu yazabiliriz:

$$\ln(\text{oranlar}) = b_0 + b_1 * X$$

Oranlar log olarak dönüştürüldüğünden, sol tarafa log-olasılık denir.

Üssü sağa geri taşıyabilir ve şu şekilde yazabiliriz:

$$\text{oran} = e^{(b_0 + b_1 * X)}$$

Tüm bunlar, aslında modelin hala girdilerin doğrusal bir kombinasyonu olduğunu, ancak bu doğrusal kombinasyonun varsayılan sınıfın log-olasılıklarıyla ilgili olduğunu anlamamıza yardımcı olur.

Lojistik regresyon algoritmasının katsayıları (Beta değerleri b) eğitim verilerinizden tahmin edilmelidir. Bu, maksimum olasılık tahmini (maximum-likelihood estimation). kullanılarak yapılır.

Maksimum olasılık tahmini, çeşitli makine öğrenimi algoritmaları tarafından kullanılan ortak bir öğrenme algoritmasıdır, ancak verilerinizin dağıtımı hakkında varsayımlarda bulunur (verilerinizi hazırlamak hakkında konuştuğumuzda daha fazlası).

En iyi katsayılar, varsayılan sınıf için 1'e çok yakın bir değeri (örneğin erkek) ve diğer sınıf için 0'a çok yakın bir değeri (örneğin kadın) öngören bir modelle sonuçlanır. Lojistik regresyon için maksimum olasılık sezgisi, bir arama prosedürünün, model tarafından tahmin edilen olasılıklardaki hatayı verilerdekilerle en aza indiren katsayılar (Beta değerleri) için değerler aramasıdır (örneğin, veriler birincil ise 1 olasılığı sınıf).

Maksimum olasılığın matematiğine girmeyeceğiz. Eğitim verileriniz için katsayılar için en iyi değerleri optimize etmek için bir küçültme algoritmasının kullanıldığını söylemek yeterlidir. Bu genellikle pratikte verimli sayısal optimizasyon algoritması (Quasi-newton yöntemi gibi) kullanılarak uygulanır.

Lojistik öğrenirken, çok daha basit gradyan iniş algoritmasını kullanarak bunu kendiniz sıfırdan uygulayabilirsiniz.

Çok bağımsız değişkenli lojistik regresyon analizi

Çoğu durumda, birden fazla açıklayıcı değişken bağımlı değişkenin değerini etkiler. Lojistik regresyon formülleri, bu tür giriş verisi kümelerini modellemek için farklı bağımsız değişkenler arasında doğrusal bir ilişki olduğunu varsayar. Sigmoid fonksiyonunu değiştirebilir ve son çıktı değişkenini şu şekilde işleyebilirsiniz:

$$y = f(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n)$$

β sembolü, regresyon katsayısını temsil eder. Logit modeli, hem bağımlı hem de bağımsız değişkenlerin bilinen değerlerine sahip yeterince büyük bir deneysel veri kümesi verdiğinizde bu katsayı değerlerini tersine hesaplayabilir.

Logaritmik olasılıklar

Logit modeli ayrıca başarının başarısızlığa oranını veya logaritmik olasılıkları da belirleyebilir. Örneğin, arkadaşlarınızla poker oynarken 10 maçtan dördünü kazanırsanız kazanma olasılığınız altıda dördtür ve bu da başarınızın başarısızlığa oranıdır. Öte yandan kazanma olasılığınız 10'da dördtür.

Matematiksel olarak, olasılık açısından olasılıklarınız $p/(1 - p)$ ve logaritmik olasılıklarınız $(p/(1 - p))$ şeklindedir. Lojistik fonksiyonu aşağıda gösterildiği gibi logaritmik olasılıklar olarak temsil edebilirsiniz:

$$\text{Logit Function} = \log \left(\frac{p}{1-p} \right)$$

Lojistik regresyon analizi türleri nelerdir?

Bağımlı değişkenin sonuçlarına dayalı olarak lojistik regresyon analizine ilişkin üç yaklaşım vardır.

- İkili lojistik regresyon: İkili lojistik regresyon, yalnızca iki olası sonucu olan ikili sınıflandırma problemlerinde işe yarar. Bağımlı değişkenin yalnızca "evet ve hayır" veya "0 ve 1" gibi iki değeri olabilir. Lojistik fonksiyon 0 ile 1 arasında bir değer aralığını hesaplasa da ikili regresyon modeli, cevabı en yakın değerlere yuvarlar. Çoğunlukla 0,5'in altındaki cevaplar 0'a yuvarlanır ve 0,5'in üzerindeki cevaplar 1'e yuvarlanır; böylece lojistik fonksiyon ikili bir sonuç döndürür.
- Çok terimli lojistik regresyon: Çok terimli regresyon, sonuçların sayısı sınırlı olduğu sürece birkaç olası sonucu olan problemleri analiz edebilir. Örneğin, konut fiyatlarının nüfus verilerine göre %25, %50, %75 veya %100 artacağını tahmin edebilir ancak bir evin tam değerini tahmin edemez. Çok terimli lojistik regresyon, sonuç değerlerini 0 ve 1 arasındaki farklı değerlerle eşleyerek çalışır. Lojistik fonksiyon 0,1; 0,11; 0,12 vb. gibi bir dizi sürekli veri döndürebildiğinden, çok terimli regresyon da çıktıyı mümkün olan en yakın değerlere göre gruplandırır.

- Sıralı lojistik regresyon: Sıralı lojistik regresyon veya sıralı logit modeli, sayıların gerçek değerlerden ziyade sıralamaları temsil ettiği problemler için özel bir çok terimli regresyon türüdür. Örneğin, müşterilerden sizden yıl boyunca satın aldıkları ürün sayısı gibi sayısal bir değere bağlı olarak hizmetinizi kötü, orta, iyi veya mükemmel şeklinde sıralamalarını isteyen bir anket sorusuna verdikleri yanıtı tahmin etmek için sıralı regresyon kullanırsınız.

Lojistik regresyon diğer ML tekniklerine kıyasla nasıl çalışır?

İki yaygın veri analizi tekniği; doğrusal regresyon analizi ve derin öğrenmedir.

Doğrusal regresyon analizi

Yukarıda açıklandığı gibi, doğrusal regresyon, bağımlı ve bağımsız değişkenler arasındaki ilişkiyi doğrusal bir kombinasyon kullanarak modeller. Doğrusal regresyon denklemi $y = \beta_0 X_0 + \beta_1 X_1 + \beta_2 X_2 + \dots + \beta_n X_n + \epsilon$, burada β_1 ile β_n ve ϵ regresyon katsayısıdır.

Lojistik regresyon ve doğrusal regresyon karşılaştırması

Doğrusal regresyon, belirli bir bağımsız değişken kümesini kullanarak sürekli bağımlı bir değişkeni öngörür. Sürekli değişken, fiyat veya yaş gibi bir değer aralığına sahip olabilir. Dolayısıyla doğrusal regresyon, bağımlı değişkenin gerçek değerlerini tahmin edebilir. "10 yıl sonra pirinç fiyatı ne olacak?" gibi soruları yanıtlayabilir.

Doğrusal regresyonun aksine, lojistik regresyon bir sınıflandırma algoritmasıdır. Sürekli veriler için gerçek değerleri öngöremez. "Pirinç fiyatı 10 yılda% 50 artacak mı?" gibi soruları yanıtlayabilir.

Derin öğrenme

Derin öğrenme, bilgiyi analiz etmek için insan beynini simüle eden sinir ağlarını veya yazılım bileşenlerini kullanır. Derin öğrenme hesaplamaları, vektörlerin matematiksel kavramına dayanmaktadır.

Lojistik regresyon ve derin öğrenme karşılaştırması

Lojistik regresyon, derin öğrenmeye göre daha az karmaşıktır ve bilgi işlem açısından yoğunluğu daha azdır. Daha da önemlisi, derin öğrenme hesaplamaları, karmaşıklıkları ve makine odaklı doğaları gereği geliştiriciler tarafından araştırılmaz veya değiştirilemez. Diğer yandan, lojistik regresyon hesaplamaları şeffaftır ve sorunları gidermek daha kolaydır.

Lojistik Regresyon Modelini Öğrenmek:

Lojistik regresyon modeliyle tahmin yapmak, sayıları lojistik regresyon denklemine eklemek ve bir sonucu hesaplamak kadar basittir.

Bunu belirli bir örnekle somut hale getirelim. Diyelim ki bir kişinin boyuna göre erkek mi kadın mı olduğunu tahmin edebilen (tamamen hayal ürünü) bir modelimiz var. Boyu 150 cm olarak verildiğinde, erkek veya kadındır. $b_0 = -100$ ve $b_1 = 0.6$ katsayılarını öğrendik.

Yukarıdaki denklemi kullanarak, resmi olarak $P(\text{erkek} \mid \text{boy} = 150)$ 150 cm veya daha fazla bir yükseklik verilen erkeğin olasılığını hesaplayabiliriz.

$$y = e^{(b_0 + b_1 * X)} / (1 + e^{(b_0 + b_1 * X)})$$
$$y = \exp(-100 + 0.6 * 150) / (1 + \exp(-100 + 0.6 * 150))$$
$$y = 0,0000453978687$$

Kişinin erkek olma olasılığı sıfıra yakın. Pratikte olasılıkları doğrudan kullanabiliriz. Bu sınıflandırma olduğundan ve net bir cevap istediğimizden, olasılıkları ikili bir sınıf değerine yaslayabiliriz, örneğin: 0 eğer $p(\text{erkek}) < 0.5$ 1 eğer $p(\text{erkek}) > 0.5$

Artık lojistik regresyon kullanarak nasıl tahminler yapacağımızı bildiğimize göre, teknikten en iyi şekilde yararlanmak için verilerimizi nasıl hazırlayabileceğimize bakalım.

Verilerin Lojistik Regresyon için Hazırlanması:

Verilerinizdeki dağılım ve ilişkiler hakkında lojistik regresyon tarafından yapılan varsayımlar, doğrusal regresyonda yapılan varsayımlarla büyük ölçüde aynıdır. Bu varsayımları tanımlamaya yönelik çok çalışma yapılmıştır ve kesin olasılık ve istatistiksel dil kullanılmıştır. **Nihayetinde tahmine dayalı modelleme makine öğrenimi projelerinde, sonuçları yorumlamaktan çok doğru tahminler yapmaya odaklanırsınız.**

Bu nedenle, model sağlam olduğu ve iyi performans gösterdiği sürece bazı varsayımları kırabilirsiniz.

İkili Çıktı Değişkeni: Lojistik regresyon ikili (iki sınıflı) sınıflandırma problemleri için tasarlanmıştır. Varsayılan sınıfa ait olan ve 0 veya 1 sınıflandırmasına dahil edilebilen bir örneğin olasılığını tahmin edecektir.

Gürültüyü Kaldır: Lojistik regresyon, çıktı değişkeninde (y) herhangi bir hata olmadığını varsayar, aykırı değerleri ve muhtemelen yanlış sınıflandırılmış örnekleri eğitim verilerinizden kaldırmayı düşünün.

Gauss Dağılımı: Lojistik regresyon doğrusal bir algoritmadır (çıktıda doğrusal olmayan bir dönüşüm ile). Girdi değişkenleri ile çıktı arasında doğrusal bir ilişki olduğunu varsayar. Bu doğrusal ilişkiyi daha iyi ortaya çıkaran girdi değişkenlerinizin veri dönüşümleri, daha doğru bir modelle sonuçlanabilir. Örneğin, bu ilişkiyi daha iyi ortaya çıkarmak için log, root, Box-Cox ve diğer tek değişkenli dönüşümleri kullanabilirsiniz.

İlişkili Girdileri Kaldır: Doğrusal regresyon gibi, çok sayıda yüksek korelasyonlu girdiniz varsa model gereğinden fazla sığabilir. Tüm girdiler arasındaki ikili korelasyonları hesaplamayı ve yüksek korelasyonlu girdileri kaldırmayı düşünün.

Yakınsama Başarısızlığı: Katsayıları öğrenen beklenen olasılık kestirim sürecinin yakınsamada başarısız olması mümkündür. Verilerinizde yüksek düzeyde ilişkili çok sayıda giriş varsa veya veriler çok seyrekse (örneğin, giriş verilerinizde çok fazla sıfır) bu durum meydana gelebilir.

Örnek: Logistic Regression

Lojistik regresyon, sınıflandırma problemlerini çözmeyi amaçlar. Bunu, sürekli bir sonucu öngören doğrusal regresyonun aksine, kategorik sonuçları tahmin ederek yapar.

En basit durumda, binom adı verilen iki sonuç vardır; buna bir örnek, bir tümörün kötü huylu mu yoksa iyi huylu mu olduğunu tahmin etmektir. Diğer durumların sınıflandırılması gereken ikiden fazla sonucu vardır, bu duruma multinomial denir. Çok terimli lojistik regresyon için yaygın bir örnek, 3 farklı tür arasında bir iris çiçeğinin sınıfını tahmin etmektir. Burada bir binom değişkenini tahmin etmek için temel lojistik regresyon kullanacağız. Bu, yalnızca iki olası sonucu olduğu anlamına gelir.

How does it work?

In Python we have modules that will do the work for us. Start by importing the NumPy module.

```
import numpy
```

Store the independent variables in X.

Store the dependent variable in y.

Below is a sample dataset:

#X represents the size of a tumor in centimeters.

X =

```
numpy.array([3.78, 2.44, 2.09, 0.14, 1.72, 1.65, 4.92, 4.37, 4.96, 4.52, 3.69, 5.88]).reshape(-1,1)
```

#Note: X has to be reshaped into a column from a row for the LogisticRegression() function to work.

#y represents whether or not the tumor is cancerous (0 for "No", 1 for "Yes").

```
y = numpy.array([0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1])
```

We will use a method from the sklearn module, so we will have to import that module as well:

```
from sklearn import linear_model
```

From the sklearn module we will use the LogisticRegression() method to create a logistic regression object.

This object has a method called fit() that takes the independent and dependent values as parameters and fills the regression object with data that describes the relationship:

```
logr = linear_model.LogisticRegression()  
logr.fit(X,y)
```

Now we have a logistic regression object that is ready to whether a tumor is cancerous based on the tumor size:

#predict if tumor is cancerous where the size is 3.46mm:

```
predicted = logr.predict(numpy.array([3.46]).reshape(-1,1))
```

Example

See the whole example in action:

```
import numpy
```

```
from sklearn import linear_model
```

```
#Reshaped for Logistic function.
```

```
X =
```

```
numpy.array([3.78, 2.44, 2.09, 0.14, 1.72, 1.65, 4.92, 4.37, 4.96, 4.52, 3.69, 5.88]).reshape(-1,1)
```

```
y = numpy.array([0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1])
```

```
logr = linear_model.LogisticRegression()
```

```
logr.fit(X,y)
```

#predict if tumor is cancerous where the size is 3.46mm:

```
predicted = logr.predict(numpy.array([3.46]).reshape(-1,1))  
print(predicted)
```

Result: [0]

Uygulama:

Lojistik regresyon sınıflandırma problemlerini çözmeyi amaçlar. Bunu, sürekli bir sonucu öngören doğrusal regresyonun aksine, kategorik sonuçları tahmin ederek yapar. En basit durumda, bir tümörün kötü huylu mu yoksa iyi huylu mu olduğunu tahmin eden, binom olarak adlandırılan iki sonuç vardır. Diğer vakaların sınıflandırılması gereken ikiden fazla sonucu vardır, bu durumda buna çok terimli denir. Çok terimli lojistik regresyon için yaygın bir örnek, 3 farklı tür arasında bir iris çiçeğinin sınıfını tahmin etmek olabilir. Burada bir binom değişkenini tahmin etmek için temel lojistik regresyon kullanacağız. Bu, yalnızca iki olası sonucu olduğu anlamına gelir.

Nasıl çalışır?

Python'da işi yapacak modüller var. NumPy modülünü içe aktararak başlanır.

```
import numpy
```

Bağımsız değişkenleri X'te saklayın.

Bağımlı değişkeni y'de saklayın.

Aşağıda örnek bir veri seti verilmiştir:

#X represents the size of a tumor in centimeters.

X =

```
numpy.array([3.78, 2.44, 2.09, 0.14, 1.72, 1.65, 4.92, 4.37, 4.96, 4.52, 3.69, 5.88]).reshape(-1,1)
```

#Note: X has to be reshaped into a column from a row for the LogisticRegression() function to work.

#y represents whether or not the tumor is cancerous (0 for "No", 1 for "Yes").

```
y = numpy.array([0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1])
```

Sklearn modülünden bir metod kullanacağız, bu yüzden o modülü de import etmemiz gerekecek.

```
from sklearn import linear_model
```

Sklearn modülünden bir lojistik regresyon nesnesi oluşturmak için LogisticRegression() yöntemini kullanacağız. Bu nesnenin, bağımsız ve bağımlı değerleri parametre olarak alan ve regresyon nesnesini ilişkiyi tanımlayan verilerle dolduran fit() adlı bir yöntemi vardır:

```
logr = linear_model.LogisticRegression()
```

```
logr.fit(X,y)
```

Şimdi elimizde, tümör boyutuna göre bir tümörün kanserli olup olmadığına hazır olan bir lojistik regresyon nesnemiz var.

```
#predict if tumor is cancerous where the size is 3.46mm:  
predicted = logr.predict(numpy.array([3.46]).reshape(-1,1))
```

Örnek: Tüm örneği çalışırken görülmesi.

```
import numpy
```

```
from sklearn import linear_model
```

```
#Reshaped for Logistic function.
```

```
X =
```

```
numpy.array([3.78, 2.44, 2.09, 0.14, 1.72, 1.65, 4.92, 4.37, 4.96, 4.52, 3.69, 5.88]).reshape(-1,1)
```

```
y = numpy.array([0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1])
```

```
logr = linear_model.LogisticRegression()
```

```
logr.fit(X,y)
```

```
#predict if tumor is cancerous where the size is 3.46mm:
```

```
predicted = logr.predict(numpy.array([3.46]).reshape(-1,1))
```

```
print(predicted)
```

Sonuç: 4.03541657

Bu bize bir tümörün boyutu 1 mm arttıkça tümör olma olasılığının 4 kat arttığını söyler.

Olasılık

Katsayı ve kesişme değerleri, her tümörün kanserli olma olasılığını bulmak için kullanılabilir. Yeni bir değer döndürmek için modelin katsayısını ve kesişme değerlerini kullanan bir işlev oluşturulur. Bu yeni değer, verilen gözlemin bir tümör olma olasılığını temsil eder.

```
def logit2prob(logr,x):
```

```
    log_odds = logr.coef_ * x + logr.intercept_
```

```
    odds = numpy.exp(log_odds)
```

```
    probability = odds / (1 + odds)
```

```
    return(probability)
```

İşlev Açıklaması: Her bir gözlem için log-oranları bulmak için, önce doğrusal regresyondan elde edilene benzeyen bir formül oluşturulmalıdır, katsayı ve kesişim çıkarılmalıdır.

```
log_odds = logr.coef_ * x + logr.intercept_
```

Daha sonra log oranlarını oranlara dönüştürmek için log oranlarının üssü alınır.

```
odds = numpy.exp(log_odds)
```

Artık oranlarımız olduğuna göre, onu 1 artı oranlara bölerek olasılığa dönüştürebiliriz.

$probability = odds / (1 + odds)$

Şimdi her tümörün kanserli olma olasılığını bulmak için öğrendiklerimizle işlevi kullanalım.

Örnek: Tüm örneğin çalışırken görülmesi.

```
import numpy
```

```
from sklearn import linear_model
```

```
X =
```

```
numpy.array([3.78, 2.44, 2.09, 0.14, 1.72, 1.65, 4.92, 4.37, 4.96, 4.52, 3.69, 5.88]).reshape(-1,1)
```

```
y = numpy.array([0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1])
```

```
logr = linear_model.LogisticRegression()
```

```
logr.fit(X,y)
```

```
def logit2prob(logr, X):
```

```
    log_odds = logr.coef_ * X + logr.intercept_
```

```
    odds = numpy.exp(log_odds)
```

```
    probability = odds / (1 + odds)
```

```
    return(probability)
```

```
print(logit2prob(logr, X))
```

Sonuç:

```
[[0.60749955]
```

```
 [0.19268876]
```

```
 [0.12775886]
```

```
 [0.00955221]
```

```
 [0.08038616]
```

```
 [0.07345637]
```

```
 [0.88362743]
```

```
 [0.77901378]
```

```
 [0.88924409]
```

```
 [0.81293497]
```

```
 [0.57719129]
```

```
 [0.96664243]]
```

Sonuçlar Açıklanması:

3,78 0,61 3,78 cm büyüklüğünde bir tümörün kanserli olma olasılığı %61'dir.

2,44 0,19 2,44 cm büyüklüğündeki bir tümörün kanserli olma olasılığı %60'tır.

2,09 0,13 2,09 cm büyüklüğündeki bir tümörün kanserli olma olasılığı %13'tür.

5. Sınıflandırma

Bir Makine Öğrenimindeki bir sınıflandırıcı, ayırık veya sürekli özellik değerlerinin bir vektörünü giren ve tek bir ayırık değer olan sınıfı çıkaran bir sistemdir. Bir dizi ögenin sınıfını veya kategorisini tahmin etmek için sınıflandırma algoritmaları kullanılır.

Sınıflandırma algoritmaları:

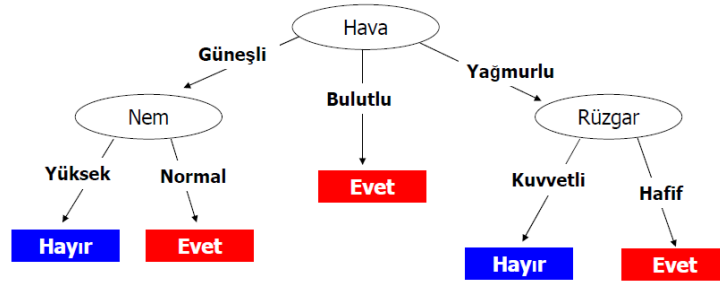
Diğer sınıflandırma tekniklerinden bazıları aşağıda verilmiştir:

- K-En Yakın Komşu Algoritması (K-Nearest Neighbour Algorithm)
- Lojistik Regresyon (Logistic Regression)
- Destek Vektör Makineleri (Support Vector Machines)
- Karar Ağaçları (Decision Tree)
- Rasgele Orman Kümeleri (Random Forests)
- Yapay Sinir Ağları (Artificial Neural Networks)
- Naive Bayes

5.1.1. Karar Ağaçları

Karar ağacı algoritması, denetimli öğrenme kategorisine girer. Hem regresyon hem de sınıflandırma problemlerini çözmek için kullanılırlar. Karar ağacı, her yaprak düğümün bir sınıf etiketine karşılık geldiği ve özniteliklerin ağacın iç düğümünde temsil edildiği sorunu çözmek için ağaç temsili kullanır. Karar ağacını kullanarak herhangi bir boole fonksiyonunu ayırık öznitelikler üzerinde temsil edebiliriz. Karar ağacı, istatistik, veri madenciliği ve makine öğrenmesinde kullanılan öngörülü modelleme yaklaşımlarından biridir.

Karar ağaçları metodu, sınıflama işlemidir. Giriş verisinin bir kümeleme algoritması yardımıyla tekrar tekrar gruplara bölünmesine dayanır. Grubun tüm elemanları aynı sınıf etiketine sahip olana kadar kümeleme işlemi derinlemesine devam eder.



Karar ağaçları çok boyutlu (özellikli) veriyi belirlenmiş şartlara bağlı olarak parçalara böler.

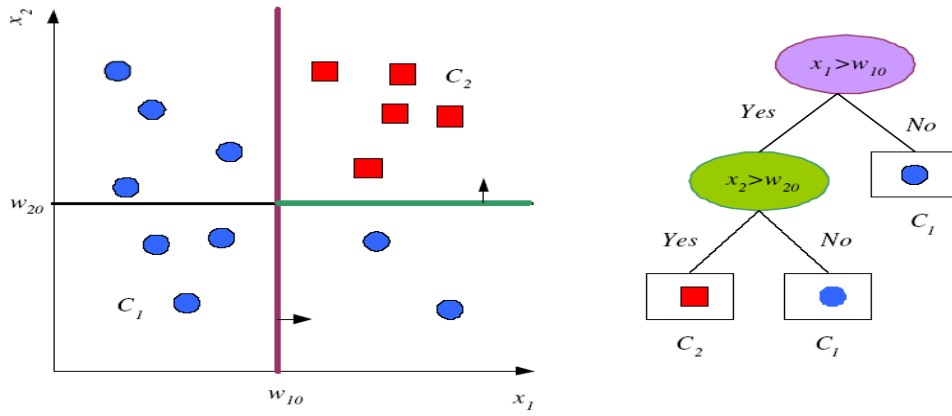
Karar ağacı tipleri ikiye ayrılır:

- Entropiye dayalı sınıflandırma ağaçları (ID3, C4.5)
- Regresyon ağaçları (CART).

Karar Ağacı Algoritması:

Karar ağaçları eğitici öğrenme için çok yaygın bir yöntemdir. Algoritmanın adımları:

- 1) Veri yığınınından öğrenme kümesi oluşturulur.
- 2) Öğrenme kümesindeki örnekleri en iyi ayıran nitelikler belirlenir.
- 3) Seçilen nitelikler ile ağacın düğümleri oluşturulur ve her bir düğümde alt düğümler veya ağacın yapraklarını oluşturulur. Alt düğümlere ait alt veri kümesinin örneklerini belirlenir
- 4) 3. adımda oluşturulan her alt veri kümesi için
 - Örneklerin hepsi aynı sınıfa aitse
 - Örnekleri bölecek nitelik kalmamışsa
 - Kalan niteliklerin değerini taşıyan örnek yoksa işlemi sonlandır. Diğer durumda alt veri kümesini ayırmak için 2. adımdan devam edilir.



Ezber (Overfitting: Aşırı Uyum): Tüm makine öğrenmesi yöntemlerinde verinin ana hatlarının modellenmesi esas alındığı için öğrenme modelinde ezberden (overfitting: aşırı uyum) kaçınılmalıdır. **Tüm karar ağaçları önlem alınmazsa ezber yapar.** Bu yüzden ağaç oluşturulurken veya oluşturulduktan sonra budama yapılmalıdır.

Ağaç Budama:

Budama, sınıflandırmaya katkısı olmayan bölümlerin karar ağacından çıkarılması işlemidir. Bu sayede karar ağacı hem sade hem de anlaşılabilir hale gelir. İki çeşit budama yöntemi vardır: Ön budama, sonradan budama.

Ön budama işlemi ağaç oluşturulurken yapılır. Bölünen nitelikler, değerleri belli bir eşik değerinin (hata toleransının) üstünde değilse o noktada ağaç bölümleme işlemi durdurulur ve o an elde bulunan kümedeki baskın sınıf etiketi, yaprak olarak oluşturulur.

Sonradan Budama: Sonradan budama işlemi ağaç oluşturulduktan sonra devreye girer. Alt ağaçları silerek yaprak oluşturma, alt ağaçları yükseltme, dal kesme şeklinde yapılabilir.

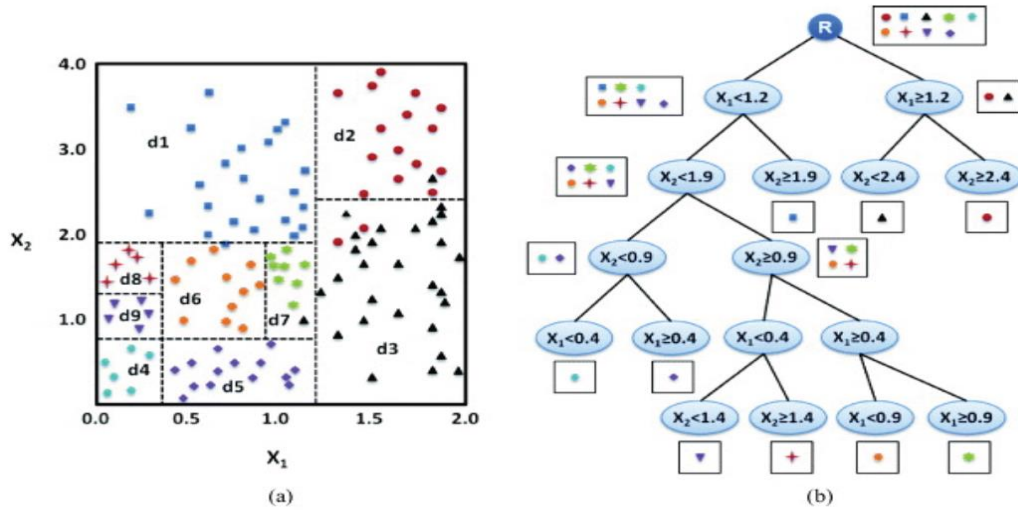
Aşırı uyumu önlemek için ağacı büyütme durdurulabilir, ancak durdurma kriteri miyop olma eğilimindedir. Bu nedenle standart yaklaşım, "dolu" bir ağaç oluşturmak ve ardından budama yapmaktır. Düğümdeki noktalar için yanlış bir sınıflandırma yapma olasılığı olduğu da unutulmamalıdır. Tüm ağaç için yanlış sınıflandırma olasılığını elde etmek için, toplam olasılık formülüne göre yaprak düğüm içi hata oranının ağırlıklı toplamı hesaplanır.

Aşırı uyuma karşı hiçbir savunma yoktur. Aşırı büyümüş ağaç er ya da geç budanacaktır. Ne zaman durulacağına karar vermenin birkaç yolu vardır:

- Tüm terminal düğümleri saf olana kadar devam edilir.
- Her bir terminal düğümündeki veri sayısı belirli bir eşikten, örneğin 5'ten, hatta 1'den büyük olmayana kadar devam edilir.
- Ağaç yeterince büyük olduğu sürece, ilk ağacın boyutu kritik değildir.

Buradaki anahtar, ilk ağacı yeniden budamadan önce yeterince büyük yapmaktır!

Sınıflandırma Ağaçları:



Öznitelik seçim ölçüsü:

Karar Ağacında en büyük zorluk, her seviyede kök düğüm için özniteliğin tanımlanmasıdır. Bu işlem öznitelik seçimi olarak bilinir. İki popüler öznitelik seçim ölçüsü bulunmaktadır:

- 1) Bilgi Kazancı
- 2) Gini İndeksi

1) Bilgi Kazancı

Eğitim örneklerini daha küçük alt kümelere bölmek için karar ağacında bir düğüm kullandığımızda entropi değişir. Bilgi kazancı, entropideki bu değişimin bir ölçüsüdür.

Tanım: Diyelim ki S bir örnekler kümesi, A bir nitelik, Sv, S'nin A = v ile alt kümesi ve Değerler (A), A'nın tüm olası değerlerinin kümesidir, o zaman

$$Gain(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} \cdot Entropy(S_v)$$

Örnek:

$X = \{a,a,a,b,b,b,b\}$ kümesi için

Toplam örnek: 8

b: 5 örnekleri

a: 3'ün örnekleri

$$\begin{aligned} Entropy H(X) &= - \left[\left(\frac{3}{8} \right) \log_2 \frac{3}{8} + \left(\frac{5}{8} \right) \log_2 \frac{5}{8} \right] \\ &= - [0.375 * (-1.415) + 0.625 * (-0.678)] \\ &= - (-0.53 - 0.424) \\ &= 0.954 \end{aligned}$$

$$\log_2(a) = b \text{ ise } 2^b = a \text{ dır.}$$

$$\log_{10}(2^b) = \log_{10}(a)$$

$$b = \frac{1}{0.3} \log_{10}(a)$$

Bilgi Kazanımını Kullanarak Karar Ağacı Oluşturma

Gereklilikler:

- Kök düğümle ilişkili tüm eğitim örnekleriyle başlanır
- Her bir düğümün hangi öznitelikle etikleneceğini seçmek için bilgi kazancı kullanılır.
- Not: Hiçbir kökten yaprağa yol, aynı ayırık özniteliği iki kez içermemelidir
- Her alt ağacı, ağaçta o yolda sınıflandırılacak eğitim örneklerinin alt kümesinde yinelemeli olarak oluşturulur.

Sınır vakaları:

- Tüm pozitif veya tüm negatif eğitim örnekleri kalırsa, o düğümü buna göre “evet” veya “hayır” olarak etiketlenir.
- Hiçbir öznitelik kalmazsa, o düğümde kalan eğitim örneklerinin çoğunluk oyu ile etiketlenir.
- Örnek kalmadıysa, ebeveynin eğitim örnekleri çoğunluk oyu ile etiketlenir.

Örnek:

Şimdi aşağıdaki veriler için Bilgi kazanımını kullanarak bir Karar Ağacı çizelim.

Eğitim seti: 3 özellik ve 2 sınıf

X	Y	Z	C
1	1	1	I
1	1	0	I
0	0	1	II
1	0	0	II

Burada 3 özellik ve 2 çıktı sınıf var. Bilgi kazanımını kullanarak bir karar ağacı oluşturmak. Her bir özellik alınır ve her bir özellik için bilgi kazancı hesaplanır.

ID3 Algoritması:

Sadece kategorik veri ile çalışan bir yöntemdir. Her iterasyonun ilk adımında veri örneklerine ait sınıf bilgilerini taşıyan vektörün entropisi belirlenir. Daha sonra özellik vektörlerinin sınıfa bağımlı entropileri hesaplanarak ilk adımda hesaplanan entropiden çıkartılır. Bu şekilde elde edilen değer ilgili özellik vektörüne ait kazanç değeridir. En büyük kazanca sahip özellik vektörü ağacın o iterasyonda belirlenen dallanmasını gerçekleştirir.

Örnek:

2 özellik vektörü (V1 ve V2) ile S sınıf vektörüne sahip 4 örnekli veri kümesi verilmistir. ID3 algoritması ile ilk dallanma hangi özellik üzerinde gerçekleşir ?

- $H(S) - H(V1, S)$
- $H(S) - H(V2, S)$

V1	V2	S
A	C	E
B	C	F
B	D	E
B	D	F

Sınıf Entropisi

$$H(S) = -\left(\frac{1}{2}\log_2 \frac{1}{2} + \frac{1}{2}\log_2 \frac{1}{2}\right) = 1$$

V1 Entropisi

$$\begin{aligned} H(V1) &= \frac{1}{4}H(A) + \frac{3}{4}H(B) \\ &= \frac{1}{4}0 - \frac{3}{4}\left(\frac{1}{3}\log_2 \frac{1}{3} + \frac{2}{3}\log_2 \frac{2}{3}\right) = 0 + \frac{3}{4}0,9183 = 0,6887 \end{aligned}$$

V2 Entropisi

$$H(V2) = \frac{1}{2}H(C) + \frac{1}{2}H(D) = \frac{1}{2} + \frac{1}{2} = 1$$

V1 seçilir...

C4.5 Algoritması:

ID3 algoritmasının nümerik özellik içeren veriye uygulanabilen seklidir. ID3'ten tek farkı nümerik özelliklerin kategorik hale getirilebilmesini sağlayan bir esikleme yöntemini içermesidir. Temel mantık nümerik özellik vektöründeki tüm değerler ikili olarak ele alınarak ortalamaları esik olarak denenir. Hangi esik değeriyle bilgi kazanımı en iyi ise o değer seçilir. Seçilen esik değere göre özellik vektörü kategorize edilir ve ID3 uygulanır.

Örnek:

Kredilendirmede “Mükemmel”, “İyi” ve “Kötü” değerleri alabilen bir özelliğimiz vardır. Toplam 14 gözlem var. Bunlardan 7'si Normal Sorumluluk sınıfına, 7'si ise Yüksek Sorumluluk Sınıfına aittir. Yani kendi başına eşit bir bölünmedir. En üst satırda özetlersek, kredi notu özelliği için Mükemmel değerine sahip 4 gözlem olduğunu görebiliriz. Ayrıca, “Mükemmel” Kredi Notu için hedef değişkenimin nasıl bölündüğünü de görülebiliyor. Kredi notu için “Mükemmel” değeri alan gözlemler için Normal Sorumluluk sınıfına ait 3 adet ve Yüksek Sorumluluk sınıfına ait sadece 1 adet gözlem bulunmaktadır. Benzer şekilde diğer Kredi Notu değerleri için de beklenmedik durum tablosundan bu değerler bulunabilir.

Credit Rating		Liability		
		Normal	High	Total
Excellent		3	1	4
Good		4	2	6
Poor		0	4	4
Total		7	7	14

Bu örnek için, beklenmedik durum tablosunu, hedef değişkeninin entropisini kendi başına hesaplamak için kullanılacak ve ardından özellik, kredi notu hakkında ek bilgiler verilen hedef değişkenin entropisi hesaplanacak. Bu, "Kredi Notu"nun hedef değişkenin "Yükümlülük" için ne kadar ek bilgi sağladığını hesaplanmasına olanak sağlayacak.

$$\begin{aligned} E(Liability) &= -\frac{7}{14}\log_2\left(\frac{7}{14}\right) - \frac{7}{14}\log_2\left(\frac{7}{14}\right) \\ &= -\frac{1}{2}\log_2\left(\frac{1}{2}\right) - \frac{1}{2}\log_2\left(\frac{1}{2}\right) \\ &= 1 \end{aligned}$$

Hedef değişkenin entropisi, “Normal” ve “Yüksek” sınıf etiketi arasındaki eşit bölünme nedeniyle maksimum düzensizlikte 1'dir. Bir sonraki adım, kredi puanı hakkında ek bilgi verilen hedef değişken Yükümlülük'ün entropisini hesaplamaktır. Bunun için her Kredi Puanı değeri için Sorumluluk entropisi hesaplanacak ve her bir değerde sonuçlanan gözlemlerin

oranının ağırlıklı ortalaması kullanılarak bunlar eklenecektir. Neden ağırlıklı ortalama kullanıldığı, bunu karar ağaçları bağlamında tartışıldığında daha da netleşecektir.

$$E(Liability \mid CR = Excellent) = -\frac{3}{4}\log_2\left(\frac{3}{4}\right) - \frac{1}{4}\log_2\left(\frac{1}{4}\right) \approx 0.811$$

$$E(Liability \mid CR = Good) = -\frac{4}{6}\log_2\left(\frac{4}{6}\right) - \frac{2}{6}\log_2\left(\frac{2}{6}\right) \approx 0.918$$

$$E(Liability \mid CR = Poor) = -0\log_2(0) - \frac{4}{4}\log_2\left(\frac{4}{4}\right) = 0$$

Weighted Average:

$$E(Liability \mid CR) = \frac{4}{14} \times 0.811 + \frac{6}{14} \times 0.918 + \frac{4}{14} \times 0 = 0.625$$

Credit Rating	Liability		
	Normal	High	Total
Excellent	3	1	4
Good	4	2	6
Poor	0	4	4
Total	7	7	14

Kredi Derecelendirme özelliği verilen hedef değişken için entropi elde edildi. Artık bu özelliğin ne kadar bilgilendirici olduğunu görmek için Kredi Notundan Yükümlülük Bilgi Kazancı hesaplanmalıdır.

Information Gain:

$$IG(Liability, CR) = E(Liability) - E(Liability \mid CR)$$

$$= 1 - 0.625 = 0.375$$

Kredi Notunu bilmek, hedef değişkenimiz olan Sorumluluk etrafındaki belirsizliği azaltmamıza yardımcı oldu! İyi bir özelliğin yapması gereken de bu değil mi? Hedef değişkenimiz hakkında bize bilgi verir misiniz? İşte tam olarak bu, karar ağaçlarının, her bir bölünmede hedef değişkeni tahmin etmeye yaklaşmak ve aynı zamanda ağacı bölmeyi ne zaman durduracağını belirlemek için düğümlerini hangi özelliğin üzerine böleceğini belirlemek için entropi ve bilgi kazancını nasıl ve neden kullandığıdır! (elbette maksimum derinlik gibi hiper parametrelere ek olarak).

Gini yöntemi

Makine öğrenmesi karar ağacı algoritmasında örnekleri bölmenin birçok yolu vardır, bunlardan biride eğitimde GINI yöntemidir.

Gini yöntemi formülü kullanır: $Gini = 1 - (x/n)^2 - (y/n)^2$

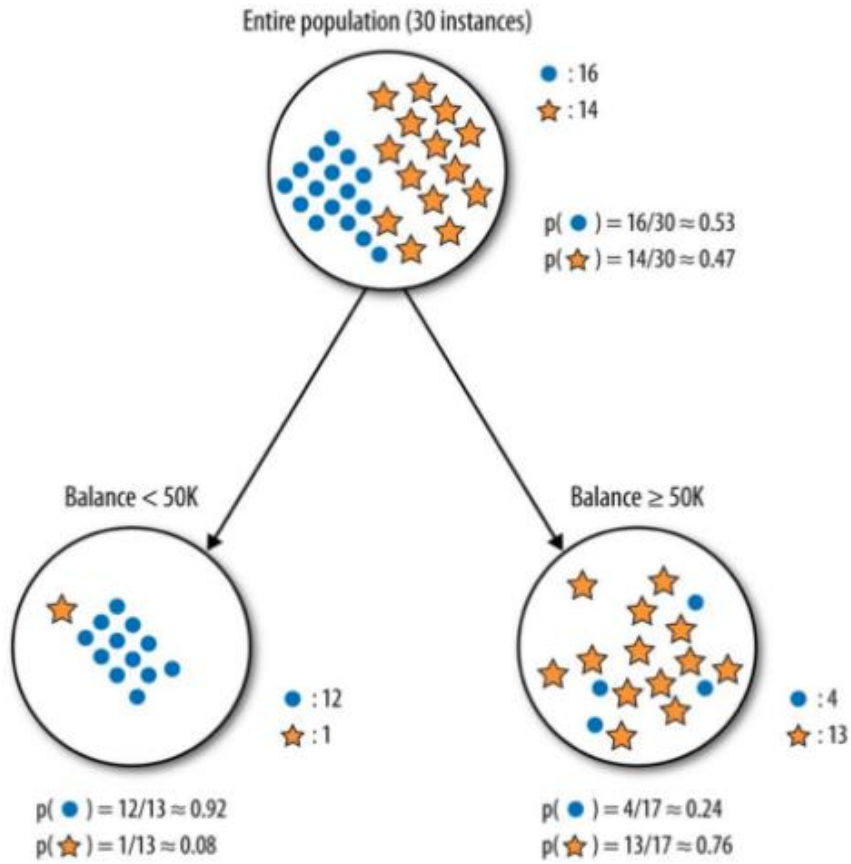
Burada n örnek sayısı, x olumlu sayısı, y olumsuzların sayısıdır. 40 oyunda 4'u kazanılmış, 36'u kaybedilmiş ise Gini indeksini hesaplayınız.

$$Gini = 1 - (4/40)^2 - (36/40)^2 = 1 - 1/100 - 81/100 = 18/100 = 0.18$$

Yorum: %18'nin bir yöne %82'nin ise diğer yöne gideceğini anlamına gelir.

Örnek: Karar Ağacı

Bir kişiye verilen bir kredinin bir zararla sonuçlanıp sonuçlanmayacağını tahmin etmek için bir karar ağacı oluşturduğumuz bir örneği ele alalım. Tüm popülasyonumuz 30 örnekten oluşmaktadır. 16'sı silinen sınıfa, diğer 14'ü silinmeyen sınıfa aittir. İki değer alabilen "Bakiye" -> "< 50K" veya ">50K" ve üç değer alabilen "Konut" -> "KENDİ", "KİRALIK" veya "DİĞER" olmak üzere iki özelliğimiz var. Entropi ve Bilgi Kazanımı kavramlarını kullanarak bir karar ağacı algoritmasının hangi özniteliğin ilk olarak bölüneceğine ve hangi özelliğin daha fazla bilgi sağladığına veya hedef değişkenimiz hakkındaki belirsizliği ikisinden daha fazla azalttığına nasıl karar vereceğini göstereceğim.



Özellik 1: Denge

Noktalar, sınıf hakkı olan veri noktalarıdır ve yıldızlar, silinmeyenlerdir. Ana düğümü öznelitik dengesine bölmek bize 2 alt düğüm verir. Sol düğüm, silme sınıfından 12/13 (0.92 olasılık) gözlem ile toplam gözlemlerin 13'ünü ve sınıfın yazılmayan sınıfından sadece 1/13 (0.08 olasılık) gözlemi alır. Sağ düğüm, silinmeyen sınıftan 13/17 (0.76 olasılık) ve silinen sınıftan 4/17 (0.24 olasılık) ile toplam gözlemin 17'sini alır.

Ana düğümün entropisini hesaplayalım ve Denge üzerinde bölerek ağacın ne kadar belirsizliği azaltabileceğini görelim.

$$E(Parent) = - \frac{16}{30} \log_2 \left(\frac{16}{30} \right) - \frac{14}{30} \log_2 \left(\frac{14}{30} \right) \approx 0.99$$

$$E(Balance < 50K) = - \frac{12}{13} \log_2 \left(\frac{12}{13} \right) - \frac{1}{13} \log_2 \left(\frac{1}{13} \right) \approx 0.39$$

$$E(Balance > 50K) = - \frac{4}{17} \log_2 \left(\frac{4}{17} \right) - \frac{13}{17} \log_2 \left(\frac{13}{17} \right) \approx 0.79$$

Weighted Average of entropy for each node:

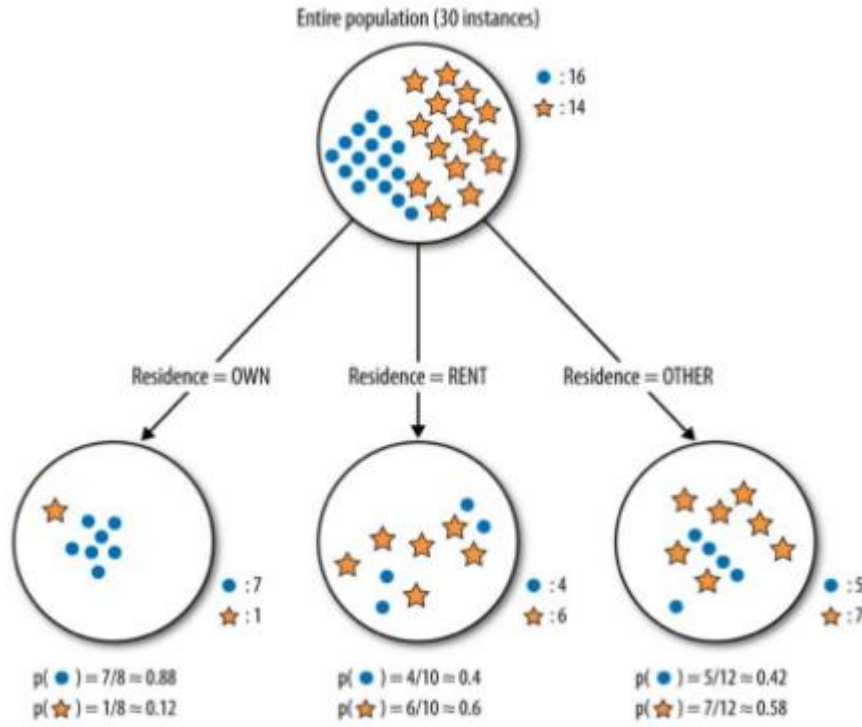
$$\begin{aligned} E(Balance) &= \frac{13}{30} \times 0.39 + \frac{17}{30} \times 0.79 \\ &= 0.62 \end{aligned}$$

Information Gain:

$$\begin{aligned} IG(Parent, Balance) &= E(Parent) - E(Balance) \\ &= 0.99 - 0.62 \\ &= 0.37 \end{aligned}$$

Özelliğe göre bölme, “Denge” hedef değişkenimiz üzerinde 0.37’lik bir bilgi kazanımına yol açar. Aynı şeyi nasıl karşılaştırdığını görmek için “Konut” özelliği için yapalım.

Özellik 2: Konut



Ağacı Residence'ta bölmek bize 3 alt düğüm verir. Sol alt düğüm, silinen sınıftan 7/8 (0.88 olasılık) gözlem ile toplam gözlemlerin 8'ini ve silinmeyen sınıftan sadece 1/8 (0.12 olasılık) gözlem alır. Orta alt düğümler, silinen sınıftan 4/10 (0,4 olasılık) ve silinmeyen sınıftan 6/10(0,6 olasılık) gözlem ile toplam gözlemlerin 10'unu alır. Sağ alt düğüm, silme sınıfından 5/12 (0.42 olasılık) gözlem ve silinmeyen sınıftan 7/12 (0.58) gözlem ile toplam gözlemlerin 12'sini alır. Ana düğümün entropisini zaten biliyoruz. “Konut”tan elde edilen bilgi kazancını hesaplamak için bölmeden sonraki entropiyi hesaplamamız yeterlidir.

$$E(\text{Residence} = \text{OWN}) = -\frac{7}{8}\log_2\left(\frac{7}{8}\right) - \frac{1}{8}\log_2\left(\frac{1}{8}\right) \approx 0.54$$

$$E(\text{Residence} = \text{RENT}) = -\frac{4}{10}\log_2\left(\frac{4}{10}\right) - \frac{6}{10}\log_2\left(\frac{6}{10}\right) \approx 0.97$$

$$E(\text{Residence} = \text{OTHER}) = -\frac{5}{12}\log_2\left(\frac{5}{12}\right) - \frac{7}{12}\log_2\left(\frac{7}{12}\right) \approx 0.98$$

Weighted Average of entropies for each node:

$$E(\text{Residence}) = \frac{8}{30} \times 0.54 + \frac{10}{30} \times 0.97 + \frac{12}{30} \times 0.98 = 0.86$$

Weighted Average of entropies for each node:

$$E(Residence) = \frac{8}{30} \times 0.54 + \frac{10}{30} \times 0.97 + \frac{12}{30} \times 0.98 = 0.86$$

Information Gain:

$$\begin{aligned} IG(Parent, Residence) &= E(Parent) - E(Residence) \\ &= 0.99 - 0.86 \\ &= 0.13 \end{aligned}$$

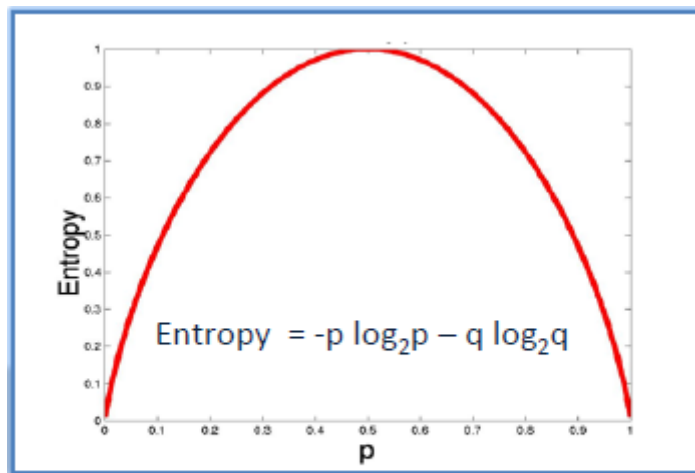
Denge özelliğinden gelen bilgi kazanımı, Konut'tan elde edilen bilgiden neredeyse 3 kat daha fazla! Geri dönüp grafiklere bir göz atarsanız, Denge'de bölünen alt düğümlerin, Yerleşim düğümlerinden daha saf göründüğünü görebilirsiniz. Bununla birlikte, ikamet için en soldaki düğüm de çok saftır, ancak ağırlıklı ortalamaların devreye girdiği yer burasıdır. Bu düğüm çok saf olmasına rağmen, toplam gözlemlerin en az miktarına sahiptir ve bir sonuç, Yığındaki bölmeden toplam entropiyi hesapladığımızda saflığının küçük bir kısmına katkıda bulunur. Bu önemlidir çünkü bir özelliğin genel bilgi gücünü arıyoruz ve sonuçlarımızın bir özellikteki nadir bir değer tarafından çarpıtılmasını istemiyoruz.

Kendi başına Balance özelliği, hedef değişkenimiz hakkında Konuttan daha fazla bilgi sağlar. Hedef değişkenimizde daha fazla düzensizliği azaltır. Bir karar ağacı algoritması, Balance kullanarak verilerimizde ilk bölmeyi yapmak için bu sonucu kullanır. Bundan sonra, karar ağacı algoritması, bir sonraki hangi özelliği böleceğine karar vermek için her bölmede bu süreci kullanacaktır. Gerçek dünya senaryosunda, ikiden fazla özellik ile ilk bölme en bilgilendirici özellik üzerinde yapılır ve daha sonra her bölmede, her bir ek özellik için bilgi kazancının yeniden hesaplanması gerekir, çünkü her birinden elde edilen bilgi kazancı ile aynı olmaz. özellik kendi başına. Entropi ve bilgi kazancı, sonuçları değiştirecek bir veya daha fazla bölme yapıldıktan sonra hesaplanmalıdır. Bir karar ağacı, önceden tanımlanmış bir derinliğe ulaşana ya da hiçbir ek bölünme, genellikle hiper parametre olarak da belirlenebilen belirli bir eşiğin ötesinde daha yüksek bir bilgi kazanımına neden olana kadar derinleştikçe ve derinleştikçe bu işlemi tekrarlar!

Example: Decision Tree - Classification



Bir karar ağacı, bir kök düğümden yukarıdan aşağıya oluşturulur ve verileri benzer değerlere sahip (homojen) örnekler içeren alt kümelere ayırmayı içerir. ID3 algoritması, bir örneğin homojenliğini hesaplamak için entropiyi kullanır. Örnek tamamen homojen (Olma olasılığı 1 ise olma olasılığı sıfırdır. Tüm olasıklar toplamı bire eşit olmak zorundadır) ise entropi sıfırdır ve örnek eşit olarak bölünmüşse entropisi birdir.



$$Entropy = -0.5 \log_2 0.5 - 0.5 \log_2 0.5 = 1$$

$$y = \log_2(x) = \log_{10}(x) / 0.3$$

$$\log_{10}(2) = 0.3, \log_{10}(3) = 0.477, \log_{10}(5) = 0.7, \log_{10}(7) = 0.845$$

Bir karar ağacı oluşturmak için aşağıdaki gibi sıklık tablolarını kullanarak iki tür entropi hesaplamamız gerekir:

a) Bir özelliğin sıklık tablosunu kullanan entropi:

$$E(S) = \sum_{i=1}^c -p_i \log_2 p_i$$

Play Golf	
Yes	No
9	5



$$\begin{aligned} \text{Entropy(PlayGolf)} &= \text{Entropy}(5,9) \\ &= \text{Entropy}(0.36, 0.64) \\ &= -(0.36 \log_2 0.36) - (0.64 \log_2 0.64) \\ &= 0.94 \end{aligned}$$

$$5/14=0.36$$

$$9/14=0.64$$

b) İki özelliğin sıklık tablosunu kullanan entropi:

$$E(T, X) = \sum_{c \in X} P(c)E(c)$$

		Play Golf		
		Yes	No	
Outlook	Sunny	3	2	5
	Overcast	4	0	4
	Rainy	2	3	5
				14



$$\begin{aligned} E(\text{PlayGolf}, \text{Outlook}) &= P(\text{Sunny}) \cdot E(3,2) + P(\text{Overcast}) \cdot E(4,0) + P(\text{Rainy}) \cdot E(2,3) \\ &= (5/14) \cdot 0.971 + (4/14) \cdot 0.0 + (5/14) \cdot 0.971 \\ &= 0.693 \end{aligned}$$

Information Gain

Bilgi kazancı, bir veri kümesi bir özniteliğe bölündükten sonra entropideki azalmaya dayanır. Bir karar ağacı oluşturmak, en yüksek bilgi kazancını (yani en homojen dalları) döndüren özniteliği bulmakla ilgilidir.

Adım 1: Hedefin entropisini hesaplanır.

$$\begin{aligned}
 \text{Entropy}(\text{PlayGolf}) &= \text{Entropy}(5,9) \\
 &= \text{Entropy}(0.36, 0.64) \\
 &= -(0.36 \log_2 0.36) - (0.64 \log_2 0.64) \\
 &= 0.94
 \end{aligned}$$

Adım 2: Veri kümesi daha sonra farklı niteliklere bölünür. Her dal için entropi hesaplanır. Daha sonra, bölme için toplam entropi elde etmek için orantılı olarak eklenir. Ortaya çıkan entropi, bölünmeden önceki entropiden çıkarılır. Sonuç, Bilgi Kazanımı veya entropideki azalmadır.

		Play Golf	
		Yes	No
Outlook	Sunny	3	2
	Overcast	4	0
	Rainy	2	3
		Gain = 0.247	

		Play Golf	
		Yes	No
Temp.	Hot	2	2
	Mild	4	2
	Cool	3	1
		Gain = 0.029	

		Play Golf	
		Yes	No
Humidity	High	3	4
	Normal	6	1
		Gain = 0.152	

		Play Golf	
		Yes	No
Windy	False	6	2
	True	3	3
		Gain = 0.048	

$$Gain(T, X) = Entropy(T) - Entropy(T, X)$$

$$\begin{aligned}
 G(\text{PlayGolf}, \text{Outlook}) &= E(\text{PlayGolf}) - E(\text{PlayGolf}, \text{Outlook}) \\
 &= 0.940 - 0.693 = 0.247
 \end{aligned}$$

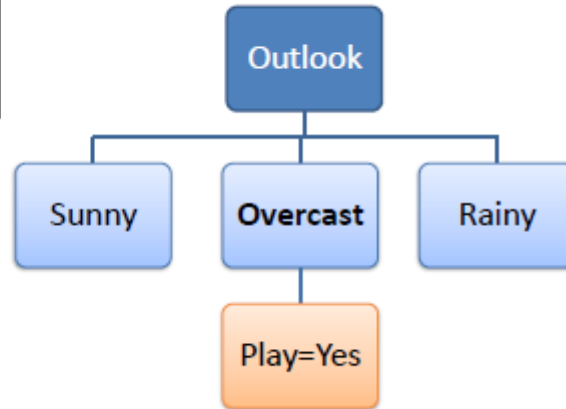
Adım 3: Karar düğümü olarak en büyük bilgi kazancına sahip öznelik seçilir, veri seti dallarına bölünür ve aynı işlemi her dalda tekrarlanır.

		Play Golf	
		Yes	No
★ Outlook	Sunny	3	2
	Overcast	4	0
	Rainy	2	3
		Gain = 0.247	

Outlook	Sunny	Outlook	Temp	Humidity	Windy	Play Golf
		Sunny	Mild	High	FALSE	Yes
		Sunny	Cool	Normal	FALSE	Yes
		Sunny	Cool	Normal	TRUE	No
		Sunny	Mild	Normal	FALSE	Yes
	Sunny	Mild	High	TRUE	No	
	Overcast	Overcast	Hot	High	FALSE	Yes
		Overcast	Cool	Normal	TRUE	Yes
		Overcast	Mild	High	TRUE	Yes
		Overcast	Hot	Normal	FALSE	Yes
	Rainy	Rainy	Hot	High	FALSE	No
		Rainy	Hot	High	TRUE	No
		Rainy	Mild	High	FALSE	No
		Rainy	Cool	Normal	FALSE	Yes
		Rainy	Mild	Normal	TRUE	Yes

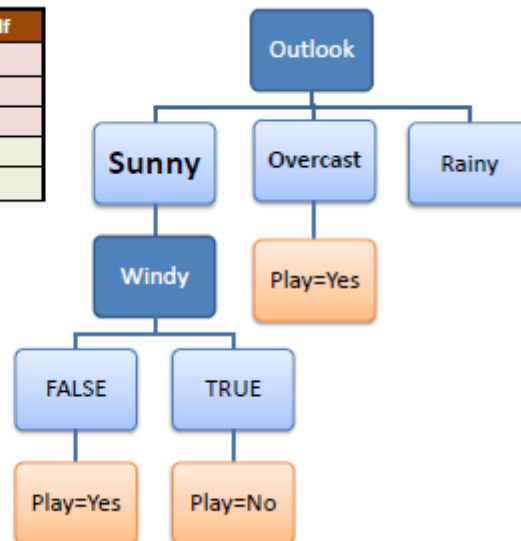
Step 4a: Entropisi 0 olan bir dal, bir yaprak düğümdür.

Temp	Humidity	Windy	Play Golf
Hot	High	FALSE	Yes
Cool	Normal	TRUE	Yes
Mild	High	TRUE	Yes
Hot	Normal	FALSE	Yes



Adım 4b: Entropisi 0'dan büyük olan bir dalın daha fazla bölünmesi gerekir.

Temp	Humidity	Windy	Play Golf
Mild	High	FALSE	Yes
Cool	Normal	FALSE	Yes
Mild	Normal	FALSE	Yes
Cool	Normal	TRUE	No
Mild	High	TRUE	No



Adım 5: ID3 algoritması, tüm veriler sınıflandırılana kadar yaprak olmayan dallarda özyinelemeli olarak çalıştırılır.

Karar Ağacından Karar Kurallarına

Bir karar ağacı, kök düğümden yaprak düğümlere tek tek eşlenerek kolayca bir dizi kurala dönüştürülebilir.

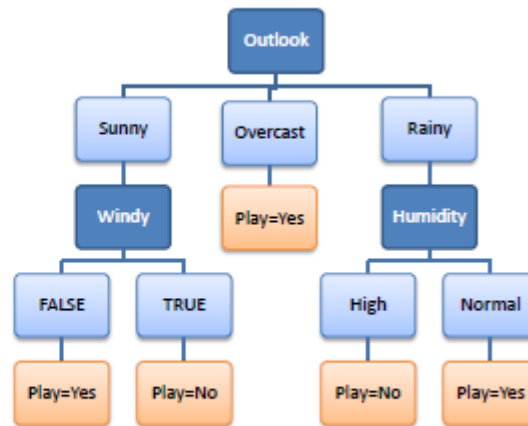
R_1 : IF (Outlook=Sunny) AND (Windy=FALSE) THEN Play=Yes

R_2 : IF (Outlook=Sunny) AND (Windy=TRUE) THEN Play=No

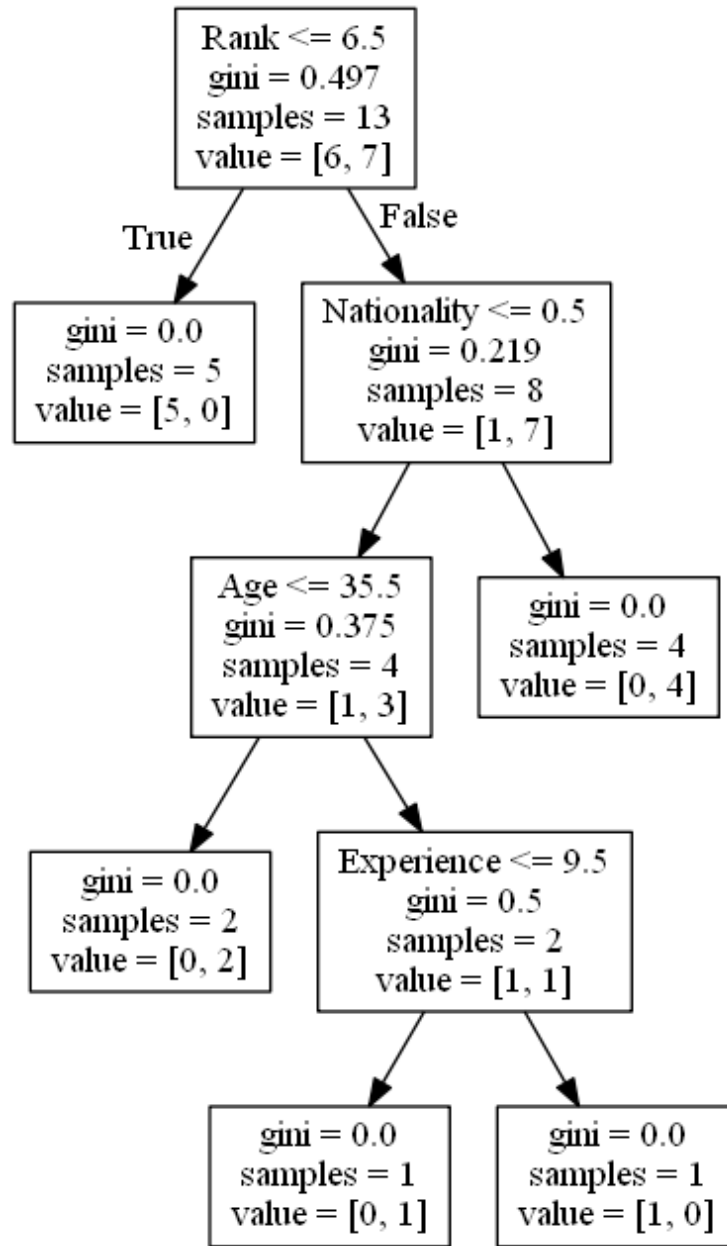
R_3 : IF (Outlook=Overcast) THEN Play=Yes

R_4 : IF (Outlook=Rainy) AND (Humidity=High) THEN Play=No

R_5 : IF (Outlook=Rain) AND (Humidity=Normal) THEN Play=Yes



Uygulama:



Karar Ağacı bir Akış Şemasıdır ve önceki deneyimlere dayalı kararlar vermenize yardımcı olabilir.

Örnekte, bir kişi bir komedi şovuna gidip gitmeyeceğine karar vermeye çalışacaktır. Neyse ki örnek kişimiz kasabada her komedi şovu olduğunda kayıt olmuş ve komedyen hakkında bazı bilgileri kaydetmiş, gidip gitmediğini de kaydetmiştir.

Age	Experience	Rank	Nationality	Go
36	10	9	UK	NO
42	12	4	USA	NO
23	4	6	N	NO
52	4	4	USA	NO
43	21	8	USA	YES
44	14	5	UK	NO
66	3	7	N	YES
35	14	9	UK	YES
52	13	7	N	YES
35	5	9	N	YES
24	3	5	USA	NO
18	3	7	UK	YES
45	9	9	UK	YES

Şimdi, bu veri setine dayanarak Python, herhangi bir yeni şovun katılmaya değer olup olmadığına karar vermek için kullanılabilecek bir karar ağacı oluşturabilir.

Nasıl çalışır?

İlk önce ihtiyacınız olan modülleri içe aktarılır ve veri seti pandalarla okunur.

Örnek: Veri setini okunur ve yazdırılır.

```
import pandas
from sklearn import tree
import pydotplus
from sklearn.tree import DecisionTreeClassifier
import matplotlib.pyplot as plt
import matplotlib.image as pltimg
df = pandas.read_csv("shows.csv")
print(df)
```

Bir karar ağacı oluşturmak için tüm verilerin sayısal olması gerekir.

Sayısal olmayan 'Milliyet' ve 'Git' sütunlarını sayısal değerlere dönüştürülmesi gerekiyor. Pandalar, değerlerin nasıl dönüştürüleceği hakkında bilgi içeren bir sözlük alan bir map() yöntemine sahiptir.

```
{'UK': 0, 'USA': 1, 'N': 2}
```

Değerleri dönüştürmek anlamına gelir: 'UK' to 0, 'USA' to 1, and 'N' to 2.

Örnek: Karekter değerlerinin sayısal değerlere değiştirilmesi.

```
d = {'UK': 0, 'USA': 1, 'N': 2}
df['Nationality'] = df['Nationality'].map(d)
d = {'YES': 1, 'NO': 0}
df['Go'] = df['Go'].map(d)
print(df)
```

Ardından, özellik sütunlarını hedef sütundan ayırmamız gerekir.

Özellik sütunları, tahmin etmeye çalıştığımız sütunlardır ve hedef sütun, tahmin etmeye çalıştığımız değerlerin bulunduğu sütundur.

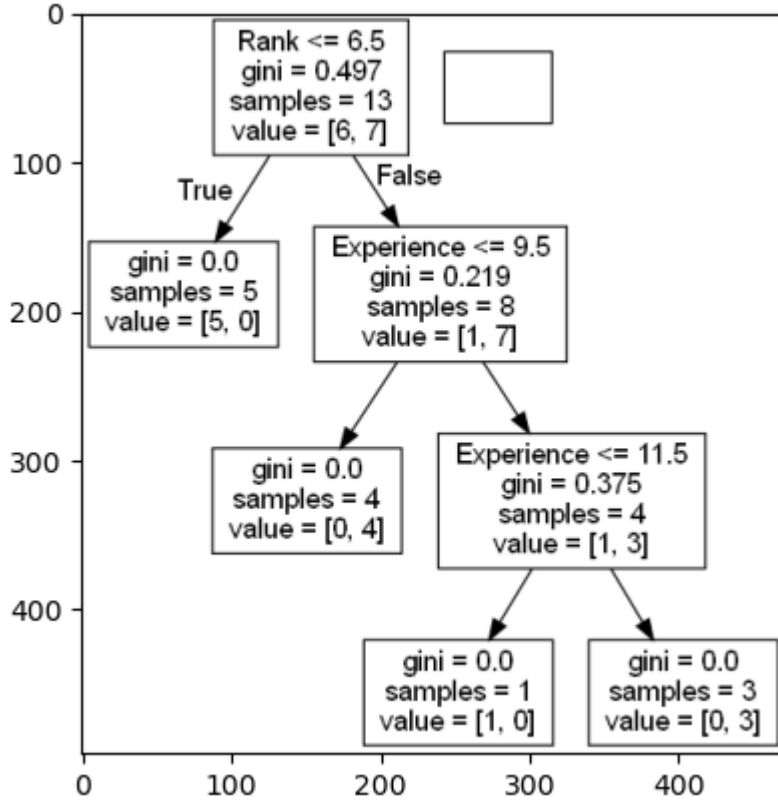
Örnek: X, özellik sütunlarıdır, y hedef sütundur.

```
features = ['Age', 'Experience', 'Rank', 'Nationality']
X = df[features]
y = df['Go']
print(X)
print(y)
```

Artık gerçek karar ağacını oluşturabilir, onu ayrıntılarımıza sığdırabilir ve bilgisayara bir .png dosyası kaydedebiliriz.

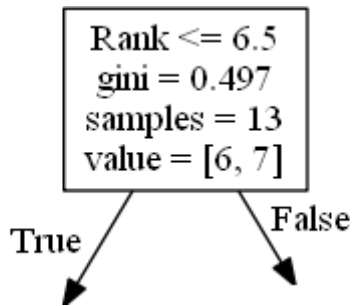
Örnek: Bir Karar Ağacı oluşturulması, bir resim olarak kaydedilmesi ve resmin gösterilmesi.

```
dtree = DecisionTreeClassifier()
dtree = dtree.fit(X, y)
data = tree.export_graphviz(dtree, out_file=None, feature_names=features)
graph = pydotplus.graph_from_dot_data(data)
graph.write_png('mydecisiontree.png')
img=pltimg.imread('mydecisiontree.png')
imgplot = plt.imshow(img)
plt.show()
```



Sonuç Açıklaması:

Karar ağacı, bir komedyeni görmek isteyip istememe ihtimalinizi hesaplamak için önceki kararlarınızı kullanır. Karar ağacının farklı yönlerini okuyalım:



Rank:

Rank <= 6.5, rankı 6.5 veya daha düşük olan her komedyenin True okunu (solda) izleyeceği ve geri kalanının False oku (sağda) izleyeceği anlamına gelir.

gini = 0.497, bölmenin kalitesini ifade eder ve her zaman 0.0 ile 0.5 arasında bir sayıdır; burada 0.0, tüm örneklerin aynı sonucu aldığı anlamına gelir ve 0.5, bölmenin tam olarak ortada yapıldığı anlamına gelir.

Samples(numuneler) = 13, kararda bu noktada 13 komedyen kaldığı anlamına gelir, bu ilk adım olduğu için hepsi bu kadar.

Value(değer) = [6, 7] bu 13 komedyenden 6'sının "HAYIR" ve 7'sinin "DEVAM" alacağı anlamına gelir.

Gini:

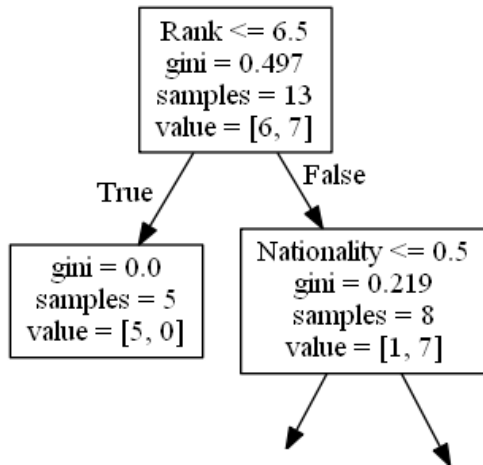
Örnekleri bölmenin birçok yolu vardır, bu eğitimde GINI yöntemini kullanıyoruz.

Gini yöntemi şu formülü kullanır:

$$\text{Gini} = 1 - (x/n)^2 - (y/n)^2$$

Burada x, olumlu yanıtların ("GO") sayısıdır, n örnek sayısıdır ve y, bize bu hesaplamayı veren olumsuz yanıtların ("HAYIR") sayısıdır.

$$1 - (7 / 13)^2 - (6 / 13)^2 = 0.497$$



Bir sonraki adım iki kutu içerir, bir kutu 'Rank' 6.5 veya daha düşük olan komedyenler için ve bir kutu geri kalanıyla birlikte.

True - 5 durumunda Komedyen Burada Bitiyor.

gini = 0.0, tüm numunelerin aynı sonucu aldığı anlamına gelir.

numuneler = 5, bu branşta 5 komedyen kaldığı anlamına gelir (5 komedyen, Rütbesi 6.5 veya daha düşük).

değer = [5, 0], 5'in "HAYIR" ve 0'ın "DEVAM" alacağı anlamına gelir.

Yanlış - 8 durumunda Komedyen Devam Ediyor:

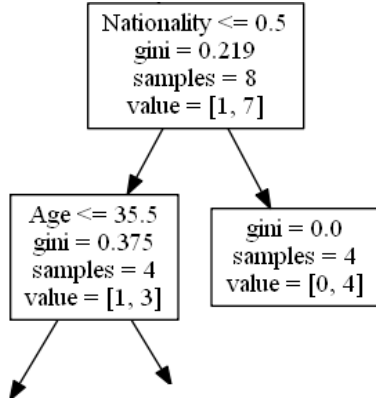
Uyruk:

Uyruk <= 0,5, uyruk değeri 0,5'ten az olan komedyenlerin soldaki oku (yani Birleşik Krallık'tan herkes,) izleyeceği ve geri kalanların sağdaki oku izleyeceği anlamına gelir.

gini = 0.219, numunelerin yaklaşık %22'sinin bir yöne gideceği anlamına gelir.

numuneler = 8, bu branşta 8 komedyen kaldığı anlamına gelir (8 komedyen, Rütbesi 6.5'ten yüksek).

değer = [1, 7], bu 8 komedyenden 1'inin "HAYIR" ve 7'sinin "DEVAM" alacağı anlamına gelir.



True - 4 durumunda Komedyen Devam.

Yaş:

Yaş <= 35.5, 35,5 veya daha küçük yaştaki komedyenlerin soldaki oku, geri kalanların da sağdaki oku izleyeceği anlamına gelir.

gini = 0.375, numunelerin yaklaşık %37,5'inin bir yöne gideceği anlamına gelir.

numuneler = 4, bu dalda 4 komedyen kaldığı anlamına gelir (İngiltere'den 4 komedyen).

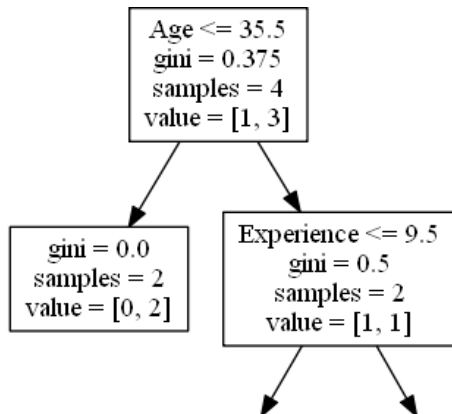
değer = [1, 3], bu 4 komedyenden 1'inin "HAYIR" ve 3'ünün "DEVAM" alacağı anlamına gelir.

False - 4 Komedyen durumu Burada Bitiriyor.

gini = 0.0, tüm numunelerin aynı sonucu aldığı anlamına gelir.

numuneler = 4, bu branşta 4 komedyen kaldığı anlamına gelir (İngiltere'den olmayan 4 komedyen).

değer = [0, 4], bu 4 komedyenden 0'ın "HAYIR" ve 4'ünün "DEVAM" alacağı anlamına gelir.



True - 2 durumunda Komedyen Burada Bitiyor.

gini = 0.0, tüm numunelerin aynı sonucu aldığı anlamına gelir.

numuneler = 2, bu branşta 2 komedyen kaldığı anlamına gelir (2 komedyen 35.5 yaşında veya daha genç).

değer = [0, 2], bu 2 komedyenden 0'ın "HAYIR" alacağı ve 2'nin "DEVAM" alacağı anlamına gelir.

Yanlış - 2 durumunda Komedyen Devam:

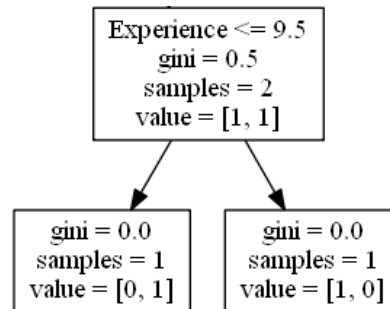
Deneyim:

Deneyim ≤ 9.5 , 9,5 yıl veya daha az deneyime sahip komedyenlerin soldaki oku, geri kalanların ise sağdaki oku izleyeceği anlamına gelir.

gini = 0,5, numunelerin %50'sinin bir yöne gideceği anlamına gelir.

numuneler = 2, bu branşta 2 komedyen kaldığı anlamına gelir (2 35,5 yaşından büyük komedyen).

değer = [1, 1], bu 2 komedyenden 1'inin "HAYIR" ve 1'inin "DEVAM" alacağı anlamına gelir.



True - 1 durumunda Komedyen Burada Bitiyor.

gini = 0.0, tüm numunelerin aynı sonucu aldığı anlamına gelir.

numuneler = 1, bu branşta 1 komedyen kaldığı anlamına gelir (1 komedyen 9,5 yıl veya daha az deneyime sahip).

değer = [0, 1], 0'ın "HAYIR" alacağı ve 1'in "DEVAM" alacağı anlamına gelir.

Yanlış - 1 Komedyen Burada Bitiyor:

gini = 0.0, tüm numunelerin aynı sonucu aldığı anlamına gelir.

numuneler = 1, bu branşta 1 komedyen kaldığı anlamına gelir (1 komedyen 9,5 yıldan fazla deneyime sahip).

değer = [1, 0], 1'in "HAYIR" alacağı ve 0'ın "DEVAM" alacağı anlamına gelir.

Tahmin Değerleri:

Yeni değerleri tahmin etmek için Karar Ağacını kullanabiliriz.

Örnek: 40 yaşında, 10 yıllık deneyime sahip ve komedi sıralaması 7 olan bir Amerikalı komedyenin oynadığı bir gösteriye gitmeli miyim?

Örnek: Yeni değerleri tahmin etmek için tahmin() yöntemini kullanın.

```
print(dtree.predict([[40, 10, 7, 1]]))
```

Örnek: Komedi sıralaması 6 olsaydı cevap ne olurdu?

```
print(dtree.predict([[40, 10, 6, 1]]))
```

Farklı Sonuçlar

Karar Ağacı'nı aynı verilerle besleseniz bile yeterince çalıştırırsanız size farklı sonuçlar verdiği göreceksiniz.

Bunun nedeni, Karar Ağacı'nın bize %100 kesin bir cevap vermemesidir. Bir sonucun olasılığına dayanır ve cevap değişecektir.

5.1.2. Önyükleme Toplama (Torbalama - Bagging)

Karar Ağaçları gibi yöntemler, yeni veriler üzerinde yanlış tahminlere yol açabilecek şekilde, eğitim setine fazla uymaya meyilli olabilir.

Bootstrap Aggregation (torbalama), sınıflandırma veya regresyon sorunları için fazla uydurmayı çözmeye çalışan bir birleştirme yöntemidir. Torbalama, makine öğrenimi algoritmalarının doğruluğunu ve performansını iyileştirmeyi amaçlar. Bunu, orijinal bir veri kümesinin rastgele alt kümelerini değiştirme ile alarak yapar ve her alt küme için sınıflandırıcı (sınıflandırma için) veya geriletilici (gerileme için) uyar. Her alt küme için tahminler daha sonra sınıflandırma için çoğunluk oyu veya regresyon için ortalama alma yoluyla toplanır, bu da tahmin doğruluğunu artırır.

Bir Temel Sınıflandırıcıyı Değerlendirmek:

Torbalamanın model performansını nasıl iyileştirebileceğini görmek için, temel sınıflandırıcının veri kümesinde nasıl performans gösterdiğini değerlendirerek başlamalıyız. Karar ağaçlarının ne olduğunu bilmiyorsanız, ilerlemeden önce karar ağaçları dersini gözden geçirin, çünkü torbalama kavramın bir devamıdır.

Sklearn'in şarap veri setinde bulunan farklı şarap sınıflarını tanımlamaya çalışacağız.

Gerekli modülleri içe aktararak başlayalım.

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier
```

Daha sonra verileri yüklememiz ve X (giriş özellikleri) ve y (hedef) içinde saklamamız gerekiyor. `as_frame` parametresi `True` olarak ayarlanır, böylece verileri yüklerken özellik adlarını kaybetmeyiz. (0.23'ten eski sklearn sürümü, desteklenmediği için `as_frame` argümanını atlamalıdır)

```
data = datasets.load_wine(as_frame = True)
```

```
X = data.data
y = data.target
```

Modelimizi görünmeyen veriler üzerinde doğru bir şekilde değerlendirmek için X ve y'yi tren ve test setlerine ayırmamız gerekiyor. Verileri bölme hakkında bilgi için, Eğitim/Test dersine bakın.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 22)
```

Hazırlanan verilerimizle artık bir temel sınıflandırıcıyı somutlaştırabilir ve onu eğitim verilerine sığdırabiliriz.

```
dtree = DecisionTreeClassifier(random_state = 22)
dtree.fit(X_train,y_train)
DecisionTreeClassifier(random_state=22)
```

Artık görünmeyen test setinin şarap sınıfını tahmin edebilir ve model performansını değerlendirebiliriz.

```
y_pred = dtree.predict(X_test)
```

```
print("Train data accuracy:",accuracy_score(y_true = y_train, y_pred =
dtree.predict(X_train)))
print("Test data accuracy:",accuracy_score(y_true = y_test, y_pred = y_pred))
```

Sonuç:

Train data accuracy: 1.0

Test data accuracy: 0.8222222222222222

Kod Örneği:

Gerekli veriler içe aktarılır ve temel sınıflandırıcı performansı değerlendirilir.

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier

data = datasets.load_wine(as_frame = True)
X = data.data
y = data.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 22)

dtree = DecisionTreeClassifier(random_state = 22)
dtree.fit(X_train,y_train)
y_pred = dtree.predict(X_test)

print("Train data accuracy:",accuracy_score(y_true = y_train, y_pred =
dtree.predict(X_train)))
print("Test data accuracy:",accuracy_score(y_true = y_test, y_pred = y_pred))
```

Temel sınıflandırıcı, geçerli parametrelerle test veri kümesinde %91 doğruluk elde ederek veri kümesinde oldukça iyi performans gösterir (random_state parametre kümeniz yoksa farklı sonuçlar oluşabilir).

Artık test veri kümesi için temel bir doğruluğa sahip olduğumuza göre, Torbalama Sınıflandırıcısının tek bir Karar Ağacı Sınıflandırıcısını nasıl gerçekleştirdiğini görebiliriz.

Torbalama Sınıflandırıcısı Oluşturma

Torbalama için `n_estimators` parametresini ayarlamamız gerekiyor, bu, modelimizin bir araya toplayacağı temel sınıflandırıcıların sayısıdır.

Bu örnek veri kümesi için tahmin edicilerin sayısı nispeten düşüktür, genellikle çok daha büyük aralıkların keşfedildiği durumdur. Hiperparametre ayarı genellikle bir grid araması ile yapılır, ancak şimdilik tahminci sayısı için seçilmiş bir dizi değer kullanacağız.

Gerekli modeli import ederek başlıyoruz.

```
from sklearn.ensemble import BaggingClassifier
```

Şimdi, her toplulukta kullanmak istediğimiz tahminci sayısını temsil eden bir değerler aralığı oluşturalım.

```
estimator_range = [2,4,6,8,10,12,14,16]
```

Torbalama Sınıflandırıcısının farklı `n_estimators` değerleriyle nasıl performans gösterdiğini görmek için, değer aralığı üzerinde yineleme yapmanın ve her topluluktan elde edilen sonuçları saklamanın bir yoluna ihtiyacımız var. Bunu yapmak için, daha sonraki görselleştirmeler için modelleri ve puanları ayrı listelerde saklayan bir for döngüsü oluşturacağız.

Not: `BaggingClassifier`'daki temel sınıflandırıcı için varsayılan parametre

`DecisionTreeClassifier`'dır, bu nedenle torbalama modelini başlatırken bunu ayarlamamız gerekmez.

```
models = []
```

```
scores = []
```

```
for n_estimators in estimator_range:
```

```
    # Create bagging classifier
```

```
    clf = BaggingClassifier(n_estimators = n_estimators, random_state = 22)
```

```
    # Fit the model
```

```
    clf.fit(X_train, y_train)
```

```
    # Append the model and score to their respective list
```

```
    models.append(clf)
```

```
    scores.append(accuracy_score(y_true = y_test, y_pred = clf.predict(X_test)))
```

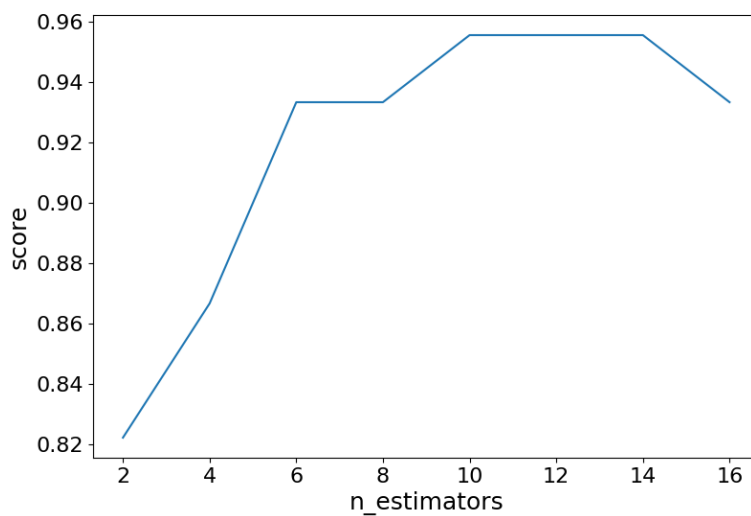
Depolanan modeller ve puanlarla artık model performansındaki gelişmeyi görselleştirebiliriz.

```
import matplotlib.pyplot as plt
```

```
# Generate the plot of scores against number of estimators
plt.figure(figsize=(9,6))
plt.plot(estimator_range, scores)
```

```
# Adjust labels and font (to make visible)
plt.xlabel("n_estimators", fontsize = 18)
plt.ylabel("score", fontsize = 18)
plt.tick_params(labelsize = 16)
```

```
# Visualize plot
```



Kod örneği:

Gerekli verileri içe aktarın ve BaggingClassifier performansını değerlendirin.

```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.ensemble import BaggingClassifier
```

```
data = datasets.load_wine(as_frame = True)
```

```
X = data.data
y = data.target
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 22)
```

```
estimator_range = [2,4,6,8,10,12,14,16]
```

```
models = []
```

```
scores = []
```

```
for n_estimators in estimator_range:
```

```
    # Create bagging classifier
```

```
    clf = BaggingClassifier(n_estimators = n_estimators, random_state = 22)
```

```
    # Fit the model
```

```
    clf.fit(X_train, y_train)
```

```
    # Append the model and score to their respective list
```

```
    models.append(clf)
```

```
    scores.append(accuracy_score(y_true = y_test, y_pred = clf.predict(X_test)))
```

```
# Generate the plot of scores against number of estimators
```

```
plt.figure(figsize=(9,6))
```

```
plt.plot(estimator_range, scores)
```

```
# Adjust labels and font (to make visible)
```

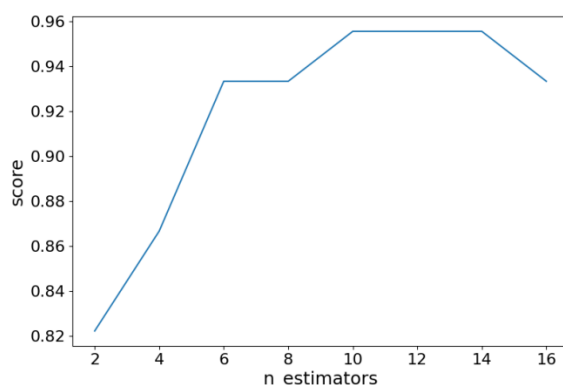
```
plt.xlabel("n_estimators", fontsize = 18)
```

```
plt.ylabel("score", fontsize = 18)
```

```
plt.tick_params(labelsize = 16)
```

```
# Visualize plot
```

```
plt.show()
```



Sonuçların Açıklanması:

Tahminci sayısı için farklı değerleri yineleyerek model performansında %82,2'den %95,5'e bir artış görebiliriz. 14 tahminden sonra doğruluk düşmeye başlar, yine farklı bir

rastgele_durum (random_state) ayarlarsanız, gördüğünüz değerler değişecektir. Bu nedenle, istikrarlı sonuçlar elde etmek için çapraz doğrulama kullanmak en iyi uygulamadır. Bu durumda, şarabın türünü belirlemeye gelince doğrulukta %13,3'lük bir artış görüyoruz.

Başka Bir Değerlendirme Şekli:

Önyükleme, sınıflandırıcılar oluşturmak için rastgele gözlem alt kümelerini seçtiğinden, seçim sürecinde dışarıda bırakılan gözlemler vardır. Bu "torbadan çıkmış" gözlemler, daha sonra, bir test setine benzer şekilde modeli değerlendirmek için kullanılabilir. Torba dışı tahminin ikili sınıflandırma problemlerindeki hatayı olduğundan fazla tahmin edebileceğini ve yalnızca diğer ölçümlere tamamlayıcı olarak kullanılması gerektiğini unutmayın.

Son alıştırmada 12 tahmincinin en yüksek doğruluğu verdiğini gördük, bu yüzden modelimizi oluşturmak için bunu kullanacağız. Bu sefer, çanta dışı puanı olan modeli değerlendirmek için oob_score parametresini true olarak ayarlıyoruz.

Kod örneği:

Torba dışı metriğe sahip bir model oluşturulur.

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.ensemble import BaggingClassifier
```

```
data = datasets.load_wine(as_frame = True)
```

```
X = data.data
```

```
y = data.target
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 22)
```

```
oob_model = BaggingClassifier(n_estimators = 12, oob_score = True, random_state = 22)
```

```
oob_model.fit(X_train, y_train)
```

```
print(oob_model.oob_score_)
```

OOB'de kullanılan örnekler ve test seti farklı olduğu ve veri seti nispeten küçük olduğu için doğrulukta bir fark vardır. Tamamen aynı olmaları nadirdir, yine OOB, hatayı tahmin etmek için hızlı araçlar kullanılmalıdır, ancak tek değerlendirme metriği değildir.

Out-of-bag error (OOB):

Torbadan çıkma tahmini olarak da adlandırılan torba dışı hata, rastgele ormanların, artırılmış karar ağaçlarının ve önyükleme toplamayı kullanan diğer makine öğrenimi modellerinin

tahmin hatasını ölçme yöntemidir. Torbalama, modelin öğreneceği eğitim örnekleri oluşturmak için alt örnekleme yöntemi kullanır.

Torbalama Sınıflandırıcısından Karar Ağaçları Oluşturma:

Karar Ağacı dersinde görüldüğü gibi modelin oluşturduğu karar ağacının grafiğini çıkarmak mümkündür. Toplu sınıflandırıcıya giren bireysel karar ağaçlarını da görmek mümkündür. Bu, torbalama modelinin tahminlerine nasıl ulaştığı konusunda daha sezgisel bir anlayış kazanmamıza yardımcı olur.

Not: Bu, yalnızca ağaçların nispeten sık ve dar olduğu ve görselleştirilmesini kolaylaştıran daha küçük veri kümelerinde işlevseldir.

Plot_tree işlevini sklearn.tree'den içe aktarmamız gerekecek. Farklı ağaçlar, görselleştirmek istediğiniz tahmin ediciyi değiştirerek grafik haline getirilebilir.

Kod örneği:

Torbalama Sınıflandırıcısından Karar Ağaçları Oluşturulur.

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import plot_tree

X = data.data
y = data.target

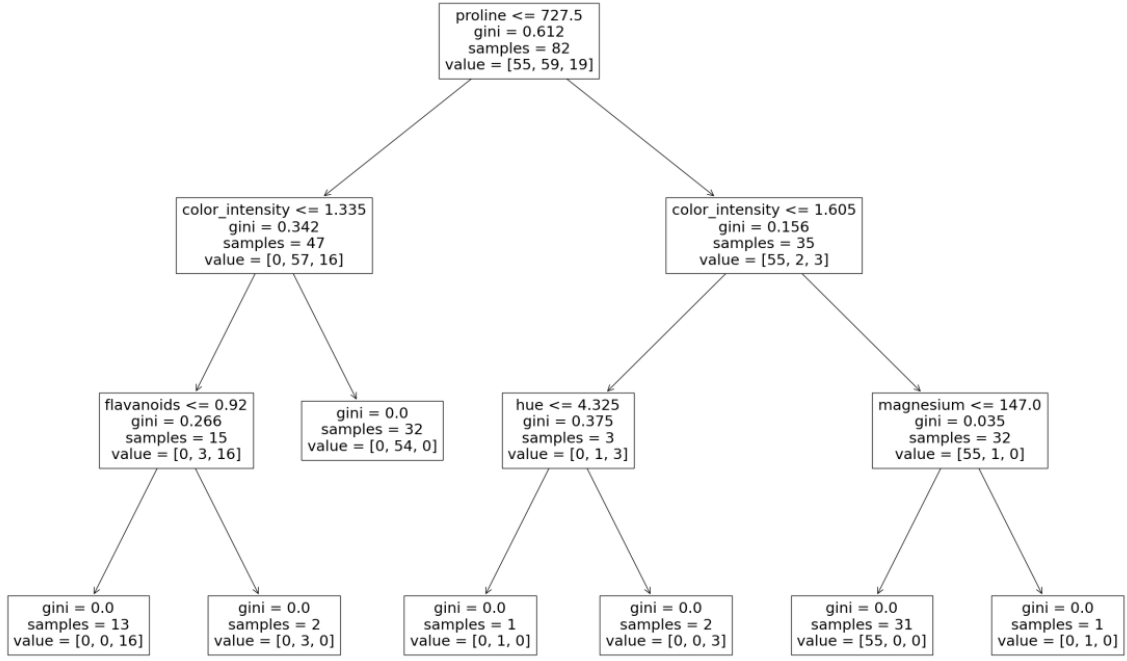
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 22)

clf = BaggingClassifier(n_estimators = 12, oob_score = True, random_state = 22)

clf.fit(X_train, y_train)

plt.figure(figsize=(30, 20))

plot_tree(clf.estimators_[0], feature_names = X.columns)
```



Burada sadece nihai tahmine oy vermek için kullanılan ilk karar ağacını görebiliriz. Yine, sınıflandırıcının indeksini değiştirerek toplanan ağaçların her birini görebilirsiniz.

5.1.3. K-NN En Yakın Komşu

1967 yılında T. M. Cover ve P. E. Hart tarafından önerilen, örnek veri noktasının bulunduğu sınıfın ve en yakın komşunun, K değerine göre belirlendiği bir sınıflandırma yöntemidir. Denetimli öğrenmede sınıflandırma ve regresyon için kullanılan algoritmalarından biridir. En basit makine öğrenmesi algoritması olarak kabul edilir.

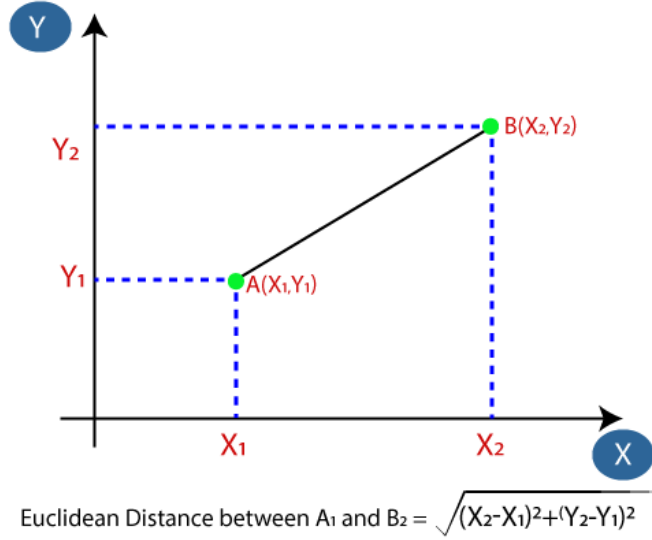
KNN amacı, yeni bir örnek geldiğinde var olan öğrenme verisi üzerinde sınıflandırma yaparak onun en yakın K komşusuna bakarak örneğin sınıfına karar verir.

K-NN algoritması sınıflandırma algoritmasıdır. Sınıflandırmak, belirli bir veri kümesini farklı sınıflara ayırma işlemidir. Sınıflandırma hem yapılandırılmış hem de yapılandırılmamış veri türleri üzerinde uygulanabilir.

KNN algoritması sınıflandırılmak istenen bir veriyi daha önceki verilerle olan yakınlık ilişkisine göre sınıflandıran bir algoritmadır. Algoritma adının içinde bulunduğu “K” algoritmaya dahil edilecek veri kümesindeki veri sayısını ifade etmektedir. Yani algoritmada “k” adet komşu aranır. Bir tahmin yapmak istediğimizde, tüm veri setinde en yakın komşuları arar. Algoritmanın çalışmasında bir K değeri belirlenir. Bu K değerinin anlamı bakılacak eleman sayısıdır. Bir değer geldiğinde en yakın K kadar eleman alınarak gelen değer arasındaki uzaklık hesaplanır. İlgili uzaklılardan en yakın k komşu ele alınır. Öznitelik değerlerine göre k komşu veya komşuların sınıfına atanır. Seçilen sınıf, tahmin edilmesi beklenen gözlem değerinin sınıfı olarak kabul edilir. Yani yeni veri etiketlenmiş (label) olur.

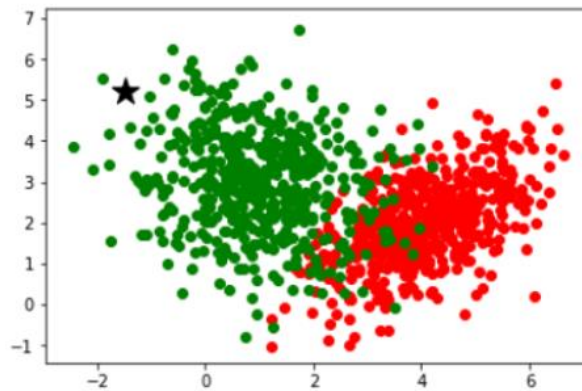
K-NN non-parametric (parametrik olmayan), lazy (tembel) bir öğrenme algoritmasıdır. lazy kavramını anlamaya çalışırsak “eager learning” aksine “lazy learning”’in bir eğitim aşaması yoktur. Eğitim verilerini öğrenmez, bunun yerine eğitim veri kümesini “ezberler”. Uzaklık hesaplama işleminde genelde Öklid fonksiyonu kullanılır.

Öklid fonksiyonuna alternatif olarak Manhattan, Minkowski ve Hamming fonksiyonları da kullanılabilir. Uzaklık hesaplandıktan sonra sıralanır ve gelen değer uygun olan sınıfa atanır.



Bir tahmin yapmak için KNN algoritması, lojistik veya doğrusal regresyonda olduğu gibi bir eğitim veri kümesinden öngörücü bir model hesaplamaz. Aslında, KNN'nin tahmine dayalı bir model oluşturmaya gerek yoktur. Bu nedenle, KNN için gerçek bir öğrenme aşaması yoktur. Bu yüzden genellikle tembel bir öğrenme yöntemi olarak kategorize edilir. Bir tahmin yapabilmek için, KNN herhangi bir eğitim aşaması olmadan bir sonuç üretmek için veri setini kullanır.

KNN, bir tahmin yapmak için tüm veri kümesini depolar. KNN herhangi bir tahmine dayalı modeli hesaplamaz ve tembel öğrenme algoritmaları ailesinin bir parçasıdır. KNN, bir girdi gözlemi ile veri kümesindeki farklı gözlemler arasındaki benzerliği hesaplayarak tam zamanında (anında) tahminler yapar.



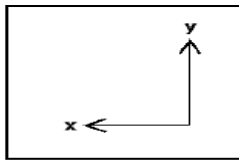
Yukarıdaki şekilde, kırmızı veya yeşil olarak sınıflandırılan veri noktalarında siyah bir veri noktası kırmızı ya da yeşil olabilecek iki sınıfı göstermektedir. Siyah noktanın ne olduğu KNN algoritması tarafından nasıl hesaplanır?

KNN algoritmaları, sınıflandırılacak veri noktasına en yakın komşu olan bir k sayısına karar verir. K değeri 5 ise, o veri noktasına en yakın 5 Komşuyu arayacaktır. Bu örnekte, k = 4. KNN en yakın 4 komşuyu bulur. Bu veri noktasının bu komşulara yakın olması nedeniyle sadece bu sınıfa ait olacağı görülmektedir.

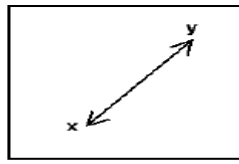
K-en yakın komşu sınıflandırıcı algoritmalarının basit versiyonu, en yakın komşu sınıfı bularak hedef etiketini tahmin etmektir. Sınıflandırılacak noktaya en yakın sınıf, Öklid mesafesi kullanılarak hesaplanır.

Mesafe, benzerliği ölçmek için kullanılır. İki örnek arasındaki mesafeyi ölçmenin birçok yolu vardır.

- Manhattan Distance, $|X1-X2| + |Y1-Y2|$
- Euclidean Distance, $\sqrt{((x1-x2)^2)+(y1-y2)^2}$



Manhattan



Euclidean

Mesafe Ölçüm yöntemleri:

Minkowsky: $D(x, y) = \left(\sum_{i=1}^m x_i - y_i ^r \right)^{1/r}$	Euclidean: $D(x, y) = \sqrt{\sum_{i=1}^m (x_i - y_i)^2}$	Manhattan / city-block: $D(x, y) = \sum_{i=1}^m x_i - y_i $
Camberra: $D(x, y) = \sum_{i=1}^m \frac{ x_i - y_i }{ x_i + y_i }$	Chebychev: $D(x, y) = \max_{i=1}^m x_i - y_i $	
Quadratic: $D(x, y) = (x - y)^T Q (x - y) = \sum_{j=1}^m \left(\sum_{i=1}^m (x_i - y_i) q_{ji} \right) (x_j - y_j)$ Q is a problem-specific positive definite $m \times m$ weight matrix		
Mahalanobis: $D(x, y) = [\det V]^{1/m} (x - y)^T V^{-1} (x - y)$	V is the covariance matrix of $A_1..A_m$, and A_j is the vector of values for attribute j occurring in the training set instances $1..n$.	
Correlation: $D(x, y) = \frac{\sum_{i=1}^m (x_i - \bar{x}_i)(y_i - \bar{y}_i)}{\sqrt{\sum_{i=1}^m (x_i - \bar{x}_i)^2 \sum_{i=1}^m (y_i - \bar{y}_i)^2}}$	$\bar{x}_i = \bar{y}_i$ and is the average value for attribute i occurring in the training set.	
Chi-square: $D(x, y) = \sum_{i=1}^m \frac{1}{sum_i} \left(\frac{x_i}{size_x} - \frac{y_i}{size_y} \right)^2$	sum_i is the sum of all values for attribute i occurring in the training set, and $size_x$ is the sum of all values in the vector x.	
Kendall's Rank Correlation: sign(x)=-1, 0 or 1 if $x < 0$, $x = 0$, or $x > 0$, respectively.	$D(x, y) = 1 - \frac{2}{n(n-1)} \sum_{i=1}^m \sum_{j=1}^{i-1} \text{sign}(x_i - x_j) \text{sign}(y_i - y_j)$	

Figure 1. Equations of selected distance functions.
(x and y are vectors of m attribute values).

Mesafe ölçüsünün de yalnızca sürekli değişkenler için geçerli olduğu unutulmamalıdır. Kategorik değişkenler durumunda Hamming mesafesi kullanılmalıdır. Veri setinde sayısal ve kategorik değişkenlerin bir karışımı olduğunda, 0 ile 1 arasındaki sayısal değişkenlerin standardizasyonu konusunu da gündeme getirir.

Hamming Distance

$$D_H = \sum_{i=1}^k |x_i - y_i|$$

$$x = y \Rightarrow D = 0$$

$$x \neq y \Rightarrow D = 1$$

X	Y	Distance
Male	Male	0
Male	Female	1

K'nin önemi nedir?

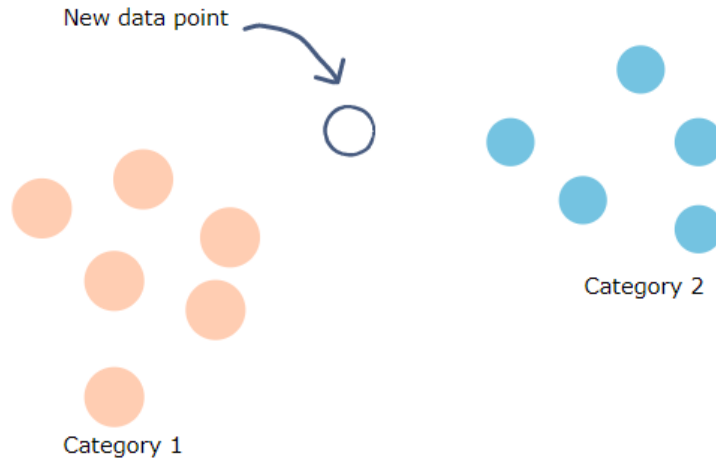
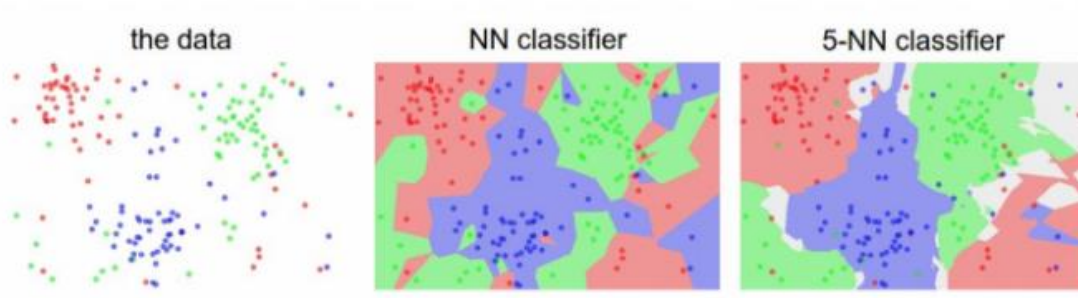
K değeri büyüdükçe tahmine duyulan güveni artırır. Öte yandan K çok büyük bir değere sahipse, kararlar çarpık olabilir.

K nasıl seçilir?

Algoritma, yeni bir veri noktasının diğer tüm eğitim veri noktalarına olan mesafesini hesaplar. Mesafe herhangi bir türde olabilir, örneğin Öklid, Manhattan, vb. Algoritma daha sonra k'ye en yakın veri noktalarını seçer, burada k herhangi bir tam sayı olabilir. Sayısal değerlerin hangi özelliği temsil ettiğine bakılmaksızın, seçimini diğer veri noktalarına yakınlığına göre yapar. Son olarak, veri noktasını benzer veri noktalarının bulunduğu sınıfa atar.

Seçilen veri kümesine uyan K değerini seçmek için, KNN algoritması farklı K değerleri ile defalarca çalıştırılır. Sonra, algoritma, yeni değerler için hassas tahminler yapma yeteneğini korurken karşılaşılan hata sayısını azaltan K'yi seçilir.

- K'ye karar vermek, K-en yakın Komşular'ın en kritik kısmıdır.
- K değeri küçükse, gürültü sonuca daha fazla bağımlı olacaktır. Bu gibi durumlarda modelin aşırı uyumu çok fazladır.
- K'nin değeri ne kadar büyükse, KNN'nin arkasındaki prensibi yok edecektir.
- Çapraz doğrulamayı kullanarak K'nin optimum değeri bulunabilir.

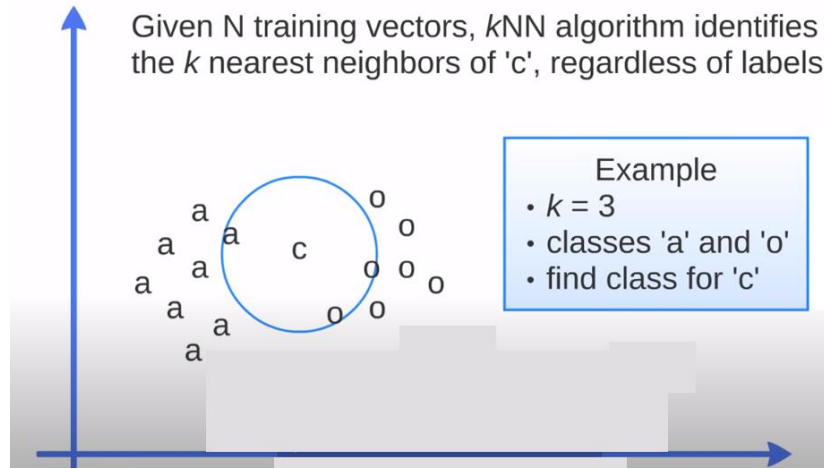


KNN algoritması sözde kod uygulaması

- İstenilen verileri yüklenir.
- k parametresi belirlenir. Bu parametre verilen bir noktaya en yakın komşuların sayısıdır. Örneğin: $k=2$ olsun. Bu durumda en yakın 2 komşuya göre sınıflandırma yapılacaktır.
- Örnek veri setine katılacak olan yeni verinin, mevcut verilere göre uzaklığı tek tek hesaplanır. İlgili uzaklık fonksiyonları yardımıyla.
- İlgili uzaklıklardan en yakın k komşu ele alınır. Öznitelik değerlerine göre k komşu veya komşuların sınıfına atanır.
- Seçilen sınıf, tahmin edilmesi beklenen gözlem değerinin sınıfı olarak kabul edilir. Yani yeni veri etiketlenmiş (label) olur.

KNN, anlaşılması oldukça basit bir algoritmadır. Bunun başlıca nedeni, tahmin yapabilmek için bir modele ihtiyaç duymamasıdır. Bunun tersi, tahminini yapabilmek için tüm gözlemlerini hafızasında tutması gerektiğidir. Bu nedenle, girdi veri kümesinin boyutuna dikkat etmeniz gerekir. Ayrıca, mesafenin hesaplanması için yöntemin seçimi ve komşuların K sayısı hemen belli olmayabilir. Kullanım durumunuz için tatmin edici bir sonuç elde etmek için birkaç kombinasyon denemeniz ve algoritmayı ayarlamanız gerekebilir.

Örnek:



Öğrencinin Adı	Girdi Özellikleri		Çıktı Özneliği
	Sayfa Sayısı	Soru Sayısı	Başarı Durumu
Selin	2100	1500	BAŞARILI
Mert	2280	1600	BAŞARILI
Caner	800	400	BAŞARISIZ
Şenay	2400	1750	BAŞARILI
Efe	250	350	BAŞARISIZ
Burak	180	220	BAŞARISIZ
Esra	1800	1200	?

Öklid uzaklığı kullanılarak altı adet öğrencinin Esra adlı öğrenciye olan uzaklıkları:

Öğrencinin Adı	Sayfa Sayısı	Soru Sayısı	Başarı Durumu	Uzaklık Hesaplaması	Sonuç
Selin	2100	1500	BAŞARILI	$\sqrt{(2100 - 1800)^2 + (1500 - 1200)^2}$	424.2
Mert	2280	1600	BAŞARILI	$\sqrt{(2280 - 1800)^2 + (1600 - 1200)^2}$	624.8
Caner	800	400	BAŞARISIZ	$\sqrt{(800 - 1800)^2 + (400 - 1200)^2}$	1280
Şenay	2400	1750	BAŞARILI	$\sqrt{(2400 - 1800)^2 + (1750 - 1200)^2}$	813.9
Efe	250	350	BAŞARISIZ	$\sqrt{(250 - 1800)^2 + (350 - 1200)^2}$	1767
Burak	180	220	BAŞARISIZ	$\sqrt{(180 - 1800)^2 + (220 - 1200)^2}$	1893
Esra	1800	1200	?		

Örneğin $K=3$ seçersek ilk 3 öğrencinin başarı durumuna bakacağız.

Öğrencinin Adı	Başarı Durumu	Sonuç	Sıra No
Selin	BAŞARILI	424.2	1
Mert	BAŞARILI	624.8	2
Şenay	BAŞARILI	813.9	3
Caner	BAŞARISIZ	1280	4
Efe	BAŞARISIZ	1767	5
Burak	BAŞARISIZ	1893	6
Esra	?		

Sonuç BAŞARILI

Örneğin $K=6$ seçersek ilk 6 öğrencinin başarı durumuna bakacağız.

Öğrencinin Adı	Başarı Durumu	Sonuç	Sıra No
Selin	BAŞARILI	424.2	1
Mert	BAŞARILI	624.8	2
Şenay	BAŞARILI	813.9	3
Caner	BAŞARISIZ	1280	4
Efe	BAŞARISIZ	1767	5
Burak	BAŞARISIZ	1893	6
Esra	?		

Sonuç ?

Bu yüzden K parametresinin **çift** olarak seçilmesi tercih edilmemektedir.

Örnek:

KNN, eğitim seti yardımıyla veri noktasını belirli bir kategoriye sınıflandırmaya çalıştığımız parametrik olmayan denetimli bir öğrenme tekniğidir. Basit bir deyişle, tüm eğitim durumlarının bilgilerini yakalar ve yeni durumları benzerliğe göre sınıflandırır.

Yeni bir örnek (x) için, en benzer K durum (komşular) için tüm eğitim seti aranarak ve bu K durumlar için çıktı değişkeni özetlenerek tahminler yapılır. Sınıflandırmada bu, mod (veya en yaygın) sınıf değeridir.

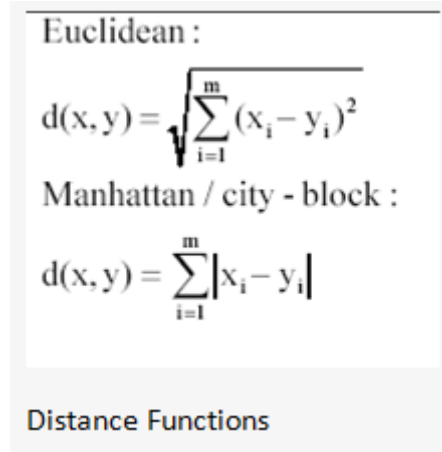
KNN algoritması nasıl çalışır?

Diyelim ki bazı müşterilerin boy, kilo ve tişört bedenleri var ve yeni bir müşterinin Tişört bedenini sadece sahip olduğumuz boy ve kilo bilgisine göre tahmin etmemiz gerekiyor. Boy, kilo ve tişört beden bilgilerini içeren veriler aşağıda gösterilmiştir.

Height (in cms)	Weight (in kgs)	T Shirt Size
158	58	M
158	59	M
158	63	M
160	59	M
160	60	M
163	60	M
163	61	M
160	64	L
163	64	L
165	61	L
165	62	L
165	65	L
168	62	L
168	63	L
168	66	L
170	63	L
170	64	L
170	68	L

Adım 1: Uzaklık fonksiyonuna göre Benzerliği hesaplayın

Pek çok uzaklık fonksiyonu vardır ama en yaygın olarak kullanılan ölçü Ökliddir. Esas olarak veriler sürekli olduğunda kullanılır. Manhattan mesafesi de sürekli değişkenler için çok yaygındır.



Euclidean :

$$d(x, y) = \sqrt{\sum_{i=1}^m (x_i - y_i)^2}$$

Manhattan / city - block :

$$d(x, y) = \sum_{i=1}^m |x_i - y_i|$$

Distance Functions

Mesafe ölçümü kullanma fikri, yeni örnek ve eğitim durumları arasındaki mesafeyi (benzerliği) bulmak ve ardından boy ve ağırlık açısından yeni müşteriye en yakın k-müşterileri bulmaktır.

'Monica' adlı yeni müşteri 161cm boyunda ve 61kg ağırlığındadır. İlk gözlem ile yeni gözlem (monica) arasındaki Öklid uzaklığı aşağıdaki gibidir:

$$= \text{SQRT}((161-158)^2 + (61-58)^2)$$

Benzer şekilde, yeni vaka ile tüm eğitim vakalarının mesafesini hesaplayacağız ve mesafe açısından sıralamayı hesaplayacağız. En küçük mesafe değeri 1 olarak sıralanır ve en yakın komşu olarak kabul edilir.

Step 2 : Find K-Nearest Neighbors

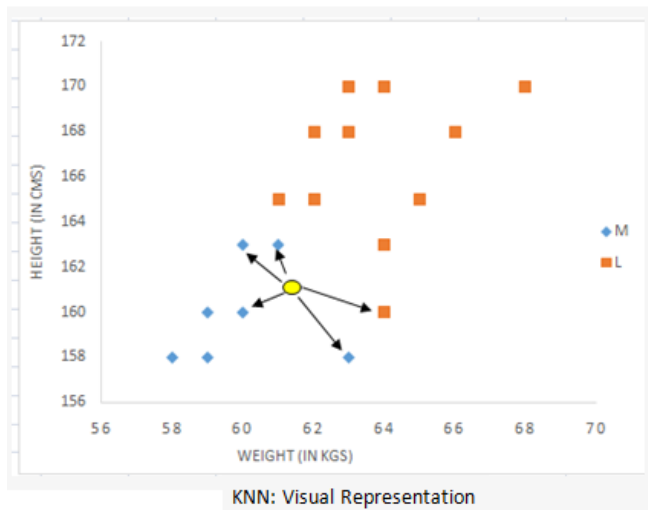
Let k be 5. Then the algorithm searches for the 5 customers closest to Monica, i.e. most similar to Monica in terms of attributes, and see what categories those 5 customers were in. If 4 of them had 'Medium T shirt sizes' and 1 had 'Large T shirt size' then your best guess for Monica is 'Medium T shirt. See the calculation shown in the snapshot below –

2. Adım : K-En Yakın Komşuları Bulunması

K=5 olsun. Ardından algoritma, özellikler açısından Monica'ya en yakın, yani Monica'ya en çok benzeyen 5 müşteriye bakar ve bu 5 müşterinin hangi kategorilerde olduğunu görür. 4 tanesi 'Orta T shirt bedenleri' ve 1 tanesi ise 'Büyük T gömlek bedeni' vardı, o zaman Monica için en iyi tahmininiz 'Orta T gömlek. Aşağıdaki anlık görüntüde gösterilen hesaplama bakın

	=SQRT((A21-A6)^2+(B21-B6)^2)				
	A	B	C	D	E
	Height (in cms)	Weight (in kgs)	T Shirt Size	Distance	
1					
2	158	58	M	4.2	
3	158	59	M	3.6	
4	158	63	M	3.6	
5	160	59	M	2.2	3
6	160	60	M	1.4	1
7	163	60	M	2.2	3
8	163	61	M	2.0	2
9	160	64	L	3.2	5
10	163	64	L	3.6	
11	165	61	L	4.0	
12	165	62	L	4.1	
13	165	65	L	5.7	
14	168	62	L	7.1	
15	168	63	L	7.3	
16	168	66	L	8.6	
17	170	63	L	9.2	
18	170	64	L	9.5	
19	170	68	L	11.4	
20					
21	161	61			

Aşağıdaki grafikte ikili bağımlı değişken (T-shirt bedeni) mavi ve turuncu renkte görüntülenmektedir. 'Orta boy tişört' mavi renkte ve 'Büyük boy tişört' turuncu renktedir. Yeni müşteri bilgileri sarı daire içinde sergilenir. Dört mavi vurgulanmış veri noktası ve bir turuncu vurgulanmış veri noktası sarı daireye yakındır. bu nedenle yeni vaka için tahmin, Orta T-shirt boyutu olan mavi vurgulu veri noktasıdır.



KNN Varsayımları:

1. Standardizasyon

Eğitim verilerindeki bağımsız değişkenler farklı birimlerde ölçüldüğünde, mesafeyi hesaplamadan önce değişkenleri standardize etmek önemlidir. Örneğin, bir değişken cm cinsinden yüksekliği temel alıyorsa ve diğeri kg cinsinden ağırlığı temel alıyorsa, uzunluk mesafe hesaplamasını daha fazla etkileyecektir. Bunları karşılaştırılabilir hale getirmek için, aşağıdaki yöntemlerden herhangi biriyle yapılabilecekleri standartlaştırmamız gerekir:

$$X_s = \frac{X - \text{mean}}{s.d.}$$
$$X_s = \frac{X - \text{mean}}{\text{max} - \text{min}}$$
$$X_s = \frac{X - \text{min}}{\text{max} - \text{min}}$$

Standardization

Standardizasyondan önce yükseklik hakim olduğundan, standardizasyondan sonra 5. en yakın değer değişmiştir. Bu nedenle, K-en yakın komşu algoritmasını çalıştırmadan önce tahmin edicileri standart hale getirmek önemlidir.

	A	B	C	D	E
1	Height (in cms)	Weight (in kgs)	T Shirt Size	Distance	
2	-1.39	-1.64	M	1.3	
3	-1.39	-1.27	M	1.0	
4	-1.39	0.25	M	1.0	
5	-0.92	-1.27	M	0.8	4
6	-0.92	-0.89	M	0.4	1
7	-0.23	-0.89	M	0.6	3
8	-0.23	-0.51	M	0.5	2
9	-0.92	0.63	L	1.2	
10	-0.23	0.63	L	1.2	
11	0.23	-0.51	L	0.9	5
12	0.23	-0.13	L	1.0	
13	0.23	1.01	L	1.8	
14	0.92	-0.13	L	1.7	
15	0.92	0.25	L	1.8	
16	0.92	1.39	L	2.5	
17	1.39	0.25	L	2.2	
18	1.39	0.63	L	2.4	
19	1.39	2.15	L	3.4	
20					
21	-0.7	-0.5			

Knn after standardization

2. Aykırı Değer

Düşük k-değeri aykırı değerlere duyarlıdır ve daha yüksek bir K-değeri, daha fazla seçmenin tahmine karar vereceğini düşündüğü için aykırı değerlere karşı daha dirençlidir.

KNN neden parametrik değildir?

Parametrik olmayan, temel alınan veri dağılımı üzerinde herhangi bir varsayımda bulunmamak anlamına gelir. Parametrik olmayan yöntemler, modelde sabit sayıda parametreye sahip değildir. Benzer şekilde KNN'de model parametreleri aslında eğitim veri seti ile birlikte büyür - her eğitim durumunu modelde bir "parametre" olarak düşünebilirsiniz.

KNN ve K-ortalama: Birçok insan bu iki istatistiksel teknik - K-ortalama ve K-en yakın komşu arasında kafa karıştırır. Aşağıdaki farklardan bazılarına bakın:

- K-ortalama denetimsiz bir öğrenme tekniğidir (bağımlı değişken yoktur), KNN denetimli bir öğrenme algoritmasıdır (bağımlı değişken vardır)
- K-ortalama, veri noktalarını her kümedeki noktalar birbirine yakın olacak şekilde K-kümelerine ayırmaya çalışan bir kümeleme tekniğidir, K-en yakın komşu ise bir noktanın sınıflandırmasını belirlemeye çalışır, K sınıflandırmasını en yakın noktalara birleştirmeye çalışır.

KNN regresyon için kullanılabilir mi?

Evet, regresyon için K-en yakın komşu kullanılabilir. Başka bir deyişle, bağımlı değişken sürekli olduğunda K-en yakın komşu algoritması uygulanabilir. Bu durumda, tahmin edilen değer, en yakın k komşusunun değerlerinin ortalamasıdır.

KNN'nin Artıları ve Eksileri

Artıları:

- Anlaması kolay
- Veriler hakkında varsayım yok
- Hem sınıflandırma hem de regresyona uygulanabilir
- Çok sınıflı problemlerde kolayca çalışır

Eksileri:

- Yoğun Bellek / Hesaplama açısından pahalı
- Veri ölçeğine duyarlı
- Nadir olay (çarpık) hedef değişkeni üzerinde iyi çalışmıyor
- Çok sayıda bağımsız değişken olduğunda mücadele
- Herhangi bir problem için, küçük bir k değeri, tahminlerde büyük bir varyansa yol açacaktır. Alternatif olarak, k değerini büyük bir değere ayarlamak, büyük bir model yanlılığına yol açabilir.

KNN'de kategorik deęişkenler nasıl ele alınır?

Kategorik bir deęişkendenden kukla deęişkenler oluřturun ve orijinal kategorik deęişken yerine bunları dahil edin. Regresyondan farklı olarak, $(k-1)$ yerine k kukla oluřturun. Örneęin, "Departman" adlı kategorik bir deęişkenin 5 benzersiz düzeyi/kategorisi vardır. Böylece 5 kukla deęişken oluřturacaęız. Her kukla deęişkenin departmanına karşı 1 ve 0'a sahiptir.

En iyi K deęeri nasıl bulunur?

Çapraz doęrulama, optimal K deęerini bulmanın akıllı bir yoludur. Model oluřturma sürecinden eğitim setinin bir alt kümesini dıřarıda tutarak doęrulama hata oranını tahmin eder.

Çapraz doęrulama (diyelim ki 10 kat doęrulama), eğitim setinin rastgele olarak yaklaşık eşit büyüklükte 10 gruba veya katlara bölünmesini içerir. Verilerin %90'ı modeli eğitmek için, kalan %10'u ise onu doęrulamak için kullanılır. Yanlıř sınıflandırma oranı daha sonra %10 doęrulama verisi üzerinden hesaplanır. Bu prosedür 10 kez tekrarlanır. Farklı gözlem grupları, 10 defadan her biri bir doęrulama seti olarak ele alınır. Daha sonra ortalaması alınan doęrulama hatasının 10 tahminiyle sonuçlanır.

K-Nearest Neighbor(KNN) Algorithm for Machine Learning

- K-Nearest Neighbor, Denetimli Öğrenme tekniğine dayalı en basit Makine Öğrenimi algoritmalarından biridir.
- K-NN algoritması, yeni durum/veriler ile mevcut durumlar arasındaki benzerliği varsayar ve yeni durumu mevcut kategorilere en çok benzeyen kategoriye koyar.
- K-NN algoritması mevcut tüm verileri saklar ve benzerliğe göre yeni bir veri noktasını sınıflandırır. Bu, yeni veriler görüldüğünde, K-NN algoritması kullanılarak kolayca bir grup paketi kategorisine sınıflandırılabilmesi anlamına gelir.
- K-NN algoritması Sınıflandırmanın yanı sıra Regresyon için de kullanılabilir ancak çoğunlukla Sınıflandırma problemleri için kullanılır.
- K-NN, parametrik olmayan bir algoritmadır, yani temel veriler üzerinde herhangi bir varsayımda bulunmaz.
- Eğitim setinden hemen öğrenmediği için veri setini sakladığı ve sınıflandırma anında veri seti üzerinde bir işlem yaptığı için tembel öğrenen algoritması olarak da adlandırılır.
- KNN algoritması eğitim aşamasında sadece veri setini saklar ve yeni veri aldığında bu veriyi yeni veriye çok benzeyen bir kategoride sınıflandırır.

Örnek: Diyelim ki elimizde kedi ve köpeğe benzeyen bir yaratığın görüntüsü var ama onun kedi mi yoksa köpek mi olduğunu bilmek istiyoruz. Bu tanımlama için, benzerlik ölçüsü üzerinde çalıştığı için KNN algoritmasını kullanabiliriz. KNN modelimiz, yeni veri setinin kedi ve köpek resimlerine benzer özelliklerini bulacak ve en benzer özelliklere dayanarak onu kedi veya köpek kategorisine yerleştirecektir.



Uygulama:

KNN, sınıflandırma veya regresyon görevleri için kullanılabilen basit, denetimli bir makine öğrenimi (ML) algoritmasıdır ve ayrıca eksik değer atamasında sıklıkla kullanılır. Belirli bir veri noktasına en yakın gözlemlerin bir veri kümesindeki en "benzer" gözlemler olduğu fikrine dayanır ve bu nedenle öngörülemez noktaları mevcut en yakın noktaların değerlerine göre sınıflandırabiliriz. Kullanıcı, K'yi seçerek algoritmada kullanmak üzere yakındaki gözlemlerin sayısını seçebilir.

Burada, sınıflandırma için KNN algoritmasının nasıl uygulayacağı ve farklı K değerlerinin sonuçları nasıl etkilediği gösterilecektir.

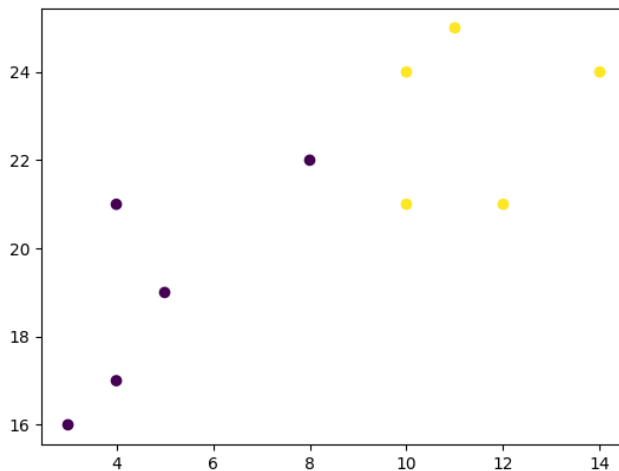
Nasıl çalışır?

K, kullanılacak en yakın komşu sayısıdır. Sınıflandırma için, yeni bir gözlemin hangi sınıfa gireceğini belirlemek için çoğunluk oyu kullanılır. Daha büyük K değerleri genellikle aykırı değerlere karşı daha sağlamdır ve çok küçük değerlerden daha kararlı karar sınırları üretir (K=3, K=1'den daha iyi olur, bu da istenmeyen sonuçlar doğurabilir).

Örnek: Bazı veri noktalarını görselleştirerek başlayalım.

```
import matplotlib.pyplot as plt
x = [4, 5, 10, 4, 3, 11, 14, 8, 10, 12]
y = [21, 19, 24, 17, 16, 25, 24, 22, 21, 21]
classes = [0, 0, 1, 0, 0, 1, 1, 0, 1, 1]
```

```
plt.scatter(x, y, c=classes)
plt.show()
```



Şimdi KNN algoritmasını K=1 ile uyumlu hale getirelim:

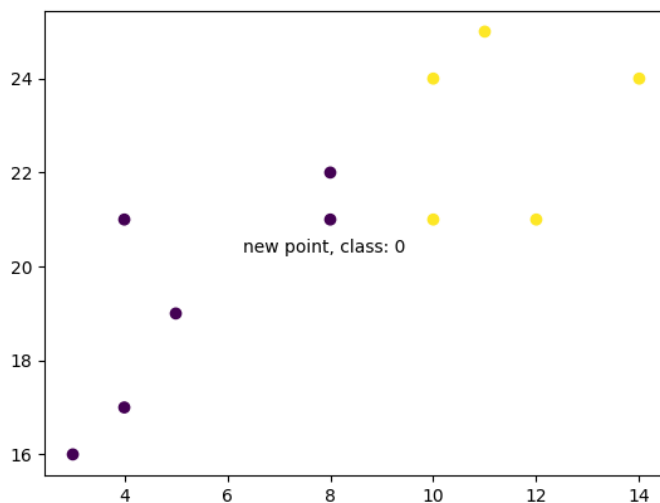
```
from sklearn.neighbors import KNeighborsClassifier
data = list(zip(x, y))
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(data, classes)
```

Ve yeni bir veri noktası sınıflandırılmak için kullanılır.

Kod örneği:

```
new_x = 8
new_y = 21
new_point = [(new_x, new_y)]
prediction = knn.predict(new_point)

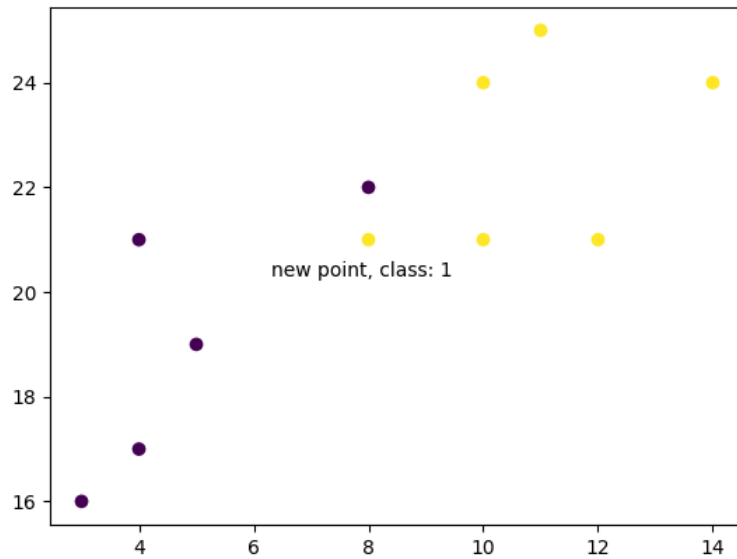
plt.scatter(x + [new_x], y + [new_y], c=classes + [prediction[0]])
plt.text(x=new_x-1.7, y=new_y-0.7, s=f"new point, class: {prediction[0]}")
plt.show()
```



Şimdi aynı şeyi yapıyoruz, ancak tahmini değiştiren daha yüksek bir K değeriyle.

Kod örneği:

```
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(data, classes)
prediction = knn.predict(new_point)
plt.scatter(x + [new_x], y + [new_y], c=classes + [prediction[0]])
plt.text(x=new_x-1.7, y=new_y-0.7, s=f"new point, class: {prediction[0]}")
plt.show()
```



Örneğin açıklanması:

İhtiyaç duyulan modüller içe aktarılır. scikit-learn, Python'da makine öğrenimi için popüler bir kütüphanedir.

```
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
```

Bir veri kümesindeki değişkenlere benzeyen diziler oluşturulur. İki girdi özelliği (x ve y) ve ardından bir hedef sınıf (sınıf) var. Hedef sınıfla önceden etiketlenmiş girdi özellikleri, yeni veri sınıfını tahmin etmek için kullanılacaktır. Burada yalnızca iki giriş özelliği kullanırken, bu yöntemin herhangi bir sayıda değişkenle çalışacağını unutmayın:

```
x = [4, 5, 10, 4, 3, 11, 14, 8, 10, 12]
y = [21, 19, 24, 17, 16, 25, 24, 22, 21, 21]
classes = [0, 0, 1, 0, 0, 1, 1, 0, 1, 1]
Turn the input features into a set of points:
data = list(zip(x, y))
print(data)
```

Sonuç:

```
[(4, 21), (5, 19), (10, 24), (4, 17), (3, 16), (11, 25), (14, 24), (8, 22), (10, 21), (12, 21)]
```

Girdi özelliklerini ve hedef sınıfı kullanarak, en yakın 1 komşuyu kullanarak modele bir KNN modeli sağlıyoruz.

```
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(data, classes)
```

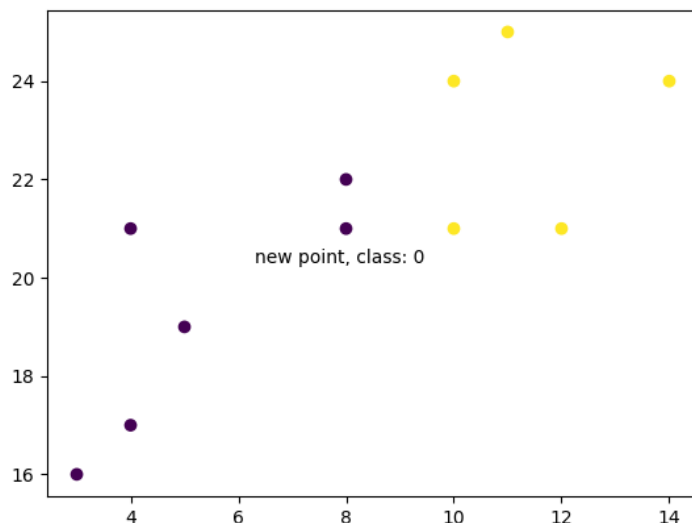
Ardından, yeni, öngörülemez veri noktalarının sınıfını tahmin etmek için aynı KNN nesnesini kullanabiliriz. İlk önce yeni x ve y özellikleri yaratırız ve ardından 0 veya 1 sınıfı elde etmek için yeni veri noktasında `knn.predict()` ögesini çağırırız:

```
new_x = 8
new_y = 21
new_point = [(new_x, new_y)]
prediction = knn.predict(new_point)
print(prediction)
```

Sonuç: [0]

Tüm verileri yeni nokta ve sınıfla birlikte çizdiğimizde, 1 sınıfı ile mavi olarak etiketlendiğini görebiliriz. Metin açıklaması yalnızca yeni noktanın konumunu vurgulamak içindir:

```
plt.scatter(x + [new_x], y + [new_y], c=classes + [prediction[0]])
plt.text(x=new_x-1.7, y=new_y-0.7, s=f"new point, class: {prediction[0]}")
plt.show()
```



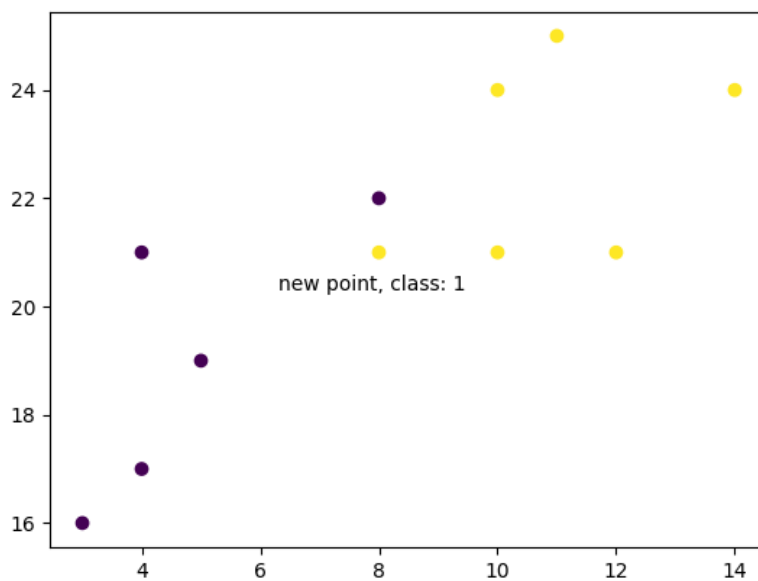
Ancak komşu sayısını 5 olarak değiştirdiğimizde, yeni noktamızı sınıflandırmak için kullanılan nokta sayısı değişir. Sonuç olarak, yeni noktanın sınıflandırılması da öyle:

```
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(data, classes)
prediction = knn.predict(new_point)
print(prediction)
```

Sonuç: [1]

Yeni noktanın sınıfını eski noktalarla birlikte çizdiğimizde, ilgili sınıf etiketine göre rengin değiştiğini not ederiz:

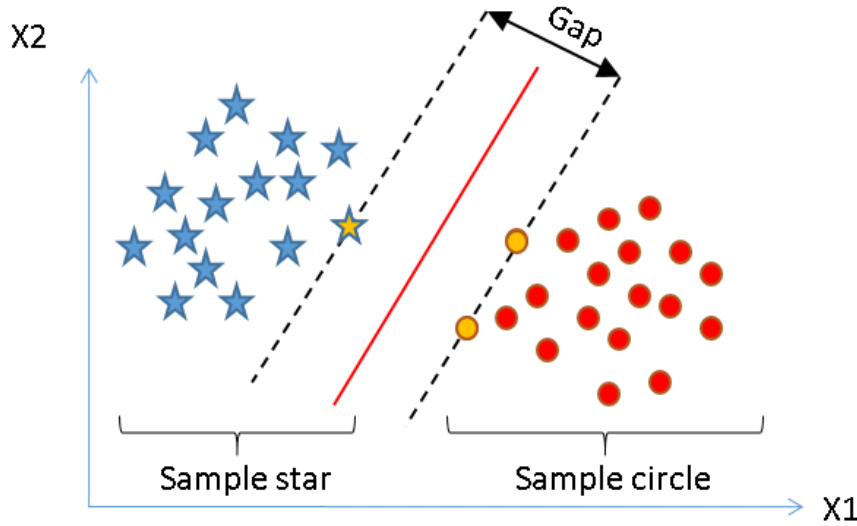
```
plt.scatter(x + [new_x], y + [new_y], c=classes + [prediction[0]])
plt.text(x=new_x-1.7, y=new_y-0.7, s=f"new point, class: {prediction[0]}")
plt.show()
```



5.1.4. Destek Vektör Makineleri

Destek Vektör Makinesi, farklı sınıfları ayırmak ve sınır marjını en üst düzeye çıkarmak için karar sınırlarını bulmaktan sorumludur. Farklı sınıfların sınırları arasındaki boşluklar, çizgi ile çizgiye en yakın noktalar arasındaki (dik) mesafelerdir. DVM'de sınıflar arasındaki sınırlar çok önemlidir.

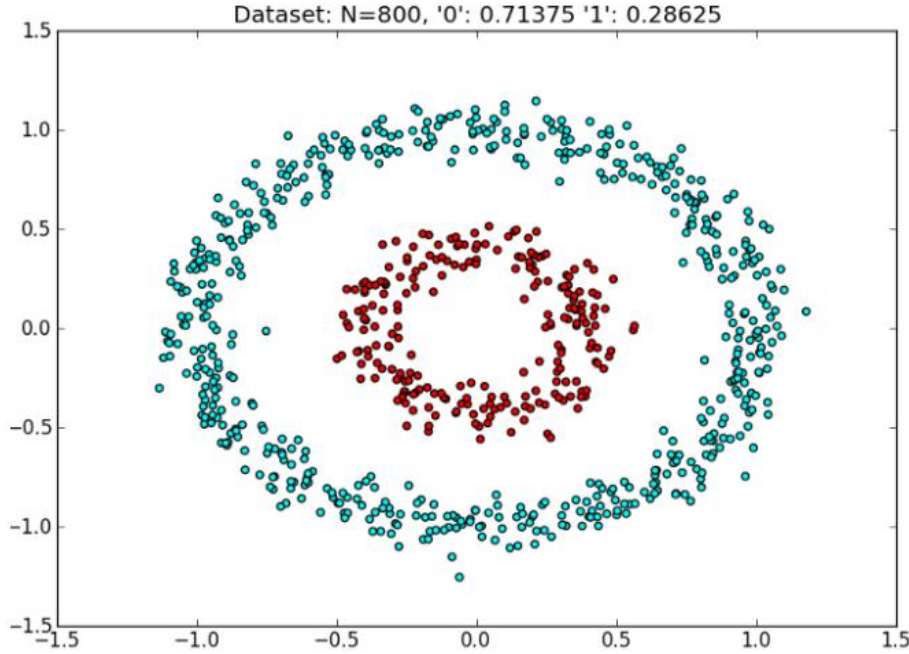
İki sınıflı bir örnekle başlayalım: Verilen X_1 ve X_2 sınıfları için, iki sınıfı en iyi, yani minimum hata ile ayıran karar sınırını bulmak istiyoruz. SVM bunu bir "Hiperplane" ile yapar. Şimdi bu hiperdüzlem 2 boyutlu veri olması durumunda tek bir doğru olabilir ve 3 boyutlu veri olması durumunda bir düzlem olabilir.



Destek Vektör Makineleri, hiper düzleme en yakın noktalar olan 'Destek Vektörleri' kavramını kullanır. Yukarıdaki örnekte, kırmızı çizgi, tampon bölgenin ortasından geçen doğrusal hattır. İki sınıfı (Mavi yıldızlar ve Kırmızı daireler) ayıran **karar sınırımızı** gösterir. Tireli çizgiler, her iki sınıfın Destek Vektörleri arasında istediğimiz veri yığını ayıran **bölge sınırıdır**, 'Marj'ımızı temsil eder.

Sınırlar önemlidir. Veri bölgeleri sınırı üzerindeki verilerin kırmızı çizgiye uzaklığı hesaplanır. Marj, Destek Vektörlerinin (dolayısıyla algoritmanın adı) yardımıyla tanımlanır. Örneğimizde, Sarı yıldızlar ve Sarı daireler, Marjı tanımlayan Destek Vektörleridir. Boşluk ne kadar iyi olursa, sınıflandırıcı o kadar iyi çalışır. Bu nedenle destek vektörleri sınıflandırıcısının geliştirilmesinde önemli bir rol oynamaktadır. Test verilerindeki her yeni veri noktası bu Marj'a göre sınıflandırılacaktır. Sağ tarafındaysa Kırmızı daire, aksi halde Mavi yıldız olarak sınıflandırılır.

En iyi yanı, SVM'nin doğrusal olmayan verileri de sınıflandırabilmesidir.



Doğrusal olmayan veriler söz konusu olduğunda işler biraz zorlaşır. Burada SVM 'Çekirdek hilesi' kullanır, **doğrusal olmayan verileri daha yüksek boyutlara eşlemek için bir çekirdek işlevi kullanır, böylece doğrusal hale gelir ve orada karar sınırını bulur**. Bir Çekirdek işlevi, ister doğrusal ister doğrusal olmayan veri olsun, SVM tarafından her zaman kullanılır, ancak ana işlevi, veriler mevcut haliyle ayrılmaz olduğunda devreye girer. Burada, Çekirdek işlevi, sınıflandırma sorununa boyutlar ekler.

Soruna bağlı olarak, farklı türde Çekirdek işlevleri kullanabilirsiniz:

- Doğrusal
- Polinom
- Radyal Temel Fonksiyonu
- Gauss
- Laplace

... ve daha fazlası. Doğru çekirdek işlevini seçmek, sınıflandırıcıyı oluşturmak için önemlidir.

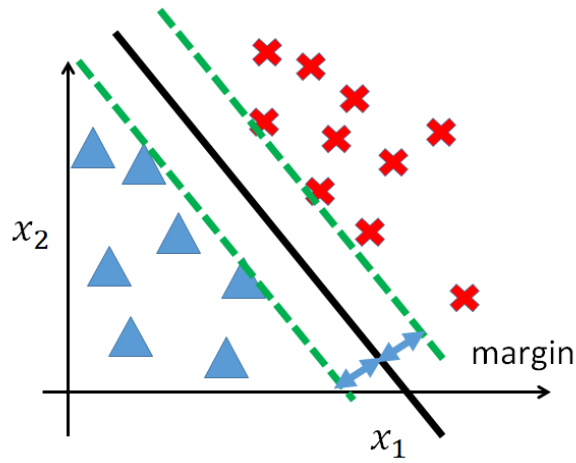
Destek vektör ağları olarak da bilinen destek vektör makineleri, sınıflandırma ve regresyon için kullanılan ilgili denetimli öğrenme yöntemleri kümesidir. Her biri iki kategoriden birine ait olarak işaretlenmiş bir dizi eğitim örneği verildiğinde, bir DVM eğitim algoritması yeni bir örneğin bir kategoriye mi yoksa diğerine mi düştüğünü tahmin eden bir model oluşturur.

SVM'nin (Destek Vektör Makinesi) kullanabileceği iki sınıflandırma yöntemi:

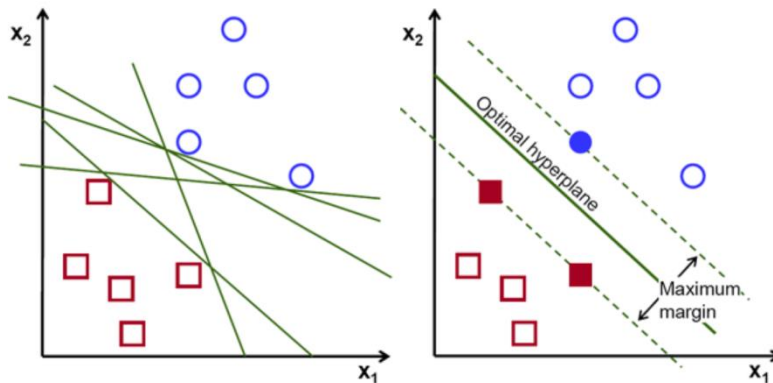
- İkili sınıflandırıcıları birleştirme
- Çok sınıflı öğrenmeyi dahil etmek için ikili programı değiştirme

Destek Vektör Makinesi algoritması oluşturulurken,

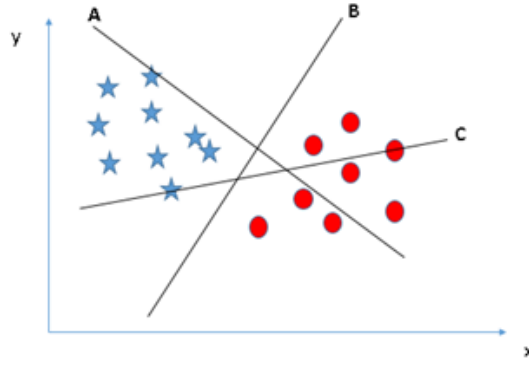
- Sınıflar arasındaki birbirine en yakın ikili seçilir. Bu noktalara destek vektörleri isimleri veriyoruz.
- Destek vektörlerinden geçecek şekilde doğrular çizilir. Bu doğrulara sınır çizgisi adı verilir.
- Her doğruya eşit uzakta çizilen doğruya karar doğrusu adı verilir. Karar doğrusuna hiper düzlem de denir.



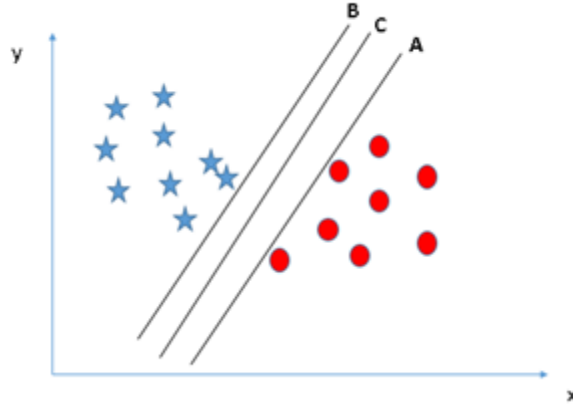
DVM'lerde sınıflar +1 ve -1 olarak etiketlenir. Karar doğrusunun üst kısmında kalan doğru denklemi, $wx+b=1$, altında kalan doğru denklemi ise $wx+b=-1$ olarak belirlenir. Karar doğrusu denklemi ise, $wx+b=0$ olur.



Doğru hiper düzlemi belirleyin: Burada üç hiper düzlemimiz var (A, B ve C). Şimdi, yıldızları ve daireleri sınıflandırmak için doğru hiper düzlemi belirleyin.

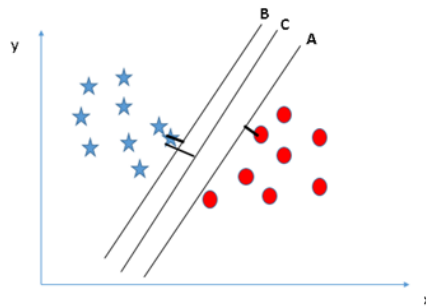


Doğru hiper düzlemi belirlemek için basit bir kuralı hatırlamanız gerekir: “İki sınıfı daha iyi ayıran hiper düzlemi seçin”. Bu senaryoda, hiper-düzlem “B” bu işi mükemmel bir şekilde gerçekleştirmiştir. **B çizgisi hiçbirine değmez ve teğet geçmez.**



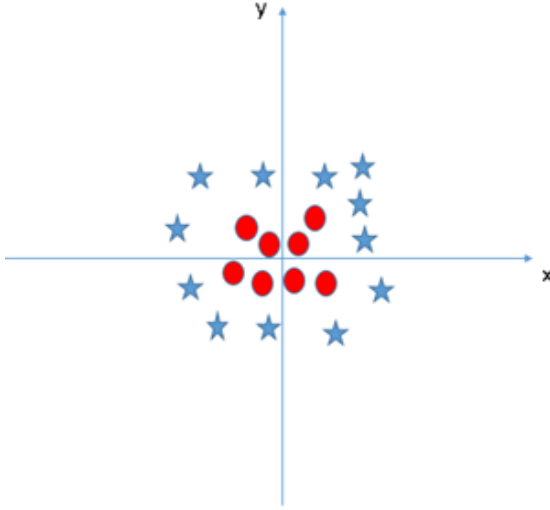
Burada en yakın veri noktası (her iki sınıf) ile hiper düzlem arasındaki mesafeleri maksimize etmek, doğru hiper düzleme karar vermemize yardımcı olacaktır. Bu mesafeye Marj denir.

Aşağıdaki anlık görüntüye bakalım:

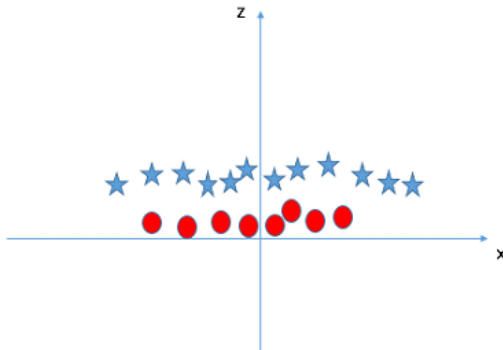


Yukarıda, C hiper düzleminin marjının hem A hem de B'ye kıyasla yüksek olduğunu görebilirsiniz. Bu nedenle, sağ hiper düzlemi C olarak adlandırıyoruz. Daha yüksek marjlı hiper düzlemi seçmenin bir başka yıldırım nedeni sağlamlıktır. Düşük marjlı bir hiper-düzlem seçersek, yüksek yanlış sınıflandırma şansı vardır.

Aşağıdaki senaryoda, iki sınıf arasında doğrusal hiper düzleme sahip olamayız, peki SVM bu iki sınıfı nasıl sınıflandırır? Şimdiye kadar sadece lineer hiper düzleme baktık.



SVM bu sorunu çözebilir. Kolayca! Ek özellik sunarak bu sorunu çözer. Burada yeni bir özellik $z=x^2+y^2$ ekleyeceğiz. Şimdi veri noktalarını x ve z eksenlerine yerleştirelim:



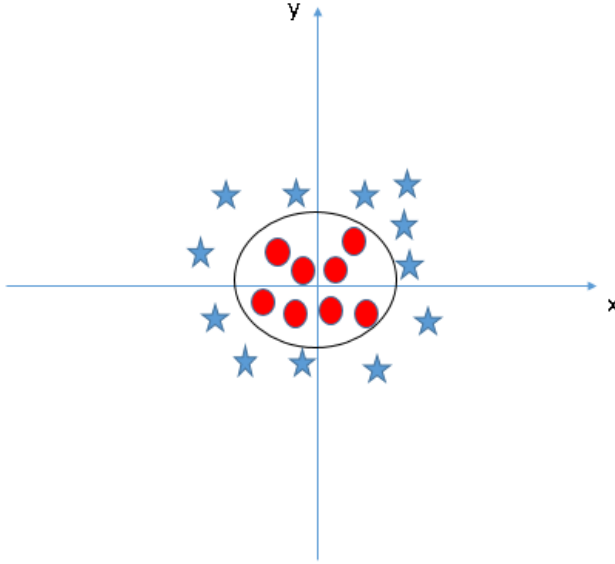
Yukarıdaki arsada, dikkate alınması gereken noktalar şunlardır:

- z için tüm değerler her zaman pozitif olacaktır çünkü z hem x hem de y'nin karelerinin toplamıdır
- Orijinal çizimde, x ve y eksenlerinin orijine yakın kırmızı daireler görünür, bu da daha düşük z değerine ve orijinden nispeten uzaktaki yıldızın daha yüksek z değerine yol açmasına neden olur.

DVM sınıflandırıcısında, bu iki sınıf arasında doğrusal bir hiper düzleme sahip olmak kolaydır. Ancak, ortaya çıkan bir diğer yakıcı soru, **bir hiper düzleme sahip olmak için bu özelliği manuel olarak eklememiz gerekip gerekmediğidir**. Hayır, SVM algoritmasının çekirdek hilesi adı verilen bir tekniği vardır. SVM çekirdeği, düşük boyutlu girdi uzayını alan ve onu daha

yüksek boyutlu bir uzaya dönüştüren, yani ayrılamayan problemi ayrılabilir probleme dönüştüren bir fonksiyondur. Çoğunlukla doğrusal olmayan ayırma probleminde kullanışlıdır. Basitçe söylemek gerekirse, bazı son derece karmaşık veri dönüşümleri yapar ve ardından tanımladığınız etiketlere veya çıktılara göre verileri ayırma sürecini bulur.

Orijinal girdi uzayındaki hiper düzleme baktığımızda bir daireye benziyor:



Kütüphaneleri İndirme, Çalışma Dizinini Ayarlama, Veri Setini İndirme

```
import numpy as np
```

```
import matplotlib.pyplot as plt
```

```
import pandas as pd
```

```
import os
```

```
os.chdir('Calisma_Dizininiz')
```

```
dataset = pd.read_csv('SosyalMedyaReklamKampanyasi.csv')
```

Spyder'ın variable explorer penceresinden veri setimizi görelim:

Index	KullaniciID	Cinsiyet	Yas	TahminiMaas	SatinAldiMi
0	15624510	Erkek	19	19000	0
1	15810944	Erkek	35	20000	0
2	15668575	Kadın	26	43000	0
3	15603246	Kadın	27	57000	0
4	15804002	Erkek	19	76000	0
5	15728773	Erkek	27	58000	0
6	15598044	Kadın	27	84000	0
7	15694829	Kadın	32	150000	1
8	15600575	Erkek	25	33000	0
9	15727311	Kadın	35	65000	0
10	15570769	Kadın	26	80000	0
11	15606274	Kadın	26	52000	0
12	15746139	Erkek	20	86000	0
13	15704987	Erkek	32	18000	0
14	15628972	Erkek	18	82000	0
15	15697686	Erkek	29	80000	0
16	15733883	Erkek	47	25000	1
17	15617482	Erkek	45	26000	1
18	15704583	Erkek	46	28000	1
19	15621083	Kadın	48	29000	1
20	15649487	Erkek	45	22000	1
21	15736760	Kadın	47	49000	1

Format Resize ☒ Background color ☒ Column min/max OK Cancel

Veriyi Anlamak

Yukarıda gördüğümüz veri seti beş nitelikten oluşuyor. Veri seti bir sosyal medya kayıtlarından derlenmiş durumda. KullaniciID müşteriye belirleyen eşsiz rakam, Cinsiyet, Yaş, Tahmini Gelir yıllık tahmin edilen gelir, SatınAldiMi ise belirli bir ürünü satın almış olup olmadığı, hadi lüks araba diyelim. Bu veri setinde kolayca anlaşılabilceği gibi hedef değişkenimiz SatınAldiMi'dir. Diğer dört nitelik ise bağımsız niteliklerdir. Bu bağımsız niteliklerle bağımlı nitelik (satın alma davranışının gerçekleşip gerçekleşmeyeceği) tahmin edilecek.

Veri Setini Bağımlı ve Bağımsız Niteliklere Ayırmak

Yukarıda gördüğümüz niteliklerden bağımsız değişken olarak sadece yaş ve tahmini maaşı kullanacağız.

X = dataset.iloc[:, [2,3]].values

y = dataset.iloc[:, 4].values

X - NumPy array

	0	1
0	19	19000
1	35	20000
2	26	43000
3	27	57000
4	19	76000
5	27	58000
6	27	84000
7	32	150000
8	25	33000
9	35	65000
10	26	80000
11	26	52000
12	20	86000
13	32	18000
14	18	82000
15	29	80000
16	47	25000

Format Resize ☒ Background color

y - NumPy array

	0
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	1
8	0
9	0
10	0
11	0
12	0
13	0
14	0
15	0
16	1

Format Resize ☒ Background color

OK Cancel

Veriyi Eğitim ve Test Olarak Ayırmak

Veri setinde 400 kayıt var bunun 300'ünü eğitim, 100'ünü test için ayıralım.

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
```

Normalizasyon – Feature Scaling

Bağımsız değişkenlerden yaş ile tahmini gelir aynı birimde olmadığı için feature scaling uygulayacağız.

```
from sklearn.preprocessing import StandardScaler
```

```
sc_X = StandardScaler()
```

```
X_train = sc_X.fit_transform(X_train)
```

```
X_test = sc_X.transform(X_test)
```

SVM Modeli Oluşturmak ve Eğitmek

Şimdi scikit-learn kütüphanesi svm modülü SVC sınıfından oluşturacağımız classifier nesnesi ile modelimiz oluşturalım.

```
from sklearn.svm import SVC
```

```
classifier = SVC(kernel='linear', random_state = 0)
```

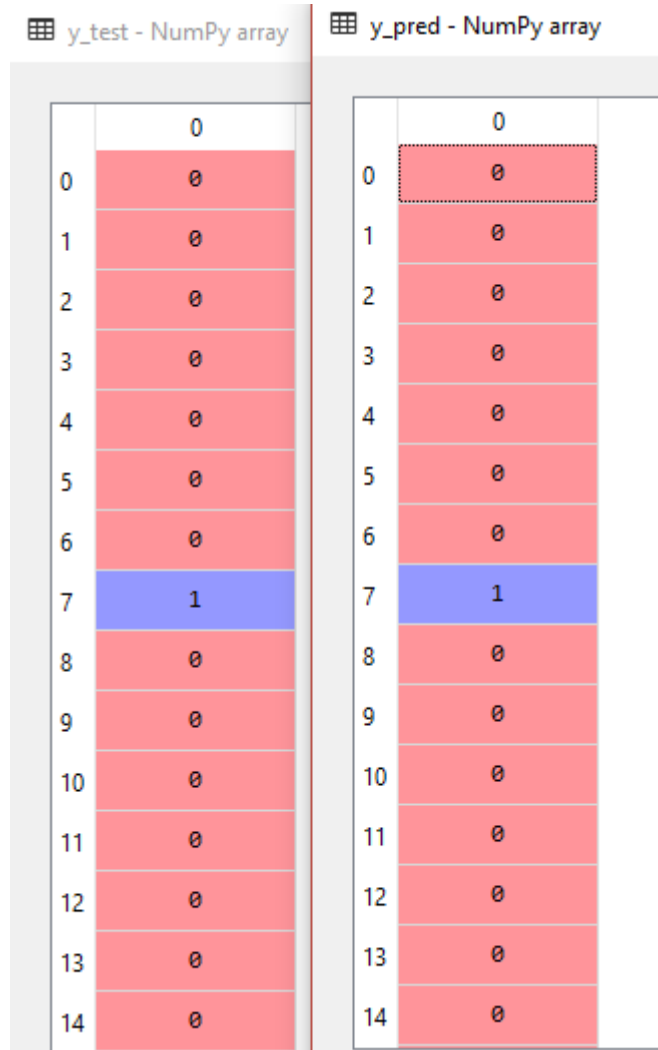
```
classifier.fit(X_train, y_train)
```

Sınıf parametrelerinden biraz bahsedelim. kernel linear, her seferinde aynı sonuçları almak için de random_state=0 diyoruz.

Test Seti ile Tahmin Yapmak

Ayırdığımız test setimizi (X_test) kullanarak oluşturduğumuz model ile tahmin yapalım ve elde ettiğimiz set (y_pred) ile hedef değişken (y_test) test setimizi karşılaştıralım.

```
y_pred = classifier.predict(X_test)
```



Yukarıda y_test (gerçek veri) ile modelin tahmin ettiği y_pred bir görüntü bulunuyor. gördüğümüz kadarıyla tüm tahminler doğru görünüyor ancak daha buradan göremediğimiz bir çok değer var. Neyse hata matrisinde görünür artık.

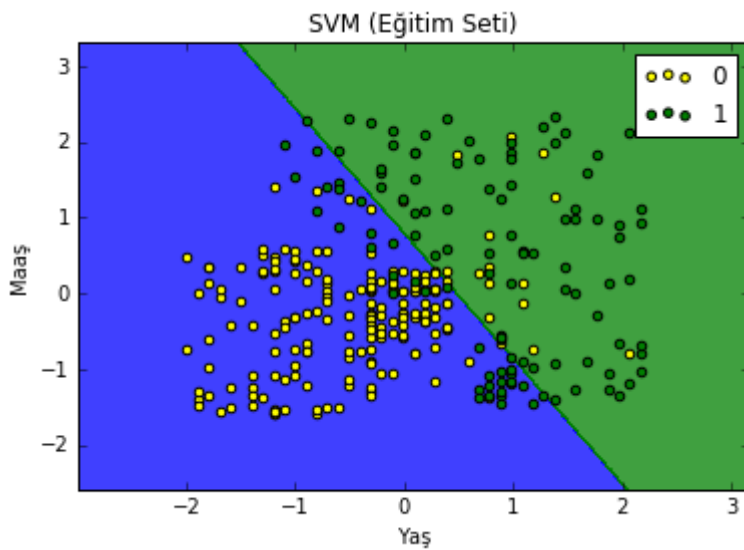
Hata Matrisini Oluşturma

Yaptığımız sınıflandırmanın doğruluğunu kontrol etme yöntemlerinden birisi de hata matrisi oluşturmaktır. Hata matrisi için scikit-learn kütüphanesi metrics modülü confusion_matrix fonksiyonunu kullanıyoruz.

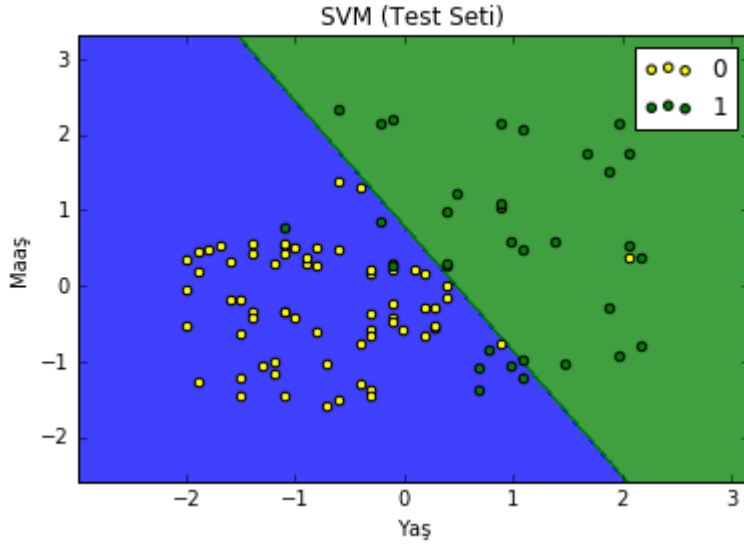
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)
[[66 2]
 [ 8 24]]
```

Bildiğiniz gibi 100 kayıtlık test verisi ayırmıştık. Yukarıda gördüğümüz hata matrisine göre 10 kayıt yanlış sınıflandırılmış, 90 kayıt doğru sınıflandırılmış. Lojistik regresyonda yanlış sınıflandırma sayısı 11 idi. K en yakın komşuda ise 7 idi. Grafiğimizi görelim:

```
from matplotlib.colors import ListedColormap
X_set, y_set = X_train, y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step
= 0.01),
np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
alpha = 0.75, cmap = ListedColormap(('blue', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
c = ListedColormap(('yellow', 'green'))(i), label = j)
plt.title('SVM (Eğitim Seti)')
plt.xlabel('Yaş')
plt.ylabel('Maaş')
plt.legend()
plt.show()
```



Şimdi grafiğimizi test setleri için çizelim. Bunun için yukarıdaki kodda veri setini ve etiket bilgilerini değiştirmek yeterli olur.



Yanlış sınıflandırılan 10 noktayı buradan sayabiliriz. Grafik lojistik regresyona çok benzedi. Zaten hata sayımız da birbirine çok yakındı.

5.1.5. Naive Bayes

Bayes Ağı, bir dizi değişken arasındaki ilişkilerin olasılıklarını öğrenerek çıkarım yapan bir makine öğrenmesi algoritmasıdır. Düşünce ya da oluşan kanı veya yönlendirilmiş olasılıklı bir modeldir. Örneğin, bir Bayes ağı, hastalıklar ve semptomlar arasındaki olasılık ilişkilerini temsil edebilir. Belirtiler verildiğinde, ağ çeşitli hastalıkların varlığının olasılıklarını hesaplamak için kullanılabilir.

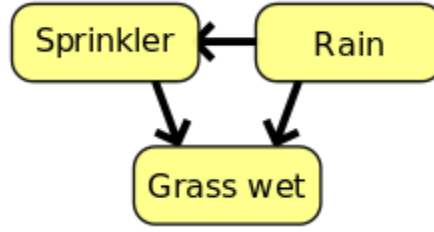
Konuşma sinyalleri veya protein dizileri gibi değişken dizilerini modelleyen Bayes ağlarına dinamik Bayes şebekeleri denir. Belirsizlik altında karar problemlerini temsil edebilen ve çözebilen Bayes ağlarının genellemelerine etki diyagramları denir.

Naïve Bayes'te sınıflandırıcı, lojistik regresyon gibi ayırt edici modellere göre daha hızlı birleşir, bu nedenle daha az eğitim verisine ihtiyacınız vardır. Ana avantajı, özellikler arasındaki etkileşimleri öğrenememesidir.

Bayesci mantık programı iki bileşenden oluşur. İlk bileşen mantıklı bir bileşendir; alanın niteliksel yapısını yakalayan bir dizi Bayes Cümlelerinden oluşur. İkinci bileşen niceliksel, alanla ilgili nicel bilgileri kodlar.

Yağmur, yağmurlama sisteminin etkinleştirilip etkinleştirilmeyeceğini etkiler ve hem yağmur hem de yağmurlama sistemi çimlerin ıslak olup olmadığını etkiler. Çim ıslak ise yağmu mu yağdı yoksa yağmurlama sistemi mi çalıştı.

Örneğin, Cebrail pikniğe gitmiş olsun, bu durumda Cebrail'in olduğu yer de yağmur yağmuyor.



Örnek:

Test altındaki sistemin giriş sayısı 5 çıkış sayısı 1 dir. Girişler ikili sayı sisteminde bit olarak uygulanmaktadır. Toplam test sayısı 20 cihaz ise, testten geçen cihaz sayısı 18 ise testten kalma olasılığı nedir?

Örnek:

Gün	Görünüş	Sıcaklık	Nem	Rüzgar	Play
1	Güneşli	Sıcak	Yüksek	Zayıf	Hayır
2	Güneşli	Sıcak	Yüksek	Kuvvetli	Hayır
3	Bulutlu	Sıcak	Yüksek	Zayıf	Evet
4	Yağmurlu	Hafif	Yüksek	Zayıf	Evet
5	Yağmurlu	Soğuk	Normal	Zayıf	Evet
6	Yağmurlu	Soğuk	Normal	Kuvvetli	Hayır
7	Bulutlu	Soğuk	Normal	Kuvvetli	Evet
8	Güneşli	Hafif	Yüksek	Zayıf	hayır
9	Güneşli	Soğuk	Normal	Zayıf	Evet
10	Yağmurlu	Hafif	Normal	Zayıf	Evet
11	Güneşli	Hafif	Normal	Kuvvetli	Evet
12	Bulutlu	Hafif	Yüksek	Kuvvetli	Evet
13	Bulutlu	Sıcak	Normal	Zayıf	Evet
14	Yağmurlu	Hafif	Yüksek	Kuvvetli	hayır
	Güneşli	Soğuk	Yüksek	Kuvvetli	?

Toplam Gün Sayısı=14

Evet=9

Hayır=5

$P(\text{Evet})=9/14$

$P(\text{Hayır})=5/14$

Havanın görünüş durumuna göre,

Havanın görünüşü güneşli olduğunda 2 gün oyun var. Hava güneşli olduğunda oyun oynama olasılığı, $P(H_Güneşli \mid \text{Evet})=2/9$

Havanın görünüşü güneşli olduğunda 3 gün oyun yok. $P(H_Güneşli \mid \text{Hayır})=3/5$

Toplam güneşli gün sayısı=2+3=5

Havanın görünüşü bulutlu olduğunda 4 gün oyun var. $P(H_Bulutlu \mid \text{Evet})=4/9$

Havanın görünüşü bulutlu olduğunda 0 gün oyun yok. $P(H_Bulutlu \mid \text{Hayır})=0/5$

Toplam bulutlu gün sayısı=4+0=4

Havanın görünüşü yağmurlu olduğunda 3 gün oyun var. $P(H_Yağmurlu \mid \text{Evet})=3/9$

Havanın görünüşü yağmurlu olduğunda 2 gün oyun yok. $P(H_Yağmurlu \mid \text{Hayır})=2/5$

Toplam yağmurlu gün sayısı=2+3=5

Toplam gün sayısı=5+4+5=14

Sıcaklık durumuna göre,

Sıcaklığın sıcak olduğunda 2 gün oyun var. $P(S_Sıcak \mid \text{Evet})=2/9$

Sıcaklığın sıcak olduğunda 2 gün oyun yok. $P(S_Sıcak \mid \text{Hayır})=2/5$

Toplam sıcak gün sayısı=2+2=4

Sıcaklığın hafif olduğunda 4 gün oyun var. $P(S_Hafif \mid \text{Evet})=4/9$

Sıcaklığın hafif olduğunda 2 gün oyun yok. $P(S_Hafif \mid \text{Hayır})=2/5$

Toplam hafif gün sayısı=4+2=6

Sıcaklığın soğuk olduğunda 3 gün oyun var. $P(S_Soğuk \mid \text{Evet})=3/9$

Sıcaklığın soğuk olduğunda 1 gün oyun yok. $P(S_Soğuk \mid \text{Hayır})=1/5$

Toplam soğuk gün sayısı=3+1=4

Toplam gün sayısı=4+6+4=14

Rüzgar durumuna göre,

Rüzgar zayıf olduğunda 6 gün oyun var. $P(R_Zayıf \mid \text{Evet})=6/9=2/3$

Rüzgar zayıf olduğunda 2 gün oyun yok. $P(R_Zayıf \mid \text{Hayır})=2/5$

Toplam rüzgar zayıf gün sayısı=6+2=8

Rüzgar Kuvvetli olduğunda 3 gün oyun var. $P(R_Kuvvetli \mid \text{Evet})=3/9=1/3$

Rüzgar Kuvvetli olduğunda 3 gün oyun yok. $P(R_Kuvvetli \mid \text{Hayır})=3/5$

Toplam rüzgar kuvvetli gün sayısı=3+3=6

Toplam gün sayısı=8+6=14

Nem durumuna göre,

Nem yüksek olduğunda 3 gün oyun var. $P(N_Yüksek | Evet)=3/9$

Nem yüksek olduğunda 4 gün oyun yok. $P(N_Yüksek | Hayır)=4/5$

Toplam nem yüksek gün sayısı=3+4=7

Nem normal olduğunda 6 gün oyun var. $P(N_Normal | Evet)=6/9$

Nem normal olduğunda 1 gün oyun yok. $P(N_Normal | Hayır)=1/5$

Toplam nem zayıf gün sayısı=6+1=7

Toplam gün sayısı=7+7=14

$X=\{\text{Güneşli, Soğuk, Yüksek, Kuvvetli}\}$ ise

$P(X | Evet) = P(Evet) * P(H_Güneşli | Evet) * P(S_Soğuk | Evet) * P(R_Kuvvetli | Evet) * P(N_Yüksek | Evet)$

$P(X | Evet) = (9/14) * (2/9) * (1/9) * (3/9) = (1/7) * (1/27) = 1/189 = 0.053$

$P(X | Hayır) = P(Hayır) * P(H_Güneşli | Hayır) * P(S_Soğuk | Hayır) * P(R_Kuvvetli | Hayır) * P(N_Yüksek | Hayır)$

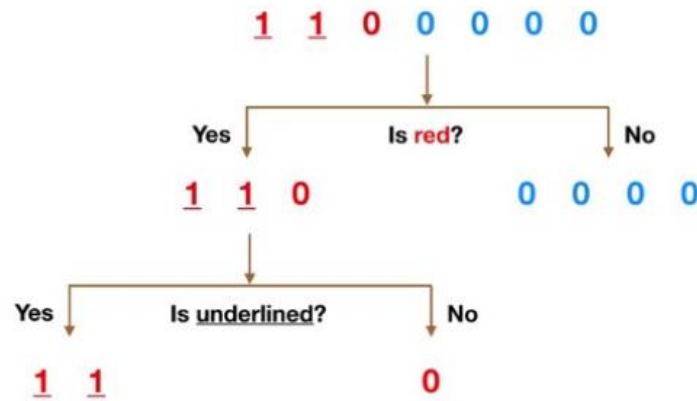
$P(X | Hayır) = (5/14) * (3/5) * (1/5) * (3/5) * (4/5) = (3/14) * (12/125) = 18/875 = 0.0206$

5.1.6. Rassel Orman Modeli

Rastgele ormanlar, denetimli bir öğrenme algoritmasıdır. Hem sınıflandırma hem de regresyon için kullanılabilir.

Karar ağaçları:

Rastgele orman modelinin yapı taşları oldukları için karar ağaçlarının bilinmesi gerekmektedir. Oldukça sezgisel yaklaşımlar içerir. Çoğu insanın hayatlarının bir noktasında bilerek ya da bilmeyerek bir karar ağacı kullandığına bahse girerim. Bir karar ağacının nasıl çalıştığını bir örnek üzerinden anlamak muhtemelen çok daha kolaydır.



Veri setimizin soldaki şeklin üstündeki sayılardan oluştuğunu hayal edin. İki 1 ve beş 0'ımız var (1'ler ve 0'lar sınıflarımızdır) ve özelliklerini kullanarak sınıfları ayırmak istiyoruz. Özellikler renklidir (kırmızıya karşı mavi) ve gözlemin altı çizili olup olmadığıdır. Peki bunu nasıl yapabiliriz?

Renk, 0'lardan biri hariç tümü mavi olduğu için, ayrılması oldukça bariz bir özellik gibi görünüyor. Böylece "Kırmızı mı?" Sorusunu kullanabiliriz. İlk düğümümüzü ayırmak için. Bir ağaçtaki bir düğümü, yolun ikiye ayrıldığı nokta olarak düşünebilirsiniz - kriterleri karşılayan gözlemler Evet dalına ve Hayır dalına inmeyenler.

Hayır dalı (blues) artık 0'lardır, bu yüzden orada işimiz bitti, ancak Evet şubemiz yine de bölünebilir. Şimdi ikinci özelliği kullanıp "Altı çizili mi?" Diye sorabiliriz. İkinci bir bölme yapmak için.

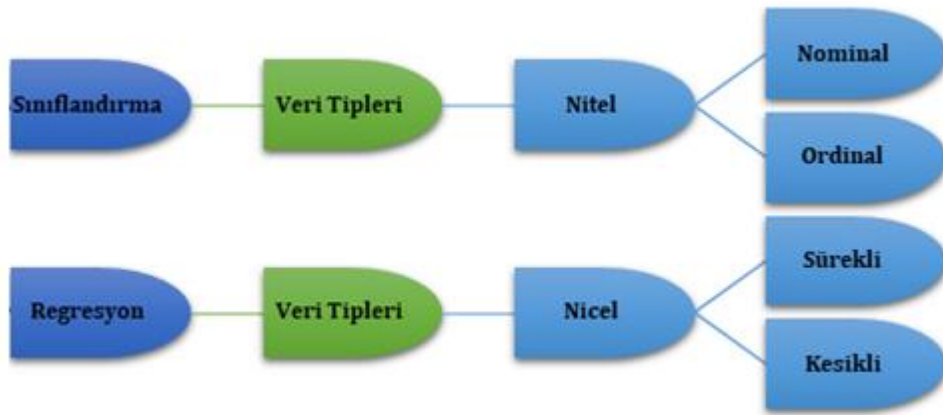
Altı çizili iki 1, Evet alt dalına gider ve altı çizilmemiş 0, sağ alt daldan aşağı gider ve hepimiz işimiz biter. Karar ağacımız, verileri mükemmel bir şekilde bölmek için iki özelliği kullanabildi. Açıkçası gerçek hayatta verilerimiz bu kadar net olmayacak, ancak bir karar ağacının kullandığı mantık aynı kalacak. Her düğümde sorulacak - Hangi özellik, eldeki gözlemleri,

ortaya çıkan grupların olabildiğince farklı olacağı şekilde (ve ortaya çıkan her alt grubun üyeleri mümkün olduğunca birbirine benzeyecek şekilde) bölmeme izin verir?

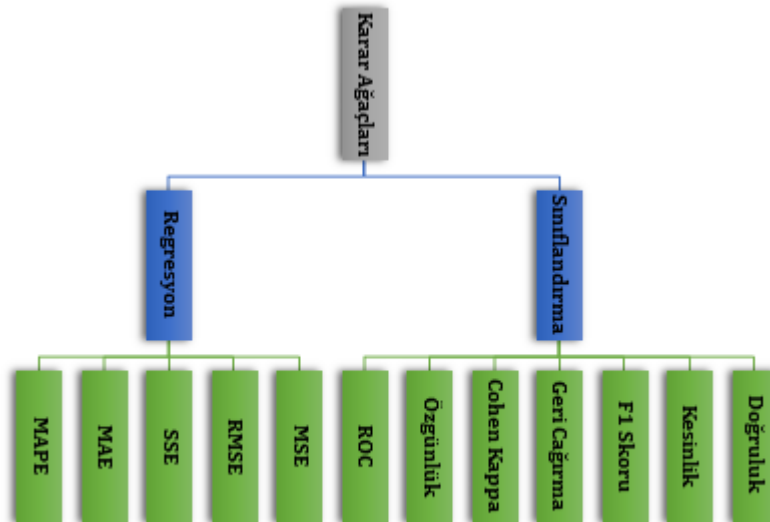
Ağaç tabanlı algoritmalar, denetimli öğrenme problemlerini çözmek için kullanılan popüler makine öğrenme yöntemleridir. Ağaç tabanlı algoritmalar, yüksek doğruluk , kararlılık ve yorumlama kolaylığına sahip tahminler üretirler .

Bir karar ağacı temel olarak 4 ana düğüm (node)'den oluşur: Kök (root) düğüm, iç düğümler ve yaprak (leaf) düğümlerden oluşur. Bu düğümler dallar aracılığıyla birbirine bağlanır.

Karar ağaçlarında problem sınıfına göre veri türleri:



Karar ağaçlarında kurulan modelin veya modellerin performansını değerlendirmede kullanılan hata metrikleri ise genel itibariyle aşağıdaki şekilde verilmiştir. Şekilde yer verilen hata metrikleri gerek makine öğrenme gerekse derin öğrenme modellerinin performansının testinde sıklıkla kullanılmaktadır.

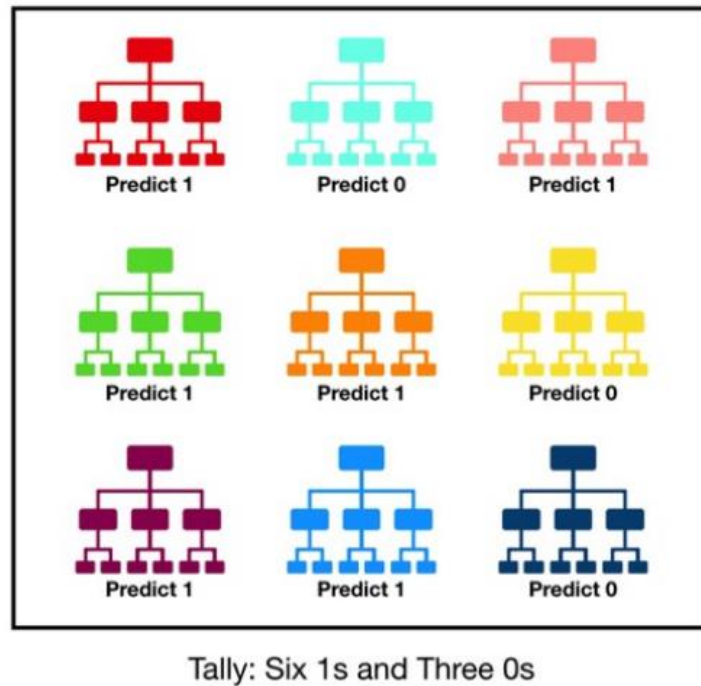


Farklı ağaç tabanlı algoritmalar vardır:

- Karar ağaçları
- Rastgele Orman
- Gradyan Artırma
- Torbalama (Bootstrap Toplama)

Rastgele Orman Sınıflandırıcısı:

Rastgele orman, adından da anlaşılacağı gibi, bir topluluk olarak çalışan çok sayıda bireysel karar ağacından oluşur. Rastgele ormandaki her bir ağaç bir sınıf tahmini verir ve en çok oyu alan sınıf, modelimizin öngörüsü haline gelir (aşağıdaki şekle bakın).



Rastgele ormanın ardındaki temel kavram basit ama güçlü bir kavramdır - kalabalıkların bilgeliği.

Rastgele orman, en popüler ağaç tabanlı denetimli öğrenme algoritmalarından biridir. Aynı zamanda en esnek ve kullanımı kolaydır. Karar ağaçlarının en büyük problemlerinden biri aşırı öğrenme-veriyi ezberlemedir (overfitting). Rassal orman modeli bu problemi çözmek için hem veri setinden hem de öznitelik setinden rassal olarak 10'larca 100'lerce farklı alt-setler seçilir ve bunlar eğitilir. Bu yöntemle 100'lerce karar ağacı oluşturulur ve her bir karar ağacı bireysel olarak tahminde bulunur. Günün sonunda problem regresyonsa karar ağaçlarının tahminlerinin ortalaması problem sınıflandırma ise tahminler arasında en çok oy alan seçilir.

Rastgele Orman denetimli bir öğrenme algoritmasıdır. Rastgele orman, yüzlerce karar ağaçları ve sonra her bir karar ağacını farklı bir gözlem örneği üzerinde eğitir. Zaten adından da anlaşılacağı gibi, bir orman oluşturur ve bunu bir şekilde rastgele yapar. Kurduğu “orman”, çoğu zaman “torbalama (bagging) ” yöntemiyle eğitilen karar ağaçları topluluğudur. Rastgele orman, birden fazla karar ağacını oluşturur ve daha doğru ve istikrarlı bir tahmin elde etmek için onları birleştirir. Rastgele ormanın büyük bir avantajı, mevcut makine öğrenmesi sistemlerinin çoğunu oluşturan hem sınıflandırma hem de regresyon problemleri için kullanılabilmesidir.

Rastgele orman, farklı ağaçlardan tahmin sonuçlarının ortalamasını alarak tek bir karar ağacından daha yüksek bir tahmin doğruluğu verir.

Rastgele orman algoritması, veri kümesinde önemli olan özelliklerin bulunmasına yardımcı olabilir. Bir veri kümesindeki önemli özellikleri seçen Boruta algoritmasının temelinde yer alır. Rastgele orman, çeşitli uygulamalarda, örneğin e-ticarette müşterilere farklı ürünlerin tavsiyelerini sağlamak için kullanılır. Tıpta, hastanın tıbbi kaydını analiz ederek hastanın hastalığını tanımlamak için rastgele bir orman algoritması kullanılabilir. Ayrıca bankacılık sektöründe müşterinin dolandırıcı veya meşru olup olmadığını kolayca tespit etmek için kullanılabilir.

Rastgele orman algoritması aşağıdaki adımları tamamlayarak çalışır:

Adım 1 : Algoritma, sağlanan veri kümesinden rastgele örnekler seçer.

Adım 2: Algoritma, seçilen her örnek için bir karar ağacı oluşturur. Ardından oluşturulan her karar ağacından bir tahmin sonucu alır.

Adım 3: Daha sonra tahmin edilen her sonuç için seçim yapılacaktır. Bir sınıflandırma problemind regresyon problemi için ortalama kullanacaktır .

Adım 4 : Son olarak, algoritma, son tahmin olarak en çok oylanan tahmin sonucunu seçecektir.

Rastgele Orman, ağaçları büyütürken, modele ek rasgelelik katıyor. Bir düğümü parçalara ayırırken en önemli özelliği aramak yerine, rastgele bir özellik alt kümesi arasında en iyi özelliği arar. Bu, genellikle daha iyi bir modelle sonuçlanan geniş bir çeşitlilikle sonuçlanır.

Bu nedenle, Rastgele Ormanda, bir düğümün bölünmesi için algoritma tarafından özelliklerin sadece rastgele bir alt kümesi dikkate alınır. Hatta mümkün olan en iyi eşikleri (normal bir karar ağacı gibi) aramak yerine, her özellik için rastgele eşikler kullanılarak ağaçlar daha rastgele yapılabilir.

Bir tatil gezisine karar vermek isteyen bir adam düşünün. Arkadaşlarına geçmişte gittiğini yerle ilgili kendisine sorular sormalarını ve tavsiyede bulunmasını ister. Arkadaşları, sorduklara soruların cevaplarını kullanarak ne tavsiye edilmesi gerektiğine dair kararını yönlendirmek için kurallar yaratırlar. Ardından arkadaşlarından daha fazla tavsiyede bulunmasını tavsiye eder ve ona farklı sorular sormalarını ister. Daha sonra, kendisine en çok tavsiye edilen yeri seçer. Bu, Rastgele Orman algoritması yaklaşımının tipik bir örneğidir.

Örneğin bir film izlemek istiyorsunuz ve karar veremiyorsunuz. Bir arkadaşınızı ararsanız ve o size tercih ettiğiniz film türü, süre, yıl, oyuncu-yönetmen, hollywood-alternatif vs. soru setinden çeşitli sorularla daha önce izlediğiniz filmlere (training set) göre bir tahminde bulunursa bu karar ağacı olur. Eğer 20 arkadaşınız bu soru setinden farklı sorular seçip verdiniz cevaplara göre tavsiyede bulunursa ve siz en çok tavsiye edilen filmi seçerseniz bu rassal orman olur.

Rastgele Orman algoritmasının bir başka büyük kalitesi, her bir özelliğin tahmin üzerindeki nispi önemini ölçmenin çok kolay olmasıdır. Sklearn, bu özelliği kullanan ağaç düğümlerinin, ormandaki tüm ağaçlar arasındaki kirliliği ne kadar azalttığına bakarak, bir özelliği ön planda tutan harika bir araç sağlar. Her bir özellik için bu skoru otomatik olarak eğitir ve sonuçları ölçeklendirir, böylece tüm önemlerin toplamı 1'e eşittir.

Bir karar ağacında her iç düğüm bir öznitelik üzerinde bir "testi" temsil eder (örn. bir para yatırma işleminin başa ya da kuyruklara mı dönüştüğü), her bir dal testin sonucunu temsil eder ve her bir yaprak düğümü bir sınıf etiketini temsil eder (tüm nitelikleri hesapladıktan sonra alınan karar). Çocukları olmayan bir düğüm bir yapraktır.

Özelliğin önemine bakarak, hangi özellikleri düşürmek istediğinize karar verebilirsiniz, çünkü bunlar tahmin sürecine yeterince katkıda bulunmaz. Bu önemlidir, çünkü makine öğreniminde genel bir kural, sahip olduğunuz daha fazla özellik olması, modelinizin aşırı uydurma ve tersi olma olasılığı daha yüksektir.

Karar Ağaçları ve Rastgele Ormanlar Arasındaki Fark:

Bir karar ağacında özelliklere ve etiketlere sahip bir eğitim veri seti girdiyseniz, tahminleri yapmak için kullanılacak bazı kurallar formüle edilir.

Örneğin, bir kişinin bir çevrimiçi reklama tıklayacağını tahmin etmek istiyorsanız, reklamı geçmişte tıklanan ve kararını açıklayan bazı özellikleri toplayabilirsiniz. Özellikleri ve etiketleri bir karar ağacına koyarsanız, bazı kurallar oluşturur. Sonra reklamın tıklanıp tıklanmayacağını tahmin edebilirsiniz. Buna karşılık, Rastgele Orman algoritması çeşitli karar ağaçları inşa etmek için gözlemleri ve özellikleri rastgele seçer ve sonuçların ortalamasını alır.

Bir başka farklılık da, “derin” karar ağaçlarının aşırı ısınmaktan muzdarip olmasıdır. Rastgele Orman, özelliklerin rastgele alt kümeleri oluşturarak ve bu alt kümeleri kullanarak daha küçük ağaçlar oluşturarak, çoğu zaman aşırı uydurmayı(overfitting) engeller. Daha sonra alt ağaçları birleştirir. Bunun her zaman çalışmadığını ve aynı zamanda rastgele ormanın kaç ağaç oluşturduğuna bağlı olarak hesaplamayı yavaşlattığını unutmayın.

1. Öngörücü Gücün Artırılması(Estimator)

En yüksek oyu almadan ya da tahminlerin ortalamalarını almadan önce algoritmanın oluşturduğu ağaç sayısı önemli bir parametredir. Genel olarak, daha yüksek sayıda ağaç performansı artırır ve tahminleri daha kararlı hale getirir, fakat aynı zamanda hesaplamayı da yavaşlatır.

Bir başka önemli parametre “max_features”, yani Random Forest’in tek bir ağaçta denemesine izin verilen maksimum özellik sayısıdır. Sklearn belgelerinde açıklanan çeşitli seçenekler sunar.

Hız açısından konuşacağımız son önemli parametre, “min_sample_leaf” dir. Bu, adından da anlaşılacağı gibi, bir iç düğümü ayırmak için gereken minimum yaprak sayısını belirler.

2. Model Hızını Artırma

“N_jobs” hiperparametre, motora kaç tane işlemcinin kullanılmasına izin verildiğini söyler. 1 değeri varsa, yalnızca bir işlemci kullanabilir. “-1” değeri, sınır olmadığını gösterir.

“Random_state”, modelin çıkışını çoğaltılamaz hale getirir. Model, her zaman rastgele bir random_state değeri olduğunda ve aynı parametreler ve aynı eğitim verisi verilmişse, aynı sonuçları üretecektir.

Son olarak, rastgele bir ormanda çapraz doğrulama yöntemi olan “oob_score” (aynı zamanda oob örnekleme olarak da adlandırılır) vardır. Bu örneklemede, verinin yaklaşık üçte biri modeli eğitmek için kullanılmamıştır ve performansını değerlendirmek için kullanılabilir. Bu numunelere bağ dışı örnekler denir. leave-one-cross-validasyon yöntemine çok benzer, ancak neredeyse hiçbir ek hesaplama yükü de beraberinde gelmiyor.

Avantajlar ve dezavantajlar:

Rassal orman modelinde farklı veri setleri üzerinde eğitim gerçekleştiği için varyans, diğer bir deyişle karar ağaçlarının en büyük problemlerinden olan overfitting azalır. Ayrıca bootstrap yöntemiyle oluşturulan alt-veri kümelerinde outlier bulunma şansını da düşürülmüş olunur. Random forest modelinin diğer bir özelliği de özniteliklerin ne kadar önemli olduğunun belirtilmesidir. Bir özniteliğin önemli olması demek o özniteliğin bağımlı değişkendeki varyansın açıklanmasına ne kadar katkı yaptığıyla alakalıdır. Random forest algoritmasında x

sayıda öznitelik verilip en faydalı y tanesinin seçilmesi istenebilir ve istenirse bu bilgi istenen başka bir modelde kullanabilir.

Rastgele Ormanların bir avantajı hem regresyon hem de sınıflandırma görevleri için kullanılabileceği ve girdi özelliklerine verdiği göreceli önemi görmenin kolay olmasıdır.

Rastgele Orman ayrıca çok kullanışlı ve kullanımı kolay bir algoritma olarak kabul edilir, çünkü varsayılan hiperparametreler genellikle iyi bir tahmin sonucu oluşturur.

Hiperparametre sayısı da o kadar yüksek değildir ve anlaşılması kolaydır.

Rastgele Ormanın ana sınırlaması, çok sayıda ağacın algoritmayı gerçek zamanlı tahminler için yavaş ve etkisiz hale getirebilmesidir. Genel olarak, bu algoritmalar hızlı bir şekilde eğitilebilir, ancak bir kez eğitildiklerinde tahminler oluşturmak için oldukça yavaştır. Daha doğru bir tahmin, daha yavaş bir modelle sonuçlanan daha fazla ağaç gerektirir. Gerçek dünyadaki uygulamaların çoğunda Rastgele Orman algoritması yeterince hızlıdır, ancak çalışma zamanı performansının önemli olduğu ve diğer yaklaşımların tercih edilebileceği durumlar olabilir.

Ve tabii ki Rastgele Orman, prediktif bir modelleme aracıdır ve açıklayıcı bir araç değildir. Bu, verilerinizdeki ilişkilerin bir tanımını arıyorsanız, başka yaklaşımların tercih edileceği anlamına gelir.

Rastgele Orman algoritması, Bankacılık, Borsa, Tıp ve E-Ticaret gibi birçok farklı alanda kullanılmaktadır. Bankacılıkta, örneğin bankanın hizmetlerini diğerlerinden daha sık kullanacak müşterileri tespit etmek ve borçlarını zaman içerisinde geri ödemek için kullanılır. Bu alanda bankayı dolandırmak isteyen dolandırıcılık müşterilerini tespit etmek için de kullanılır. Finansmanda, bir hissenin gelecekteki davranışını belirlemek için kullanılır. Sağlık alanında, tıptaki bileşenlerin doğru kombinasyonunu tanımlamak ve hastalıkları tanımlamak için bir hastanın tıbbi geçmişini analiz etmek için kullanılır. Ve son olarak, E-ticaret'te Rastgele Orman, bir müşterinin ürünü gerçekten beğenip beğenmeyeceğini belirlemek için kullanılır.

Rastgele Orman, model geliştirme sürecinde erken eğitim için ve nasıl performans gösterdiğini görmek için basit bir algoritmadır ve sadeliği nedeniyle “kötü” bir Rastgele Orman oluşturmak zordur. Kısa bir süre içinde bir model geliştirmeniz gerekiyorsa, bu algoritma da mükemmel bir seçimdir. Bunun üzerine, özelliklerine verdiğiniz önemin oldukça iyi bir göstergesidir.

Rastgele Ormanlar da performans açısından yenmek için çok zor. Elbette, sinir ağları gibi her zaman daha iyi performans gösterebilecek bir model bulabilirsin, ama bunlar genellikle

gelişimde daha fazla zaman alırlar. Üstelik, ikili, kategorik ve sayısal gibi bir çok farklı özellik tipini de kullanabilirler.

Genel olarak, Rastgele Orman, sınırlamaları olmasına rağmen (çoğunlukla) hızlı, basit ve esnek bir araçtır.

Modeli uygulama süreci:

- 1) Sütunlar isimlendirilir veri tipleri ve eksik veri olup olmadığı kontrol edilir. Modeldeki hedef değişkenin dağılımına bakılır ve veri setinin geri kalanından ayrılır.
- 2) Veri tipi 'object' olan sütunlar seçilir. Bu sütunlar yine o sütunun mode() değeriyle doldurulur.
- 3) Makine öğrenmesi modelleri kategorik değişkenleri algılayamadığı için 'object' tipindeki değişkenleri one-hot-encoding yöntemiyle 0 ve 1'lere ayrılır.
- 4) Modeli kurulmaya hazır şimdi standart modelleme süreçleri uygulanır.
- 5) Veriyi eğitim ve test alt-veri gruplarına ayrılır.
- 6) Karar ağacı modeli oluşturulur.
- 7) Modeli eğitim verisi 'fit' edilir.
- 8) Görülmeyen test verisi modele verilir tahminde bulunulur.
- 9) Modelin başarı metrikleri: Confusion matrix, Modelin başarı metrikleri: Precision, recall, f1-score.
- 10) Karar ağaçlarından birini görselleştirir.
- 11) Modelin özneliliklerinin önem sıralama analiz edilir.

Veri tiplerini kontrol etme/düzeltilme:

- Açıklayıcı veri analizi ve görselleştirme.
- Eksik verileri tahmin etme/veri atama.
- Kategori tipindeki verileri one-hot encoding ile nümerik formata çevirme.
- Veri setini eğitim ve test veri-setlerine ayırma.
- Modeli eğitme ve test verisi üzerinde tahmin yapma.
- Sınıflandırma başarı metriklerine bakma.
- Karar ağacını görselleştirme.
- Modelin yaptığı öznelilik sıralamasını görselleştirme.

İlişkisiz sonuçların neden bu kadar büyük olduğuna dair bir örnek:

Bir sayı üretmek için tekdüze dağıtılmış rasgele sayı üretici kullanılsın. Oluşturulan sayı 40'tan büyük veya ona eşitse, kazanılsın (yani% 60 zafer şansınız olur) ve size biraz para ödensin. 40'ın altındaysa kaybedilsin, kaybeden aynı miktarı ödersin. Şimdi size aşağıdaki seçenekler sunulmaktadır. Şunlardan birini yapabilirsiniz: Oyun 1 - 100 kez oynayın, her seferinde 1 \$ bahis yapın.

Oyun 2 - 10 kez oynayın, her seferinde 10 \$ bahis yapın.

Oyun 3 - bir kez oynayarak 100 \$ bahis yapın.

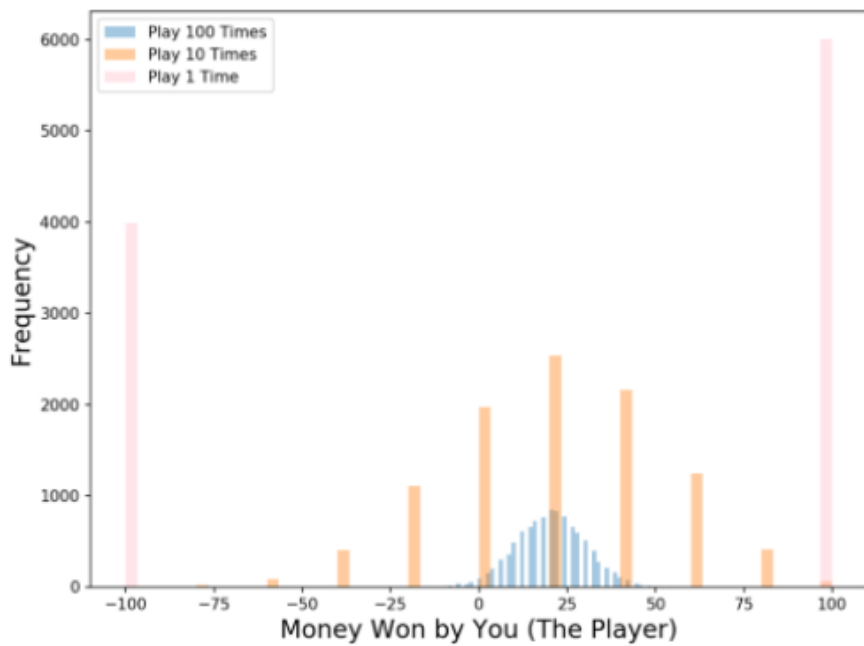
Hangisini seçerdin? Her oyunun beklenen değeri aynıdır:

Beklenen Değer Oyunu 1 = $(0.60 * 1 + 0.40 * -1) * 100 = 20$

Beklenen Değer Oyunu 2 = $(0.60 * 10 + 0.40 * -10) * 10 = 20$

Beklenen Değer Oyunu 3 = $0.60 * 100 + 0.40 * -100 = 20$

Dağılımlar ne olacak? Sonuçları bir Monte Carlo simülasyonu ile görselleştirildiğinde (her oyun türü için 10.000 simülasyon çalıştırılacak; örneğin, 1. Oyundaki 100 oyunun 10.000 katı simüle edilecek). Aşağıdaki tabloya bir göz atın - şimdi hangi oyunu seçerdiniz? Beklenen değerler aynı olsa bile, sonuç dağılımları, pozitif ve dardan (mavi) ikiliye (pembe) doğru büyük ölçüde farklıdır.



Outcome Distribution of 10,000 Simulations for each Game

Oyun 1 (100 kez oynadığımız yer), biraz para kazanmak için en iyi şans sunuyor - yürüttüğüm 10.000 simülasyondan% 97'sinde para kazanıyorsunuz! Oyun 2'de (10 kez oynadığımız yerde) simülasyonların% 63'ünde para kazanırsınız, ciddi bir düşüş (ve para kaybetme olasılığınızda ciddi bir artış). Ve sadece bir kez oynadığımız 3. Oyun, beklendiği gibi simülasyonların% 60'ında para kazanıyorsunuz.

Dolayısıyla, oyunlar aynı beklenen değeri paylaşırsa da, sonuç dağılımları tamamen farklıdır. 100 \$ 'lık bahsimizi farklı oyunlara ne kadar çok bölersek, para kazanacağımıza o kadar güvenebiliriz. Daha önce de belirtildiği gibi, bu işe yarar çünkü her oyun diğerlerinden bağımsızdır.

Rastgele orman aynıdır - her ağaç, önceki oyunumuzdaki bir oyun gibidir. Daha fazla oynadığımızda para kazanma şansımızın nasıl arttığını gördük. Benzer şekilde, rastgele bir orman modeliyle, modelimizdeki ilişkisiz ağaçların sayısı ile doğru tahminler yapma şansımız artar.

Modellerin birbirini çeşitlendirmesini sağlamak:

Öyleyse rastgele orman, her bir ağacın davranışının modeldeki diğer ağaçlardan herhangi birinin davranışıyla çok fazla ilişkili olmamasını nasıl sağlar?

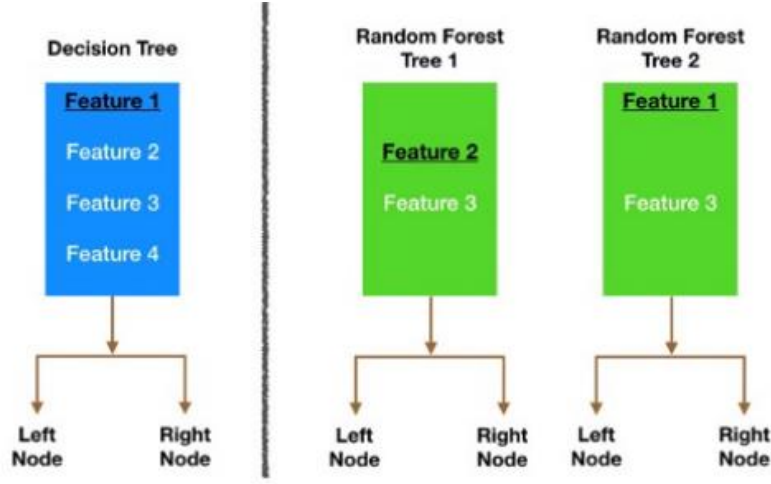
Aşağıdaki iki yöntemi kullanır:

Torbalama (Bootstrap Aggregation) - Karar ağaçları, eğitildikleri verilere karşı çok hassastır - eğitim setinde yapılan küçük değişiklikler, önemli ölçüde farklı ağaç yapılarına neden olabilir. Rastgele orman, her bir ağacın veri kümesinden değiştirilerek rasgele örneklemesine izin vererek bundan yararlanır ve farklı ağaçlarla sonuçlanır. Bu işlem torbalama olarak bilinir.

Torbalama ile eğitim verilerini daha küçük parçalara ayırmadığımıza ve her ağacı farklı bir yığın üzerinde eğitmediğimize dikkat edin. Bunun yerine, N büyüklüğünde bir örneğimiz varsa, yine de her ağaca N boyutunda bir eğitim seti besliyoruz (aksi belirtilmedikçe). Ancak orijinal eğitim verileri yerine, değiştirilmiş N boyutunda rastgele bir örnek alıyoruz. Örneğin, eğitim verilerimiz [1, 2, 3, 4, 5, 6] ise, ağaçlarımızdan birine aşağıdaki listeyi verebiliriz [1, 2, 2, 3, 6, 6]. Her iki listenin de altı uzunluğunda olduğuna ve "2" ile "6" nın ikisinin de ağacımıza verdiğimiz rastgele seçilmiş eğitim verilerinde tekrarlandığına dikkat edin (çünkü değiştirme ile örnekleme yapıyoruz).

Özellik Rastgeleliği:

Normal bir karar ağacında, bir düğümü bölme zamanı geldiğinde, mümkün olan her özelliği göz önünde bulundururuz ve sağ düğümdekilerle sol düğümdeki gözlemler arasında en fazla ayrımı yaratanı seçeriz. Bunun aksine, rastgele bir ormandaki her ağaç yalnızca rastgele bir özellik alt kümesinden seçim yapabilir. Bu, modeldeki ağaçlar arasında daha fazla çeşitliliği zorlar ve sonuçta ağaçlarda daha düşük korelasyon ve daha fazla çeşitlilik ile sonuçlanır.



Node splitting in a random forest model is based on a random subset of features for each tree.

Görsel bir örnek üzerinden geçelim - yukarıdaki resimde, geleneksel karar ağacı (mavi), düğümü nasıl böleceğine karar verirken dört özelliğin tümünden seçim yapabilir. Verileri olabildiğince ayrılmış gruplara böldüğü için Özellik 1 (siyah ve altı çizili) ile devam etmeye karar verir. Şimdi rastgele ormanımıza bir göz atalım. Bu örnekte ormanın iki ağacını inceleyeceğiz. Rastgele Orman Ağacı 1'i kontrol ettiğimizde, düğüm bölme kararı için yalnızca Özellik 2 ve 3'ü (rastgele seçilir) dikkate alabileceğini görürüz. Geleneksel karar ağacımızdan (mavi olarak) Özelliğin bölme için en iyi özellik olduğunu biliyoruz, ancak Ağaç 1, Özellik 1'i göremediğinden, Özellik 2'ye (siyah ve altı çizili) gitmek zorunda kalır. Öte yandan Ağaç 2, yalnızca Özellik 1 ve 3'ü görebilir, bu nedenle Özellik 1'i seçebilir. Dolayısıyla rastgele ormanımızda, yalnızca farklı veri kümeleri üzerinde eğitilen (torbalama sayesinde) değil, aynı zamanda karar vermek için farklı özellikler kullanan ağaçlarla karşılaşırız. Ve bu, sevgili okuyucum, birbirlerini tamponlayan ve hatalarından koruyan ilişkisiz ağaçlar yaratır.

Rastgele orman, birçok karar ağacından oluşan bir sınıflandırma algoritmasıdır. Komite tarafından tahmin edilmesi herhangi bir ağaçtan daha doğru olan ilişkisiz bir ağaç ormanı yaratmaya çalışmak için her bir ağacı inşa ederken torbalama kullanır ve rastgelelik özelliğini kullanır. Rastgele ormanımızın doğru sınıf tahminleri yapabilmesi için neye ihtiyacımız var? En azından bir miktar tahmin gücüne sahip özelliklere ihtiyacımız var. Sonuçta, eğer içine çöp koyarsak, o zaman çöpü dışarı atarız. Ormanın ağaçları ve daha da önemlisi tahminlerinin ilintisiz olması (veya en azından birbirleriyle düşük korelasyonlara sahip olması) gerekir. Algoritmanın kendisi özellik rastgeleliği aracılığıyla bu düşük korelasyonları bizim için tasarlamaya çalışırken, seçtiğimiz özellikler ve seçtiğimiz hiper parametreler nihai korelasyonları da etkileyecektir.

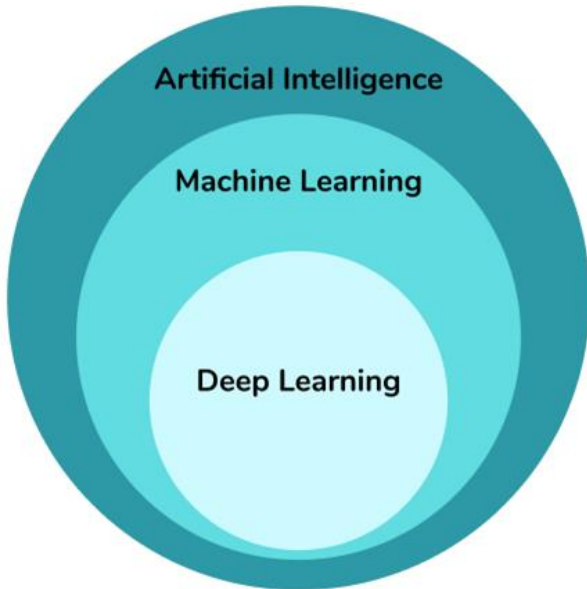
6. Derin Öğrenme Algoritmaları

Derin öğrenme, tamamen yapay sinir ağlarına dayanan bir makine öğrenimi alt kümesidir. Sinir ağları insan beynini taklit edecek şekilde tasarlandığından, derin öğrenme de aynı şekilde bir insan beyni taklitçisidir.

Derin öğrenmede her şeyi açıkça programlamak zorunda değiliz. Bir modeli eğitim veri kümesi üzerinde eğitiriz ve model, test ve doğrulama veri kümesinde de neredeyse doğru tahminde bulunana kadar onu doğaçlama yaparız.

Derin öğrenme modelleri, programcıdan yalnızca küçük bir girdi ile kendi başlarına doğru özelliklere odaklanabilir ve boyutsallık sorununu çözmede oldukça faydalıdır. Derin öğrenme yeni bir kavram değildir. Bir süredir ortalıkta dolaşıyor. Şu anda sahip olduğumuz kadar fazla işlem gücüne veya veriye sahip olmadığımız için bu günlerde tüm işlem gücü son 20 yılda muazzam bir şekilde arttığından, sahneye derin öğrenme ve makine öğrenimi girdi.

1960'ların ortalarında bir derin öğrenme ağı üzerinde çalışırken, Alexey Grigorevich Ivakhnenko ilk yayını yayınladı. Esasen, çok sayıda doğrusal olmayan işlem birimi kullanarak özellik çıkarma ve dönüştürme yapan bir makine öğrenimi alt kümesidir. Sonraki katmanların her biri, bir önceki katmanın çıktısını girdi olarak kullanır. Derin öğrenme, bilgisayarla görme, konuşma tanıma, doğal dil işleme vb. dahil olmak üzere çeşitli uygulamalar için uygundur.



Yukarıdaki görüntü, Makine Öğreniminin Yapay Zekanın bir alt kümesi olduğunu ve Derin Öğrenmenin, Makine Öğreniminin bir alt kümesi olduğunu göstermektedir.

Derin öğrenme (aynı zamanda derin yapılandırılmış öğrenme, hiyerarşik öğrenme ya da derin makine öğrenmesi) bir veya daha fazla gizli katman içeren yapay sinir ağları ve benzeri makine öğrenme algoritmalarını kapsayan çalışma alanıdır. Yani en az bir adet yapay sinir ağının (YSA) kullanıldığı ve birçok algoritma ile, bilgisayarın eldeki verilerden yeni veriler elde etmesidir. Derin öğrenme gözetimli, yarı gözetimli veya gözetimsiz olarak gerçekleştirilebilir. Derin yapay sinir ağları pekiştirmeli öğrenme yaklaşımıyla da başarılı sonuçlar vermiştir.

Derin öğrenme, Yüksek bilgi işleme gücü ve büyük veri kümeleriyle birlikte katmanlı yapay sinir ağlarının güçlü matematiksel modelleri oluşturabildiği bir makine öğrenimi alt kümesidir. **Verinin yapısına göre hangi parametrelere ne ağırlık verileceğini kendisi keşfetmektedir.** Derin öğrenme, ham girdiden daha yüksek seviyedeki özellikleri aşamalı olarak çıkarmak için birden çok katman kullanan bir makine öğrenme algoritmaları sınıfıdır.

Modern derin öğrenme modellerinin çoğu, yapay sinir ağlarına, özellikle de Konvolüsyonel Sinir Ağlarına (CNN - Convolutional Neural Networks) dayanmaktadır, ancak aynı zamanda derin düşünce ağları ve derin düğümler gibi derin üretken modellerde katmansal olarak düzenlenmiş öneri formülleri veya gizli değişkenleri de içerebilirler. Yapay sinir ağları (YSA) biyolojik sistemlerde bilgi işleme ve dağıtılmış iletişim düğümlerinden esinlenmiştir. YSA'ların biyolojik beyinlerden çeşitli farklılıkları vardır. Özellikle, sinir ağları statik ve sembolik olma eğilimindeyken, çoğu canlı organizmanın biyolojik beyni dinamik (plastik) ve analogdur.

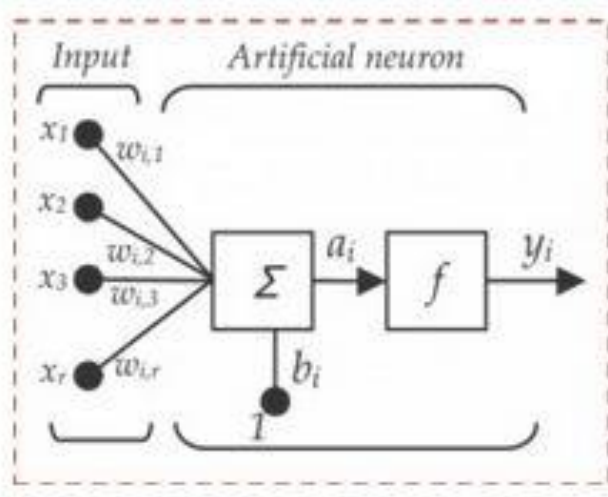
Derin öğrenme, her seviye girdi verilerini biraz daha soyut ve bileşik bir temsile dönüştürmeyi öğrenir. Bir görüntü tanıma uygulamasında, ham girdi bir piksel matrisi olabilir; birinci temsili katman pikselleri soyutlayabilir ve kenarları kodlayabilir; ikinci katman kenar düzenlemelerini oluşturabilir ve kodlayabilir; üçüncü katman bir burnu ve gözleri kodlayabilir; ve dördüncü katman görüntünün bir yüz içerdiğini tanıyabilir. Daha da önemlisi, derin bir öğrenme süreci hangi özelliklerin hangi seviyeye en uygun şekilde yerleştirileceğini öğrenebilir. Elbette bu, elle ayarlama ihtiyacını tamamen ortadan kaldırmaz; örneğin, değişen sayıda katman ve katman boyutu farklı derecelerde soyutlama sağlayabilir.

Derin öğrenme, Yapay Zekada (AI) Makine Öğreniminin bir alt kümesidir. Bir yapay zeka, verileri işlemek, kalıplar oluşturmak ve kararlar almak için insan beynini taklit ettiğinde, bu Derin Öğrenmedir. Ham girdiden aşamalı olarak üst düzey özellikleri çıkarmak için birden çok katman kullanır. "Derin öğrenmedeki" "derin" kelimesi, verilerin dönüştürüldüğü katman sayısını ifade eder. Bu sayede yapılandırılmamış ve etiketsiz verilerden örüntüler öğrenmek mümkündür. Derin öğrenme algoritması, deneyimlerden nasıl öğrendiğimize benzer şekilde, her seferinde sonucunu iyileştiren bir görevi tekrar tekrar gerçekleştirir.

Arama motorlarında, sosyal medyada, e-ticaret platformlarında her saniye yaratılan muazzam miktarda veri, derin öğrenmeyi büyük bir potansiyele dönüştürdü. Buna ek olarak, bugün mevcut olan güçlü bilgi işlem gücü, derin öğrenmede kullanılan algoritmalar üzerinde

büyük bir etki yarattı. Dahası, kendi kendine giden arabalar, AlphaGo, sesli asistanlar gibi yapay zekadaki atılımlar, derin öğrenme sayesinde mümkün.

Derin öğrenmede, her seviye girdi verilerini biraz daha soyut ve bileşik bir temsile dönüştürmeyi öğrenir. Bir görüntü tanıma uygulamasında, ham girdi bir piksel matrisi olabilir; birinci temsil katmanı pikselleri soyutlayabilir ve kenarları kodlayabilir; ikinci katman, kenar düzenlemelerini oluşturabilir ve kodlayabilir; üçüncü katman bir burnu ve gözleri kodlayabilir; ve dördüncü katman, görüntünün bir yüz içerdiğini fark edebilir. Daha da önemlisi, derin öğrenme süreci hangi özelliklerin hangi seviyeye en uygun şekilde yerleştirileceğini kendi başına öğrenebilir. (Elbette bu, elle ayarlama ihtiyacını tamamen ortadan kaldırmaz; örneğin, değişen sayıda katman ve katman boyutu, farklı soyutlama dereceleri sağlayabilir.)



Nöronun farklı bileşenleri şu şekilde belirtilir:

- $x_1, x_2, \dots, x_N$: Bunlar nöronun girdileridir. Bunlar, giriş katmanındaki gerçek gözlemler veya gizli katmanlardan birinin ara değeri olabilir.
- $w_1, w_2, \dots, w_N$: Her girişin ağırlığı.
- b_i : Eğilim ya da Sapma birimleri olarak adlandırılır. Bunlar, her ağırlığa karşılık gelen aktivasyon fonksiyonunun girişine eklenen sabit değerlerdir. Kesişme terimine benzer şekilde çalışır.
- a : Şu şekilde temsil edilebilen nöronun aktivasyonu olarak adlandırılır
- ve y : nöronun çıktısıdır

$$a = f\left(\sum_{i=0}^N w_i x_i\right)$$

Derin Öğrenme Algoritmaları:

Gözetimli öğrenme

Kümeleme

Boyut indirgeme

Yapılandırılmış tahmin

Anomali tespiti

Sinir ağları

Pekiştirmeli öğrenme

6.1.1. Anomali Tespiti

Veri analizinde, anomali tespiti (aynı zamanda aykırı değer tespiti), verilerin çoğunluğundan önemli ölçüde farklılaşarak şüphe uyandıran nadir öğelerin, olayların veya gözlemlerin tanımlanmasıdır. Tipik olarak anormal öğeler, banka dolandırıcılığı, yapısal bir kusur, tıbbi sorunlar veya bir metindeki hatalar gibi bir tür soruna dönüşecektir. Anormallikler ayrıca aykırı değerler, yenilikler, gürültü, sapmalar ve istisnalar olarak da adlandırılmaktadır.

Özellikle, kötüye kullanım ve ağa izinsiz giriş tespiti bağlamında, ilginç nesneler genellikle nadir nesneler değil, beklenmedik etkinlik patlamalarıdır. Bu model, bir aykırı değer nadir bir nesne olarak genel istatistiksel tanımına uymaz ve uygun şekilde bir araya getirilmediği sürece birçok aykırı değer algılama yöntemi (özellikle denetimsiz yöntemler) bu tür verilerde başarısız olmaktadır. Bunun yerine, bir küme analizi algoritması, bu modellerin oluşturduğu mikro kümeleri tespit edebilmektedir.

Üç geniş anomali tespit tekniği kategorisi mevcuttur[4]. Denetimsiz anomali tespit teknikleri, veri setindeki örneklerin çoğunluğunun normal olduğu varsayımı altında, veri setinin geri kalanına en az uyan örnekleri arayarak etiketlenmemiş bir test veri setindeki anormallikleri tespit etmektedir. Denetimli anomali tespit teknikleri, "normal" ve "anormal" olarak etiketlenmiş bir veri seti gerektirir ve bir sınıflandırıcının eğitimini içermektedir (diğer birçok istatistiksel sınıflandırma probleminden temel fark, aykırı değer tespitinin doğal dengesiz doğasıdır). Yarı denetimli anomali tespit teknikleri, belirli bir normal eğitim veri setinden normal davranışı temsil eden bir model oluşturur ve ardından kullanılan model tarafından bir test örneğinin oluşturulma olasılığını test etmektedir.

6.1.2. Yapay Sinir Ağları

İnsan beyni:

İnsan beyninin nasıl çalıştığını daha yeni anlamaya başladık ; daha yolun başındayız. Akıllı makineler yapmanın bir yolu, insan beynini özellikle organizasyonel prensiplerini taklit etmeye çalışmaktır.

Beyin son derece karmaşık, doğrusal olmayan ve paralel bir bilgisayardır ve yoğun şekilde bağlı 10^{11} nörondan oluşur (nöron başına $\sim 10^4$ bağlantı). Bir nöron, silikon mantık geçidine (10^{-9} sn) kıyasla çok daha yavaştır (10^{-3} sn), doğrudur, nöronlar arasındaki muazzam bağlantı, nispeten yavaş hızı oluşturur. Ancak, *karmaşık algısal kararlara çok hızlı bir şekilde ulaşılır* (birkaç yüz milisaniye içinde)

100 Adım kuralı: Bireysel nöronlar birkaç milisaniye içinde çalıştığından, hesaplamalar yaklaşık 100'den fazla seri adım içermez ve bir nörondan diğerine gönderilen bilgi çok küçüktür (birkaç bit)

Plastisite: Beynin nöral yapısının bir kısmı doğumda bulunurken, diğer kısımları çevreye uyum sağlamak için özellikle yaşamın erken dönemlerinde öğrenme yoluyla geliştirilir (yeni girdiler). Biyolojik Nöronlarda, farklı dallanma yapılarına sahip çeşitli farklı nöronlar (motor nöron, merkez üstü çevresel olmayan görsel hücreler...) mevcuttur. Ağın bağlantıları ve tek tek sinapsların güçleri ağın işlevini oluşturur.

Beyindeki Biyolojik Nöronlar:

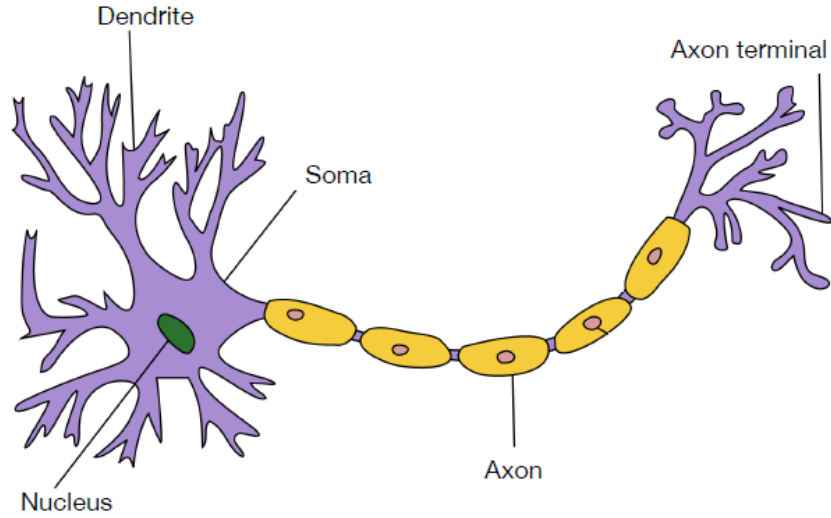
İnsan beyni yaklaşık 100 milyar nörondan oluşur.

dendritler: hücrelere elektrik sinyalleri taşıyan sinir lifleri

hücre gövdesi: girdilerinin doğrusal olmayan bir işlevini hesaplar

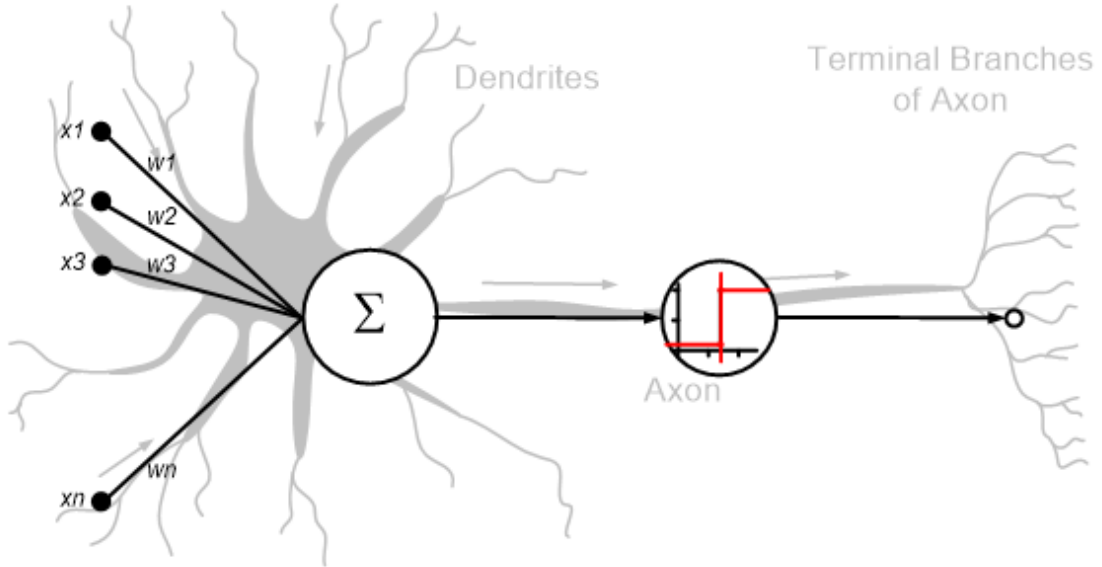
akson: elektrik sinyalini hücre gövdesinden diğer nöronlara taşıyan tek uzun lif. Nöronun aksonu, diğer birçok nöronun dendritlerine bağlıdır. Nöronlar, dendritlerden elektrik sinyalleri alır ve bunları aksone gönderir.

sinaps: gücü hücreye girişi etkileyen kimyasal bir bağlantıyı düzenleyen, bir hücrenin aksonu ile diğerinin dendriti arasındaki temas noktası.



İnsan beyninden ilham alan hesaplama model, Yapay Sinir Ağları:

- Basit işlem birimlerinden (nöronlar) oluşan büyük ölçüde paralel, dağıtılmış sistem
- Edinilen bilgiyi depolamak için nöronlar arasındaki sinaptik bağlantı güçleri kullanılır.
- Bilgi, ağ tarafından bir öğrenme süreci aracılığıyla çevresinden edinilir.



- Basit işlem birimlerinden (nöronlar) oluşan büyük ölçüde paralel, dağıtılmış sistem
- Edinilen bilgiyi depolamak için nöronlar arasındaki sinaptik bağlantı güçleri kullanılır.
- Bilgi, ağ tarafından bir öğrenme süreci aracılığıyla çevresinden edinilir.
- Örneklerden öğrenmek: etiketli veya etiketsiz
- Adaptivite: bir şeyler öğrenmek için bağlantı güçlerini değiştirmek
- Doğrusal olmama: doğrusal olmayan aktivasyon fonksiyonları gereklidir

- Hata toleransı: Nöronlardan veya bağlantılardan biri hasar görürse, tüm ağ hala oldukça iyi çalışıyor.
- Bu nedenle, aşağıdakilerle karakterize edilen sorunlar için klasik çözümlerden daha iyi alternatifler olabilirler: Yüksek boyutluluk, gürültü, kesin olmayan veya eksik veriler; ve açıkça ifade edilmiş matematiksel bir çözüm veya algoritmanın olmaması

Beyin ve Yapay Sinir Ağları:

Benzerlikler:

- Nöronlar, nöronlar arasındaki bağlantılar
- Öğrenme = bağlantıların değişmesi, nöronların değişmesi değil
- Büyük paralel işleme

Ancak yapay sinir ağları çok daha basit:

- nöron içindeki hesaplama büyük ölçüde basitleştirildi
- ayrık zaman adımları
- tipik olarak çok sayıda uyarı içeren bir tür denetimli öğrenim

Bir sinir ağı, basit işlem birimlerinden (yapay nöronlar) oluşan büyük ölçüde paralel, dağıtılmış bir işlemcidir. Beyne iki açıdan benziyor:

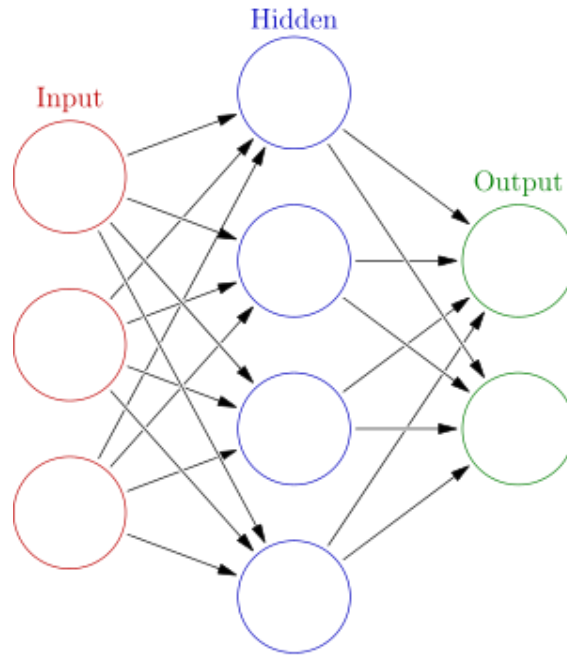
- Bilgi, ağ tarafından bir öğrenme süreci aracılığıyla çevresinden edinilir
- Edinilen bilgiyi depolamak için nöronlar arasındaki sinaptik bağlantı güçleri kullanılır.

Yapay sinir ağları (YSA) veya bağlantı sistemi, hayvan beyinlerini oluşturan biyolojik sinir ağlarından belirsiz bir şekilde esinlenen bilgisayar sistemleridir. Bu tür sistemler, genellikle herhangi bir göreve özgü kural ile programlanmadan, örnekleri dikkate alarak görevleri yerine getirmeyi "öğrenir".

YSA, biyolojik bir beyindeki nöronları gevşek bir şekilde modelleyen "yapay nöronlar" adı verilen bağlı birimler veya düğümler koleksiyonuna dayanan bir modeldir. Her bağlantı, biyolojik bir beyindeki sinapslar gibi, bir yapay nörondan diğerine bilgi, bir "sinyal" iletebilir. Bir sinyal alan yapay bir nöron, onu işleyebilir ve daha sonra ona bağlı ek yapay nöronları işaret edebilir. Ortak YSA uygulamalarında, yapay nöronlar arasındaki bağlantıdaki sinyal gerçek bir sayıdır ve her bir yapay nöronun çıkışı, girdilerinin toplamının doğrusal olmayan bir fonksiyonu tarafından hesaplanır. Yapay nöronlar arasındaki bağlantılara "kenarlar" denir. Yapay nöronlar ve kenarlar tipik olarak öğrenme ilerledikçe ayarlanan bir ağırlığa sahiptir. Ağırlık, bir bağlantıdaki sinyalin gücünü artırır veya azaltır. Yapay nöronlar, sadece toplam sinyal bu eşiği geçtiğinde sinyalin gönderileceği bir eşik değerine sahip olabilir. Tipik olarak, yapay nöronlar katmanlar halinde toplanır. Farklı katmanlar, girdilerinde farklı türde dönüşümler gerçekleştirebilir. Sinyaller, muhtemelen katmanları birden çok kez geçtikten sonra, ilk katmandan (giriş katmanı) son katmana (çıkış katmanı) gider.

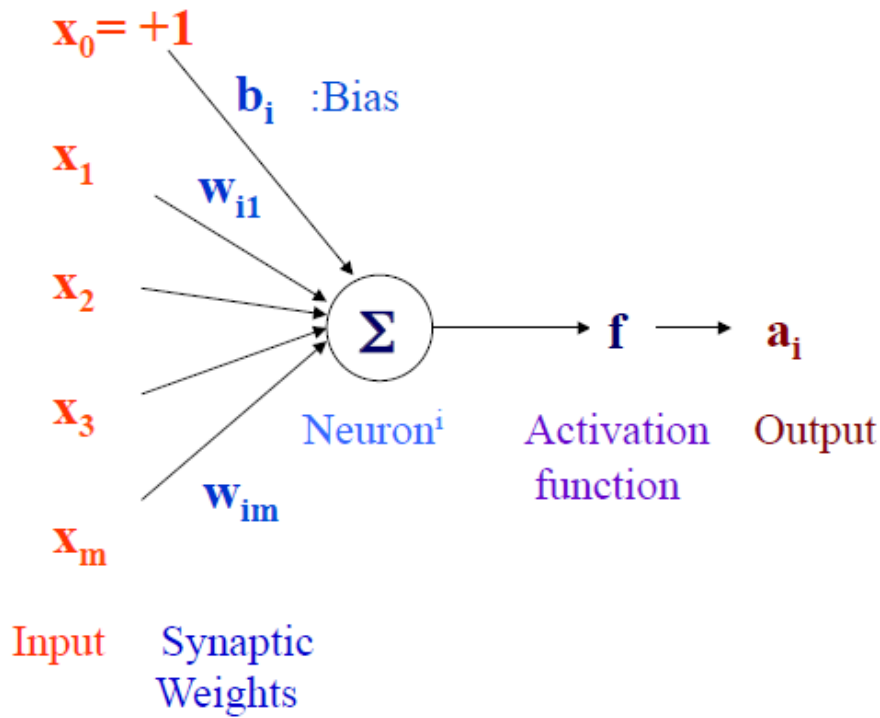
YSA yaklaşımının asıl amacı, problemleri bir insan beyinde olduğu gibi çözmektir. Bununla birlikte, zamanla dikkat, belirli görevlerin yerine getirilmesine ve biyolojiden sapmalara yol açtı. Yapay sinir ağları, bilgisayar görme, konuşma tanıma, makine çevirisi, sosyal ağ filtreleme, oyun tahtası ve video oyunları ve tıbbi teşhis gibi çeşitli görevlerde kullanılmıştır.

Derin öğrenme yapay bir sinir ağında birden fazla gizli katmandan oluşur. Bu yaklaşım, insan beyninin ışığı ve sesi görme ve işleme biçimini modellemeye çalışır. Derin öğrenmenin bazı başarılı uygulamaları bilgisayarla görme ve konuşma tanımadır.



Yapay bir sinir ağı, bir beyindeki geniş nöron ağına benzer şekilde birbirine bağlı bir düğüm grubudur. Burada, her dairesel düğüm bir yapay nöronu temsil eder ve bir ok, bir yapay nöronun çıkışından diğerinin girişine bir bağlantıyı temsil eder.

Yapay Nöron Modeli



Eğilim – Meyillenme (Bias):

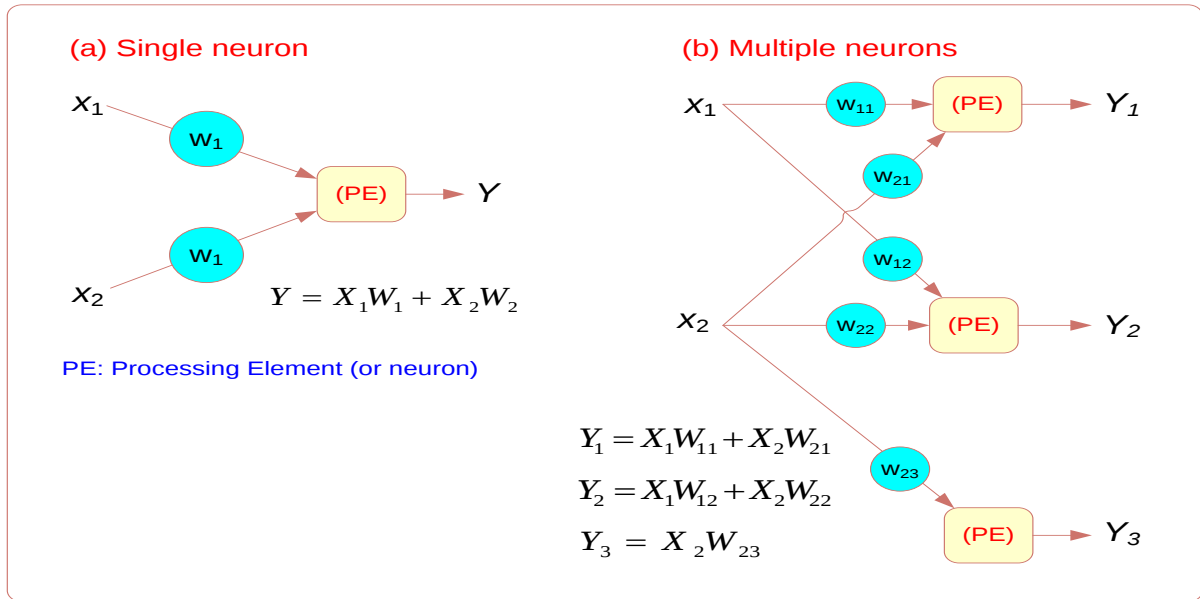
Bias: eşik değer, eğilim ya da sapma değeri.

Yapay bir nöron: Girdisinin ağırlıklı toplamını hesaplar (net girdisi denir). Sapmasını ekler. Bu değeri bir aktivasyon işlevinden geçirir. Nöronun çıkışı sıfırın üzerindeyse tetiklendiğini, "ateşlendiğini" (yani aktif hale geldiğini) söylüyoruz. Sapma, sabit +1.0 girdisine kenetlenen başka bir ağırlık olarak dahil edilebilir. Bu ekstra serbest değişken (eğilim ya da sapma), nöronu daha güçlü hale getirir.

$$a_i = f(n_i) = f\left(\sum_{j=1}^n w_{ij}x_j + b_i\right)$$

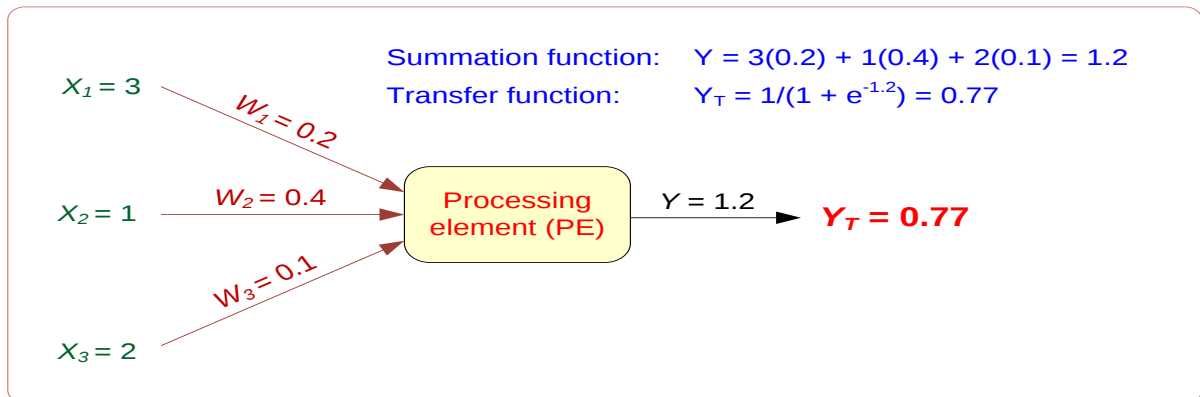
Yapay Sinir Ağları (YSA) Unsurları:

- İşleme ögesi (Processing element - PE)
- Ağ Mimarisi
 - Gömülü Katmanlar
 - Paralel İşlem
- Ağ Bilgisi İşleme
 - Girişler
 - Çıkışlar
 - Bağlantı Ağırlıkları (Connection weights)
 - Toplama İşlevi (Summation function)



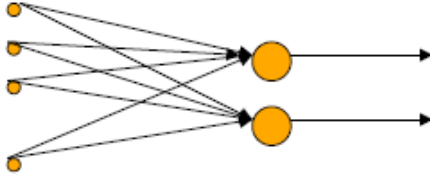
Transformation (Transfer) Function:

- Linear function
- Sigmoid (logical activation) function [0 1]
- Tangent Hyperbolic function [-1 1]



Farklı Ağ Topolojileri:

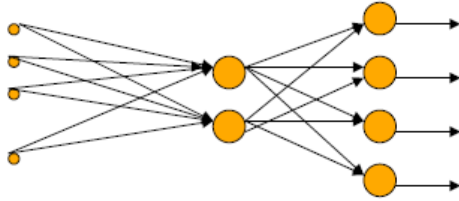
- **Tek katmanlı ileri beslemeli ağlar:** Çıktı katmanına yansıyan girdi katmanı.



Input
layer

Output
layer

- **Çok katmanlı ileri beslemeli ağlar:** Bir veya daha fazla gömülü katman. Girdi projeleri yalnızca önceki katmanlardan bir katmana yansıtılır. Tipik olarak, yalnızca bir katmandan diğerine bağlanırlar.



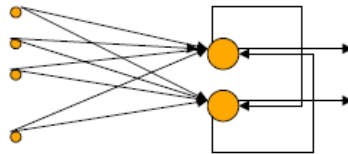
Input
layer

Hidden
layer

Output
layer

2-layer or
1-hidden layer
fully connected
network

- **Tekrarlayan ağlar:** Bazı girişlerinin bazı çıkışlarına (ayrık zaman) bağlı olduğu geri beslemeli bir ağ.



Input
layer

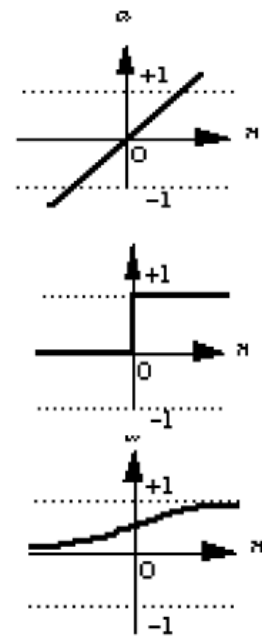
Output
layer

Aktivasyon fonksiyonları:

Nöronun çıktısının genliğini sınırladığı için limit işlevi olarak da adlandırılır.

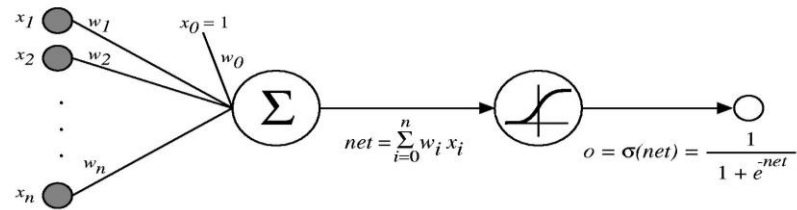
Many types of activations functions are used:

- linear: $a = f(n) = n$
- threshold: $a = \begin{cases} 1 & \text{if } n \geq 0 \\ 0 & \text{if } n < 0 \end{cases}$
(hardlimiting)
- sigmoid: $a = 1/(1+e^{-n})$
- ...



Name	Input/Output Relation	Icon	MATLAB Function
Hard Limit	$a = 0 \quad n < 0$ $a = 1 \quad n \geq 0$		hardlim
Symmetrical Hard Limit	$a = -1 \quad n < 0$ $a = +1 \quad n \geq 0$		hardlims
Linear	$a = n$		purelin
Saturating Linear	$a = 0 \quad n < 0$ $a = n \quad 0 \leq n \leq 1$ $a = 1 \quad n > 1$		satlin
Symmetric Saturating Linear	$a = -1 \quad n < -1$ $a = n \quad -1 \leq n \leq 1$ $a = 1 \quad n > 1$		satlins
Log-Sigmoid	$a = \frac{1}{1 + e^{-n}}$		logsig
Hyperbolic Tangent Sigmoid	$a = \frac{e^n - e^{-n}}{e^n + e^{-n}}$		tansig
Positive Linear	$a = 0 \quad n < 0$ $a = n \quad 0 \leq n$		poslin
Competitive	$a = 1 \quad \text{neuron with max } n$ $a = 0 \quad \text{all other neurons}$		compet

Sigmoid unit:

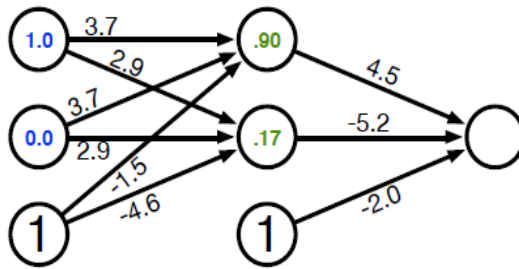


$\sigma(x)$ is the sigmoid function

$$\frac{1}{1 + e^{-x}}$$

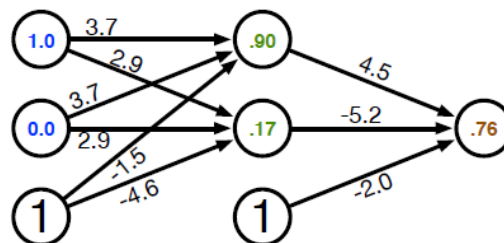
Nice property: $\frac{d\sigma(x)}{dx} = \sigma(x)(1 - \sigma(x))$

Örnek: Basit Sinir Ağı



$$\text{sigmoid}(1.0 \times 3.7 + 0.0 \times 3.7 + 1 \times -1.5) = \text{sigmoid}(2.2) = \frac{1}{1 + e^{-2.2}} = 0.90$$

$$\text{sigmoid}(1.0 \times 2.9 + 0.0 \times 2.9 + 1 \times -4.5) = \text{sigmoid}(-1.6) = \frac{1}{1 + e^{1.6}} = 0.17$$

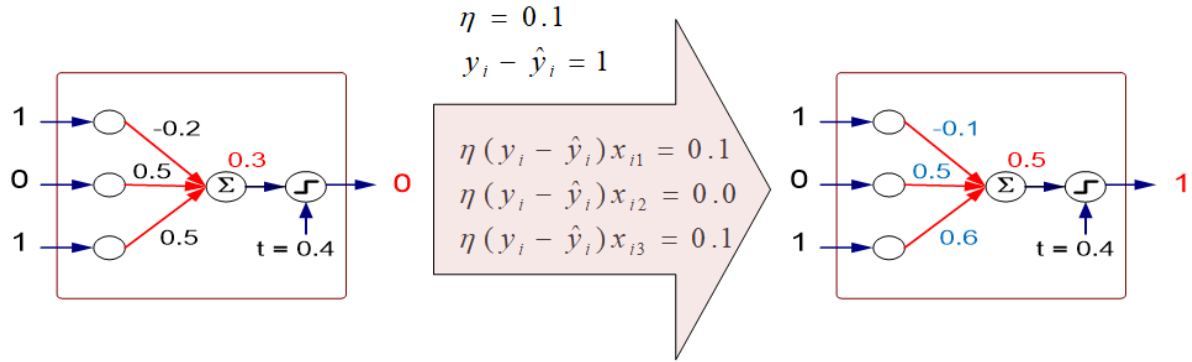


• Output

$$\text{sigmoid}(.90 \times 4.5 + .17 \times -5.2 + 1 \times -2.0) = \text{sigmoid}(1.17) = \frac{1}{1 + e^{-1.17}} = 0.76$$

Bir numuneyi işleme örneği:

X_1	X_2	X_3	Y
1	0	1	1



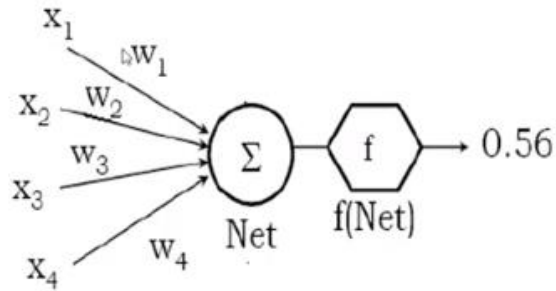
Örnek:

Girişler

$x_1 = 0.5$
 $x_2 = 0.6$
 $x_3 = 0.2$
 $x_4 = 0.7$

Ağırlıklar

$w_1 = -0.2$
 $w_2 = 0.6$
 $w_3 = 0.2$
 $w_4 = -0.1$



Hücrenin net girdisi;

$$\text{Net} = \sum x_i w_i \quad i=1 \dots 4$$

$$\text{Net} = 0.5 \cdot (-0.2) + 0.6 \cdot 0.6 + 0.2 \cdot 0.2 + 0.7 \cdot (-0.1)$$

$$\text{Net} = 0.23$$

- Sigmoid aktivasyon fonksiyonuna göre hücrenin çıkışı;

$$f(\text{Net}) = 1 / (1 + e^{-0.23})$$

$$f(\text{Net}) = 0.56$$

6.1.3. Types of Algorithms used in Deep Learning

Here is the list of top 10 most popular deep learning algorithms:

1. Convolutional Neural Networks (CNNs)
2. Long Short Term Memory Networks (LSTMs)
3. Recurrent Neural Networks (RNNs)
4. Generative Adversarial Networks (GANs)
5. Radial Basis Function Networks (RBFNs)
6. Multilayer Perceptrons (MLPs)
7. Self Organizing Maps (SOMs)
8. Deep Belief Networks (DBNs)
9. Restricted Boltzmann Machines(RBMs)
10. Autoencoders

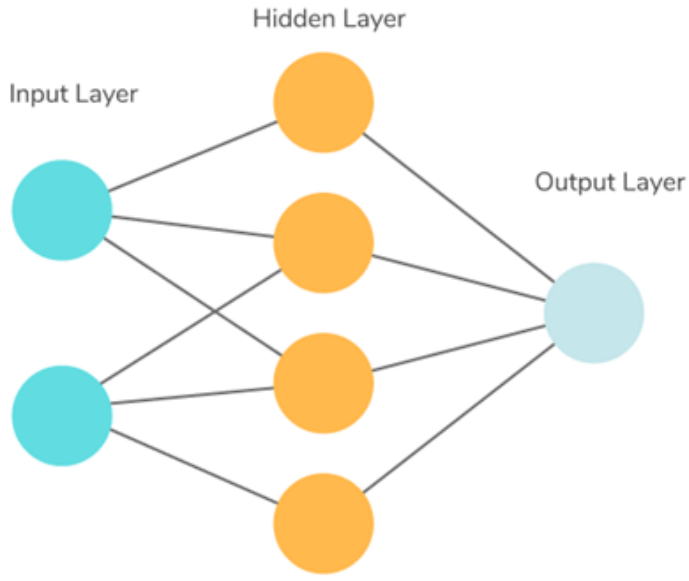
Derin öğrenme algoritmaları neredeyse her tür veriyle çalışır ve karmaşık sorunları çözmek için büyük miktarda bilgi işlem gücü ve bilgi gerektirir.

What do you understand about Neural Networks in the context of Deep Learning?

Derin Öğrenme bağlamında Sinir Ağları hakkında ne anlıyorsunuz?

Sinir Ağları, insan vücudundaki biyolojik sinir ağlarına çok benzeyen yapay sistemlerdir. Bir sinir ağı, insan beyninin nasıl çalıştığını taklit eden bir yöntem kullanarak bir veri yığınınındaki temel ilişkileri tanımaya çalışan bir dizi algoritmadır. Göreve özel kurallar olmadan, bu sistemler çeşitli veri kümelerine ve örneklerle maruz kalarak görevleri yapmayı öğrenirler. Buradaki fikir, bu veri kümelerinin önceden kodlanmış bir anlayışıyla programlanmak yerine, sistemin beslendiği verilerden tanımlayıcı özellikleri türetmesidir. Sinir ağları, eşik mantık hesaplama modelleri üzerine kuruludur. Sinir ağları, değişen girdilere uyum sağlayabildiğinden, çıktı kriterlerinin yeniden tasarlanmasını gerektirmeden mümkün olan en iyi sonucu üretebilirler.

A Simple Neural Network



What are the applications of deep learning?

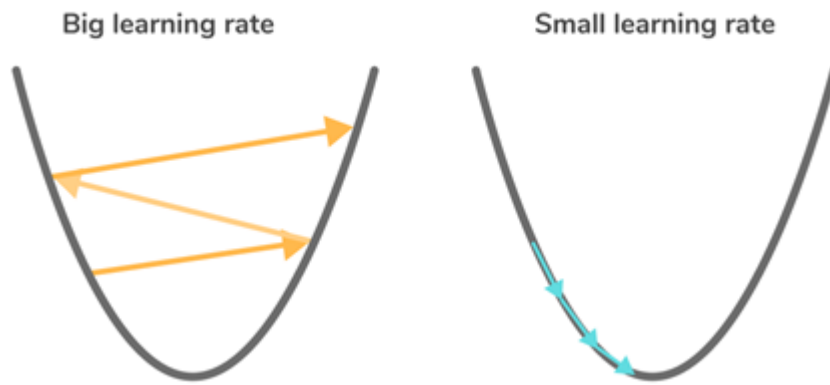
Derin öğrenme uygulamalarından bazıları şunlardır: -

- Örüntü tanıma ve doğal dil işleme.
- Görüntülerin tanınması ve işlenmesi.
- Otomatik çeviri.
- Duygu analizi.
- Soruları yanıtlamak için sistem.
- Nesnelerin Sınıflandırılması ve Tespiti.
- Makineye Göre El Yazısı Oluşturma.
- Otomatik metin oluşturma.
- Siyah Beyaz görüntülerin renklendirilmesi.

Explain learning rate in the context of neural network models. What happens if the learning rate is too high or too low?

Öğrenme hızını sinir ağı modelleri bağlamında açıklayın. Öğrenme oranı çok yüksek veya çok düşükse ne olur?

Öğrenme hızı 0 ile 1 arasında değişen bir sayıdır. Sinir ağı eğitim modellerinde en önemli ayarlanabilir hiperparametrelerden biridir. Öğrenme oranı, bir sinir ağı modelinin belirli bir duruma ne kadar hızlı veya yavaş adapte olduğunu ve öğrendiğini belirler. Daha yüksek bir öğrenme oranı değeri, modelin yalnızca birkaç eğitim dönemine ihtiyaç duyduğunu ve hızlı değişiklikler ürettiğini gösterirken, daha düşük bir öğrenme oranı, modelin yakınsamasının uzun zaman alabileceğini veya hiçbir zaman yakınsayıp kötü bir çözümde takılıp kalmayacağını gösterir. Sonuç olarak, çok düşük veya çok yüksek bir öğrenme oranı kullanmak yerine, deneme yanılma yoluyla iyi bir öğrenme oranı değeri oluşturulması önerilir.



Yukarıdaki görüntüde, büyük bir öğrenme oranının bizi istenen çıktıdan uzaklaşmaya yönlendirdiğini açıkça görebiliriz. Ancak, küçük bir öğrenme oranına sahip olmak, sonunda bizi istenen çıktıya götürür.

What are the advantages of neural networks?

Sinir ağlarının avantajları şunlardır:

- Sinir ağları son derece uyarlanabilirdir ve hem sınıflandırma hem de regresyon problemlerinin yanı sıra çok daha karmaşık problemler için kullanılabilirler. Sinir ağları da oldukça ölçeklenebilir. Her biri kendi nöron setine sahip, istediğimiz kadar katman oluşturabiliriz. Çok fazla veri noktası olduğunda, sinir ağlarının en iyi sonuçları ürettiği gösterilmiştir. Görüntüler, metinler vb. gibi doğrusal olmayan verilerle en iyi şekilde kullanılırlar. Sayısal bir değere dönüştürülebilen herhangi bir veriye uygulanabilirler.
- Sinir ağı modu bir kez eğitildikten sonra, çıktıları çok hızlı bir şekilde verirler. Böylece zamandan tasarruf sağlarlar.

What are the disadvantages of neural networks?

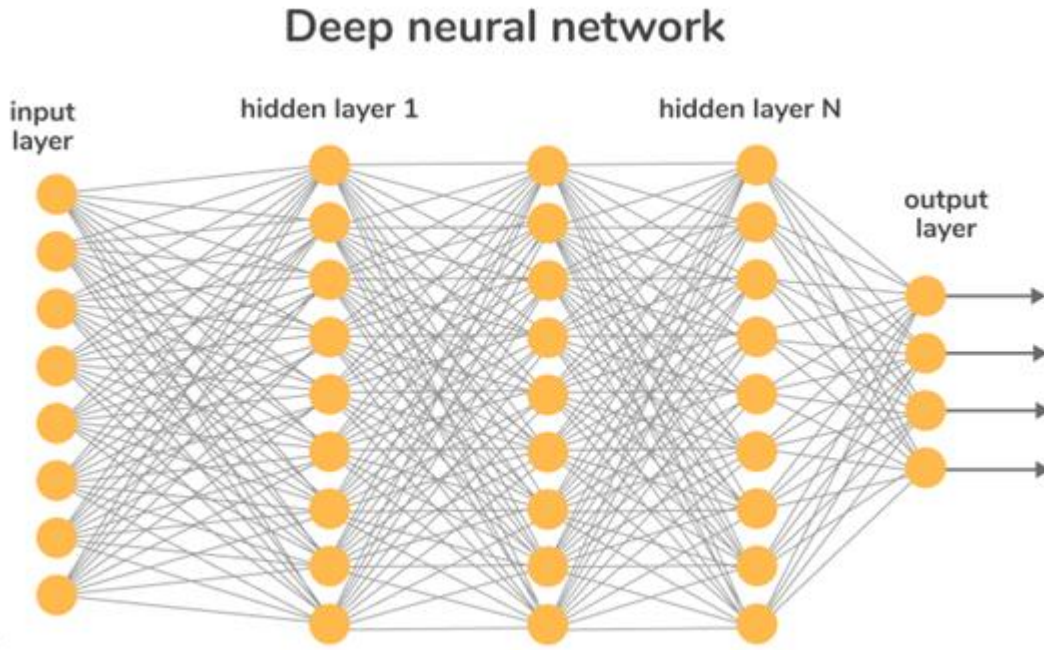
Sinir ağlarının dezavantajları şunlardır: -

- Sinir ağlarının "kara kutu" yönü, iyi bilinen bir dezavantajdır. Yani, sinir ağının nasıl veya neden belirli bir sonuç ürettiği hakkında hiçbir fikrimiz yok. Bir sinir ağına bir köpek görüntüsü girdiğimizde ve bunun bir ördek olduğunu tahmin ettiğinde, onu bu tahmini yapmaya neyin sevk ettiğini anlamakta zorlanabiliriz.
- Bir sinir ağı modeli oluşturmak uzun zaman alır.
- Sinir ağları modelleri, her katmanda çok sayıda hesaplama yapılması gerektiğinden, hesaplama açısından pahalıdır.
- Bir sinir ağı modeli, eğitmek için geleneksel bir makine öğrenimi modelinden önemli ölçüde daha fazla veri gerektirir.

Explain what a deep neural network (DNN) is.

Derin sinir ağı'nın ne olduğunu açıklayın.

Giriş ve çıkış katmanları arasında çok sayıda katmana sahip bir yapay sinir ağı (YSA), derin sinir ağı (DNN) olarak bilinir. Derin sinir ağları, derin mimarileri kullanan sinir ağlarıdır. "Derin" terimi, tek bir katmanda daha fazla sayıda katmana ve birime sahip olan işlevleri ifade eder. Daha yüksek düzeyde desenler yakalamak için daha fazla ve daha büyük katmanlar ekleyerek daha doğru modeller oluşturmak mümkündür. Aşağıdaki görüntü derin bir sinir ağını göstermektedir.



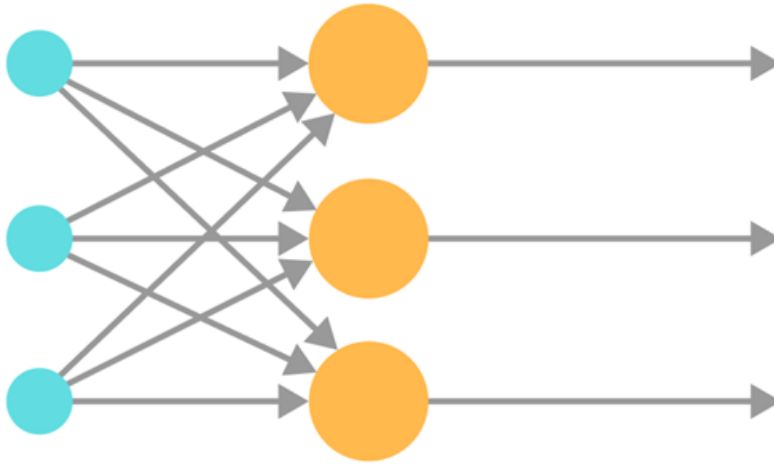
What are the different types of deep neural networks?

Derin sinir ağlarının farklı türleri nelerdir?

Aşağıda derin sinir ağlarının farklı türleri verilmiştir:

Feed Forward Sinir Ağı:

Akış kontrolünün giriş katmanında başladığı ve çıkış katmanına geçtiği en temel sinir ağı türüdür. Bu ağlar yalnızca tek bir katmana veya tek bir gizli katmana sahiptir. Bu ağda geri yayılım mekanizması yoktur çünkü veriler yalnızca tek bir şekilde akar. Bu ağın giriş katmanı, girişte bulunan ağırlıkların toplamını alır. Bu ağlar, bilgisayarla görme tabanlı yüz tanıma yönteminde kullanılmaktadır.



Radial Basis Function Neural Network:

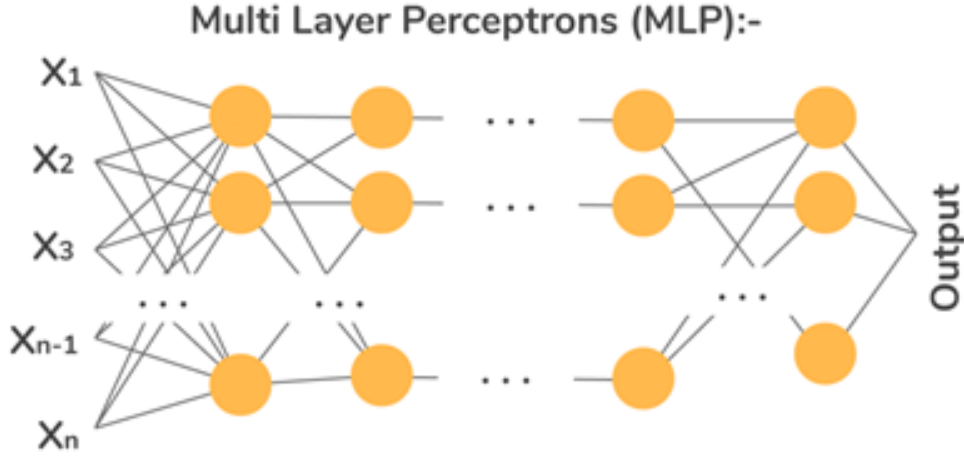
Radyal Temel Fonksiyonlu Sinir Ağı:

Bu tür sinir ağı genellikle birden fazla katmana, tercihen iki katmana sahiptir. Bu ağ türünde herhangi bir konumdan merkeze olan bağıl uzaklık belirlenir ve bir sonraki katmana aktarılır. Kesintileri önlemek için, gücü mümkün olan en kısa sürede geri yüklemek için güç restorasyon sistemlerinde yaygın olarak radyal tabanlı ağlar kullanılır.

Multi-Layer Perceptrons (MLP):

Çok Katmanlı Algılayıcılar (MLP):

Çok katmanlı algılayıcı (MLP), bir tür ileri beslemeli yapay sinir ağıdır (YSA). MLP'ler, birbirini takip eden tamamen bağlantılı katmanlardan oluşan en basit derin sinir ağlarıdır. Her ardışık katman, önceki katmanın tüm çıktılarının ağırlıklı toplamı olan (tamamen bağlantılı) doğrusal olmayan işlevler koleksiyonundan oluşur. Konuşma tanıma ve diğer makine öğrenimi sistemleri büyük ölçüde bu ağlara dayanır.



What do you mean by end-to-end learning?

Uçtan uca öğrenmeden kastınız nedir?

Bir modelin ham verilerle beslendiği ve tüm verilerin aynı anda herhangi bir ara adım olmadan istenen sonucu oluşturmak için eğitildiği bir derin öğrenme prosedürüdür. Tüm farklı adımların sıralı olarak değil aynı anda eğitildiği bir derin öğrenme yöntemidir. Uçtan uca öğrenme, genellikle daha düşük sapma ile sonuçlanan örtük özellik mühendisliği gereksinimini ortadan kaldırma avantajına sahiptir. Sürücüsüz otomobiller, uçtan uca öğrenme içeriğinizde kullanabileceğiniz mükemmel bir örnektir. İnsan girdisi tarafından yönlendirilirler ve görevleri yerine getirmek için bir CNN (**Convolutional Neural Network**) kullanarak bilgileri otomatik olarak öğrenmek ve yorumlamak üzere programlanırlar. Bir başka iyi örnek, kaydedilmiş bir ses klibinden (giriş) yazılı bir dökümün (çıkış) üretilmesidir. Buradaki model, tüm adımları ve görevleri yönetebileceği gerçeğine odaklanarak ortadaki tüm adımları atlar.

What do you understand about gradient clipping in the context of deep learning?

Derin öğrenme bağlamında gradyan kırpma hakkında ne anlıyorsunuz?

Gradyan Kırpma, geri yayılım sırasında meydana gelen gradyanların patlaması (zaman içinde büyük hata gradyanlarının biriktiği ve eğitim sırasında sinir ağı model ağırlıklarında büyük değişikliklere neden olduğu bir durum) sorunuyla başa çıkmak için bir tekniktir. Eğimlerin patlaması sorunu, eğitim sırasında eğimler aşırı büyüdüğünde ve modelin kararsız hale gelmesine neden olduğunda ortaya çıkar. Gradyan beklenen aralığı geçtiyse, gradyan değerleri, belirli bir minimum veya maksimum değere tek tek yönlendirilir. Gradyan kırpma, bir sinir ağını eğitirken sayısal kararlılığı artırır, ancak modelin performansı üzerinde çok az etkisi vardır.

Explain Forward and Back Propagation in the context of deep learning.

İleri ve Geri Yayılımı derin öğrenme bağlamında açıklayın.

İleri Yayılım: Ağın girdi katmanı ile çıktı katmanı arasındaki gizli katman, girdileri ağırlıklarla alır. Her gizli katmandaki her düğümdeki aktivasyonun çıktısını hesaplıyoruz ve bu, son çıktı katmanına ulaşana kadar bir sonraki katmana yayılıyor. Girdilerden ileriye doğru yayılma olarak bilinen son çıktı katmanına geçiyoruz.

Geri Yayılım: Ağın son katmanından hata bilgilerini ağ içindeki tüm ağırlıklara gönderir. Önceki çağın (yani yineleme) hata oranına dayalı olarak bir sinir ağının ağırlıklarının ince ayarını yapmak için kullanılan bir tekniktir. Ağırlıklara ince ayar yaparak hata oranlarını düşürebilir ve modelin genellemesini iyileştirebilir, böylece daha güvenilir hale getirebilirsiniz. Geri yayılım süreci aşağıdaki adımlara ayrılabilir: Eğitim verilerini ağ üzerinden yayarak çıktı üretebilir. Ardından, hedef ve çıkış değerlerini kullanarak çıkış aktivasyonları için hata türevini hesaplar. Önceki katmanın çıktı aktivasyonundaki hatanın türevini hesaplamak için geri yayılabilir ve tüm gizli katmanlar için böyle devam edebilir. Daha önce elde edilen türevleri ve tüm gizli katmanları kullanarak ağırlıklar için hata türevini hesaplar. Ağırlıklar, bir sonraki katmandan elde edilen hata türevlerine göre güncellenir.

What do you mean by hyperparameters in the context of deep learning?

Derin öğrenme bağlamında hiperparametreler ile ne demek istiyorsunuz?

Hiperparametreler, ağ topolojisini (örneğin, gizli birimlerin sayısı) ve ağın nasıl eğitildiğini (Örn: Öğrenme Hızı) belirleyen değişkenlerdir. Modeli eğitmeden önce, yani ağırlıkları ve sapmayı optimize etmeden önce ayarlanırlar.

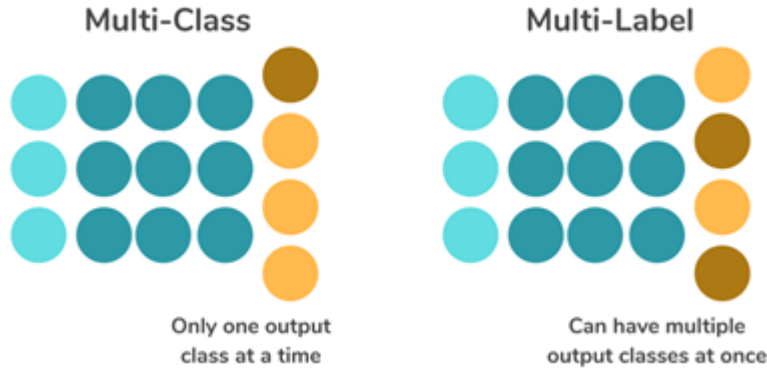
Aşağıda bazı hiperparametre örnekleri verilmiştir:

Gizli katman sayısı: Düzenleştirme teknikleri ile bir katman içindeki birçok gizli birim doğruluğu artırabilir. Birim sayısı azaltılırsa, eksik yerleştirme meydana gelebilir.

Öğrenme Hızı: Öğrenme hızı, bir ağın parametrelerinin güncellenme hızıdır. Öğrenme süreci, düşük bir öğrenme oranı ile yavaşlar, ancak sonunda birleşir. Daha hızlı bir öğrenme oranı, öğrenme sürecini hızlandırır, ancak yakınsamayabilir. Düşen bir Öğrenme oranı genellikle arzu edilir.



Difference between multi-class and multi-label classification problems.



Çok sınıflı bir sınıflandırma probleminde sınıflandırma görevi, birbirini dışlayan ikiden fazla sınıfa (kesişimi olmayan veya ortak özelliği olmayan sınıflar) sahipken, çok etiketli bir sınıflandırma probleminde, görevler farklı olmasına rağmen, her etiketin farklı bir sınıflandırma görevi vardır. bir şekilde ilişkilidir. Örneğin, kedi, köpek veya ayı olabilecek bir grup hayvan fotoğrafını sınıflandırmak, her örneğin yalnızca bir tür olabileceğini varsayan çok sınıflı bir sınıflandırma problemidir, bu da bir görüntünün kedi veya kedi olarak kategorize edilebileceğini gösterir. köpek, ama ikisi aynı anda değil. Şimdi aşağıdaki resmi değiştirmek istediğinizi varsayalım.

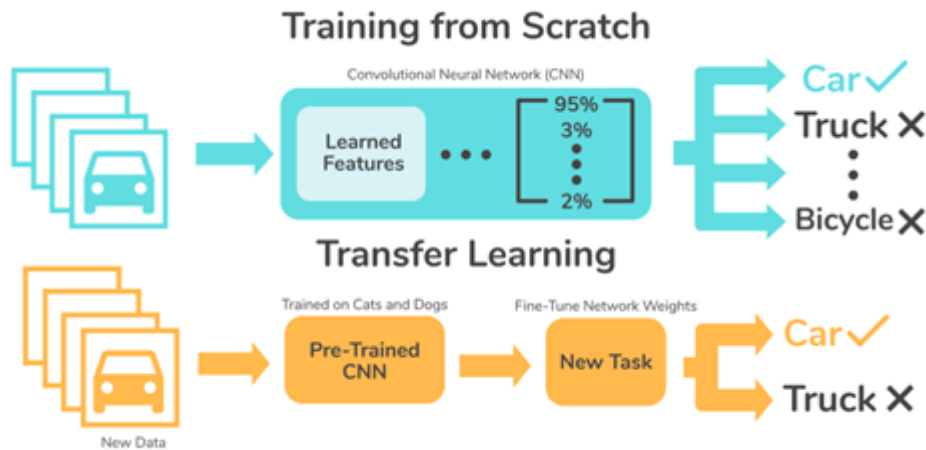


Yukarıdaki resim, her iki canlıyı da betimlediği için hem kedi hem de köpek olarak kategorize edilmelidir. Çok etiketli bir sınıflandırma sorununda her örneğe bir etiket kümesi tahsis edilir ve sınıflar birbirini dışlamaz. Çok etiketli bir sınıflandırma probleminde, bir örüntü bir veya daha fazla sınıfa ait olabilir.

Explain transfer learning in the context of deep learning.

Derin öğrenme bağlamında transfer öğrenmeyi açıkla.

Aktarım öğrenimi, veri bilimcilerinin benzer bir görev için kullanılan önceki bir makine öğrenimi modelinden öğrendiklerini kullanmalarına olanak tanıyan bir öğrenme tekniğidir. Bu öğrenmede, insanların bilgilerini aktarma yeteneği örnek olarak kullanılmaktadır. Bisiklete binmeyi öğrenirseniz, diğer iki tekerlekli araçları kullanmayı daha kolay öğrenebilirsiniz. Otonom otomobil sürüşü için eğitilmiş bir model otonom kamyon sürüşü için de kullanılabilir. Özellikler ve ağırlıklar, yeni modeli eğitmek için kullanılabilir ve yeniden kullanılmasına izin verir. Sınırlı veri olduğunda, bir modeli hızlı bir şekilde eğitmek için aktarım öğrenimi etkili bir şekilde çalışır.



Yukarıdaki görüntüde, ilk diyagram bir modeli sıfırdan eğitmeyi temsil ederken, ikinci diyagram, farklı araç sınıflarını sınıflandırmak için kediler ve köpekler üzerinde önceden eğitilmiş bir modelin kullanılmasını temsil eder, böylece transfer öğrenmeyi temsil eder.

What are the advantages of transfer learning?

Aktarım yoluyla öğrenmenin avantajları nelerdir?

Transfer öğrenmenin avantajları şunlardır:

Daha iyi başlangıç modeli: Diğer öğrenme yöntemlerinde, sıfırdan bir model oluşturmalsınız. Transfer öğrenme daha iyi bir başlangıç noktasıdır çünkü başlangıç modelinin ayrıntılarını bilmeden görevleri daha üst düzeyde gerçekleştirmemizi sağlar.

Daha yüksek öğrenme oranı: Problem zaten benzer bir görev için öğretildiği için, transfer öğrenme eğitim sırasında daha hızlı bir öğrenme oranına izin verir.

Eğitimden sonra daha yüksek doğruluk: Transfer öğrenimi, daha iyi bir başlangıç noktası ve daha yüksek öğrenme oranı sayesinde daha doğru çıktıyla sonuçlanan bir derin öğrenme modelinin daha yüksek bir performans düzeyinde yakınsamasına olanak tanır.

Is it possible to train a neural network model by setting all biases to 0? Also, is it possible to train a neural network model by setting all of the weights to 0?

Tüm yanlılıkları 0'a ayarlayarak bir sinir ağı modeli eğitmek mümkün müdür? Ayrıca, tüm ağırlıkları 0'a ayarlayarak bir sinir ağı modeli eğitmek mümkün müdür?

Evet, tüm yanlılıklar sıfıra ayarlanırsa bile, sinir ağı modelinin öğrenme şansı vardır.

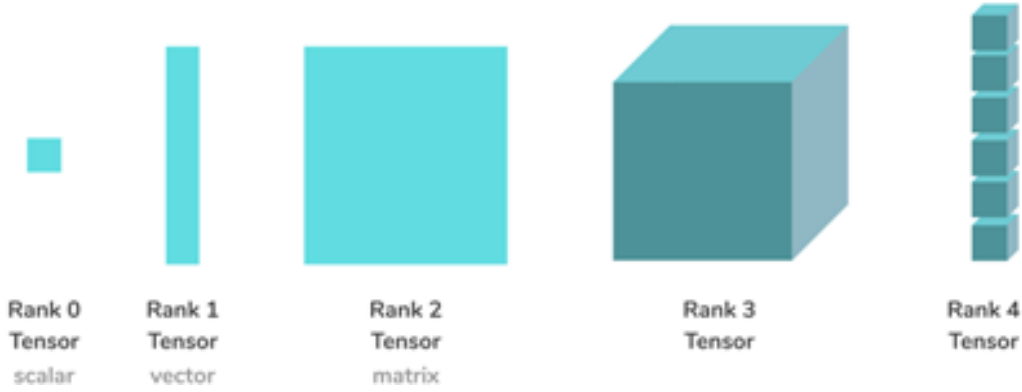
Hayır, sinir ağı bir görevi tamamlamayı asla öğrenemeyeceğinden, tüm ağırlıkları 0'a ayarlayarak bir modeli eğitmek imkansızdır. Tüm ağırlıklar sıfıra ayarlandığında, her w için türevler sabit kalır, bu da nöronların her yinelemeye aynı özellikleri öğrenmesine neden olur. Ağırlıkların herhangi bir sabit başlatılması, sadece sıfır değil, muhtemelen kötü bir sonuç üretecektir.

What is a tensor in deep learning?

Derin öğrenmede tensör nedir?

Tensör, vektörlerin ve matrislerin genelleştirilmesini temsil eden çok boyutlu bir dizidir. Derin öğrenmede kullanılan temel veri yapılarından biridir. Tensörler, temel veri türlerinin n boyutlu dizileri olarak temsil edilir. Tensördeki her elemanın veri tipi aynıdır ve veri tipi her zaman bilinir. Şeklin yalnızca bir bölümünün (yani boyutların sayısı ve her boyutun boyutu) bilinmesi mümkündür. Çoğu işlem, girdileri aynı şekilde tam olarak biliniyorsa, tam olarak bilinen tensörler verir, ancak diğer durumlarda, bir tensörün şekli yalnızca grafik yürütme zamanında belirlenebilir.

A tensor is an N-dimensional array of data

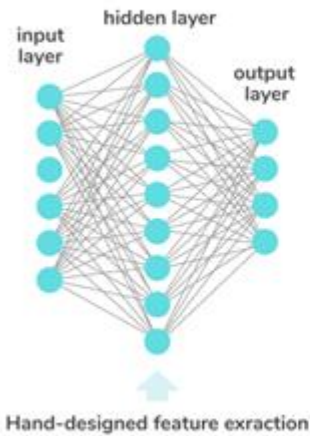


Explain the difference between a shallow network and a deep network.

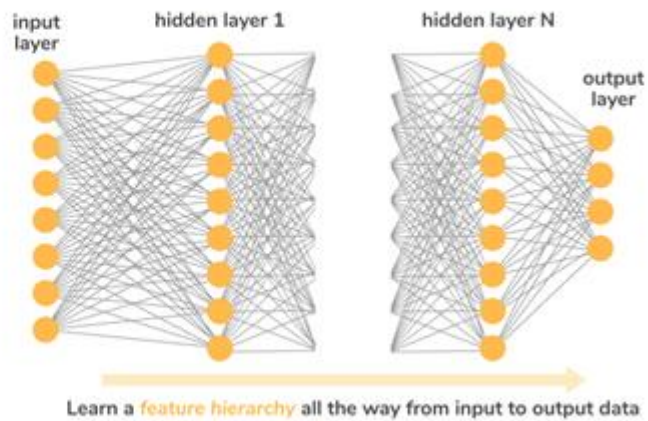
Sığ ağ ile derin ağ arasındaki farkı açıklayın.

Her sinir ağında bir gizli katman ile giriş ve çıkış katmanları bulunur. Sığ sinir ağları, yalnızca bir gizli katmana sahip olanlardır, oysa derin sinir ağları çok sayıda gizli katman içerir. Hem sığ hem de derin ağlar herhangi bir işleve sığabilir, ancak sığ ağlar çok sayıda girdi parametresi gerektirirken, derin ağlar birkaç katmanları nedeniyle az sayıda girdi parametresi ile işlevlere uyabilir. Model, her katmandaki girdinin yeni ve soyut bir temsilini öğrendiğinden, şu anda sığ ağlar yerine derin ağlar tercih edilmektedir. Sığ ağlara kıyasla, parametre sayısı ve hesaplamalar açısından da çok daha verimlidirler.

Shallow neural network



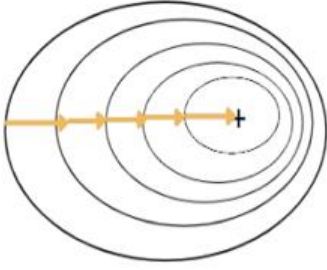
Deep neural network



Explain Batch Gradient Descent.

Toplu Dereceli İniş: Toplu Dereceli İniş, her adımda tüm eğitim seti üzerinde hesaplama gerektirir (gradyan inişinin her adımında yer alır) ve bu nedenle çok büyük eğitim setlerinde oldukça yavaştır. Sonuç olarak, Batch Gradient Descent, hesaplama açısından son derece pahalı hale gelir. Bu, dışbükey veya biraz pürüzsüz olan hata manifoldları için idealdir. Batch Gradient Descent, özelliklerin sayısı arttıkça güzel bir şekilde ölçeklenir.

Batch Gradient Descent



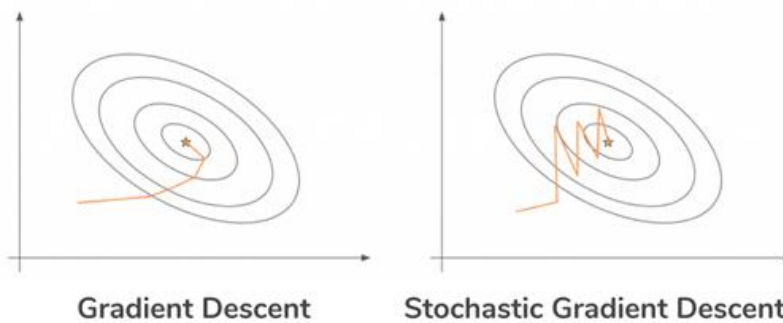
Explain Stochastic Gradient Descent. How is it different from Batch Gradient Descent?

Stochastic Gradient Descent:

Stokastik Gradyan İnişini açıklayın. Batch Gradient Descent'den farkı nedir?

Stokastik Dereceli İniş: Stokastik Dereceli İniş, her adımda degradeleri hesaplamak için tüm eğitim setinin kullanılması olan Toplu Dereceli İniş ile büyük zorluğun üstesinden gelmeyi amaçlar. Bu, doğası gereği stokastiktir, yani her adımda eğitim verilerinin "rastgele" bir örneğini seçer ve ardından gradyanı hesaplar; bu, aynı anda değiştirilecek çok daha az veri olduğundan Toplu Gradyan İniş'ten önemli ölçüde daha hızlıdır. Stokastik Gradyan İnişi, kısıtlanmamış optimizasyon problemleri için en uygun olanıdır. SGD'nin stokastik doğasının bir dezavantajı vardır, çünkü bir kez minimum değere yaklaştığında sabitlenmez ve bunun yerine sekerek bize model parametreleri için iyi ama optimal olmayan bir değer verir. Bu, zıplamayı azaltacak ve bir süre sonra SGD'nin küresel minimumda yerleşmesine izin verecek olan her adımda öğrenme oranını düşürerek çözülebilir.

İkisi arasındaki farklar şunlardır:



Toplu Gradyan İniş

- Gradyan, tüm eğitim veri kümesi kullanılarak hesaplanır.
- Stokastik Gradyan İnişten daha yavaştır ve hesaplama açısından daha pahalıdır.
- Büyük eğitim örnekleri için önerilmez.
- Doğada deterministiktir (rastgele değil).
- Yakınsamak için yeterli zaman verildiğinde, en iyi cevabı verir.
- Veri noktalarını rastgele karıştırmaya gerek yoktur.
- Bunda sık yerel minimumlardan çıkmak zordur.
- Bu durumda yakınsama yavaştır.

Stokastik Gradyan İniş

- Gradyanı hesaplamak için tek bir eğitim örneği kullanılır.
- Batch Gradient Descent'den daha hızlı ve hesaplama açısından daha ucuzdur.
- Büyük eğitim örnekleri için önerilir.
- Doğası gereği stokastiktir (rastgele).
- İyi bir çözüm sağlar, ancak en iyisi değildir.
- Veri örneğinin rastgele bir sırada olmasını istediğimiz için, her dönem için eğitim setini karıştıracağız.
- Sık yerel minimumlardan kaçma şansı daha yüksektir.
- Yakınsama noktasına çok daha hızlı ulaşır.

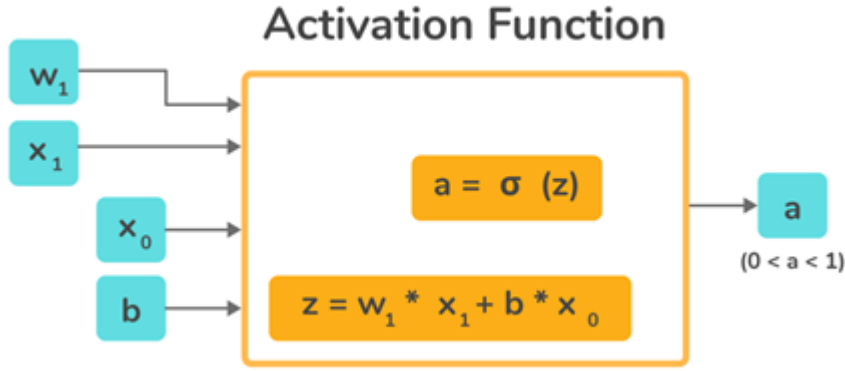
Which deep learning algorithm is the best for face detection?

Yüz tanıma, çeşitli makine öğrenimi yöntemleri kullanılarak gerçekleştirilebilir, ancak en iyileri Evrişimli Sinir Ağları ve derin öğrenmeyi kullanır. Aşağıdakiler bazı dikkate değer yüz algılama algoritmalarıdır: FaceNet, Probabilisit, Face Embedding, ArcFace, Cosface ve Spherface.

What is an activation function? What is the use of an activation function?

Aktivasyon fonksiyonu nedir? Aktivasyon fonksiyonunun kullanımı nedir?

Bir yapay sinir ağının etkinleştirme işlevi, ağın verilerdeki karmaşık kalıpları öğrenmesine yardımcı olmak için tanımlanmış bir işlevdir. Beynimizde görülen nöron tabanlı bir modelle karşılaştırıldığında, sürecin sonunda bir sonraki nörona neyin atışılacağını belirlemekten aktivasyon işlevi sorumludur. Bir YSA'da bir aktivasyon işlevi aynı işi gerçekleştirir. Bir önceki hücrenin çıkış sinyalini alır ve onu bir sonraki hücreye giriş olarak kullanılacak bir formata dönüştürür.



What do you mean by an epochs in the context of deep learning?

Derin öğrenme bağlamında bir devirden kastınız nedir?

Epoch, derin öğrenmede kullanılan ve derin öğrenme algoritmasının tam eğitim veri kümesinde yaptığı geçiş sayısını ifade eden bir terminolojidir. Gruplar genellikle veri kümelerini gruplamak için kullanılır (özellikle veri miktarı çok büyük olduğunda). "Yineleme" terimi, modelde bir toplu iş yürütme sürecini ifade eder.

Toplu iş boyutu eğitim veri kümesinin tamamıysa, dönem sayısı yineleme sayısına eşittir. Pratik nedenlerden dolayı bu genellikle böyle değildir. Birçok modelin oluşturulmasında birkaç dönem kullanılır.

Genel bir ilişki:

$$d * e = \text{ben} * b$$

burada,

d veri kümesi boyutudur

e dönem sayısıdır

ben yineleme sayısı

b yığın, grup boyutudur

While building a neural network architecture, how will you decide how many neurons and the hidden layers should the neural network have?

Bir sinir ağı mimarisi oluştururken, sinir ağının kaç nörona ve gizli katmanlara sahip olması gerektiğine nasıl karar vereceksiniz?

Bir iş problemi göz önüne alındığında, bir sinir ağı mimarisi tasarlamak için gereken nöronların ve gizli katmanların tam sayısını belirlemek için açık ve hızlı bir kural yoktur. Bir sinir ağındaki gizli katmanın boyutu, çıktı katmanlarının boyutu ile girdi katmanlarının boyutu arasında bir yerde olmalıdır. Bununla birlikte, bir sinir ağı mimarisi oluşturmaya başlamanıza yardımcı olabilecek birkaç temel yol vardır:

- Herhangi bir benzersiz gerçek dünya öngörücü modelleme problemine yaklaşmanın en iyi yöntemi, benzer gerçek dünya durumlarında sinir ağlarıyla çalışma önceki deneyimlerine dayalı olarak herhangi bir veri kümesi için neyin en iyi performansı göstereceğini görmek için bazı temel sistematik deneylerle başlamaktır. Ağ yapılandırması, kişinin problem alanı anlayışına ve sinir ağlarıyla ilgili önceki uzmanlığına dayalı olarak seçilebilir. Benzer konularda kullanılan katmanların ve nöronların sayısı, bir sinir ağının konfigürasyonunu değerlendirirken her zaman iyi bir başlangıç noktasıdır.
- Basit sinir ağı mimarisiyle başlamak ve tahmin edilen çıktı ve doğruluğa dayalı olarak sinir ağının karmaşıklığını kademeli olarak artırmak en iyisidir.

Can a deep learning model be solely built on linear regression?

Derin öğrenme modeli yalnızca doğrusal regresyon üzerine inşa edilebilir mi?

Evet, sorun doğrusal bir denklemle temsil ediliyorsa, her katman için aktivasyon işlevi olarak doğrusal bir işlev kullanılarak derin ağlar oluşturulabilir. Öte yandan, doğrusal işlevlerin bileşimi olan bir sorun doğrusal bir işlevdir ve derin bir ağ uygulayarak başarılabilecek olağanüstü bir şey yoktur çünkü ağa daha fazla düğüm eklemek makine öğrenme modelinin tahmin kapasitesini artırmaz. .

According to you, which one is more powerful - a two layer neural network without any activation function or a two layer decision tree?

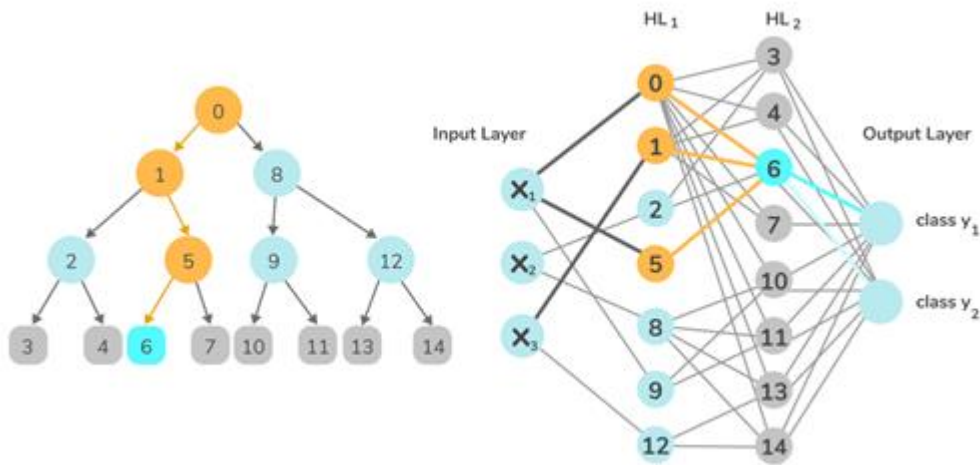
Size göre hangisi daha güçlü - aktivasyon fonksiyonu olmayan iki katmanlı bir sinir ağı mı yoksa iki katmanlı bir karar ağacı mı?

İki katmanlı bir sinir ağı, üç katmandan oluşur: bir giriş katmanı, bir gizli katman ve bir çıkış katmanı. Sinir ağlarıyla uğraşırken, girdiler ve yanıt değişkenleri arasındaki karmaşık ve doğrusal olmayan işlevsel eşlemelerle uğraşırken gerekli olduğu için bir etkinleştirme işlevi önemlidir. İki katmanlı bir sinir ağında aktivasyon fonksiyonu olmadığında, bu sadece

doğrusal bir ağıdır. Aktivasyon işlevi olmayan bir Sinir Ağı, yalnızca sınırlı kapasiteye sahip olan ve sıklıkla iyi performans göstermeyen bir Lineer Regresyon Modelidir.

İki katman derinliğine sahip bir karar ağacı, iki katmanlı bir karar ağacı olarak bilinir. Karar Ağaçları, verilerin bir parametreye göre sürekli olarak bölündüğü bir tür denetimli makine öğrenimidir (yani, makine, eğitim verilerinde girdinin ne olduğu ve ilgili çıktının ne olduğu ile beslenir). Ağacı açıklamak için iki varlık, karar düğümleri ve yapraklar kullanılabilir. Kararlar veya nihai sonuçlar yapraklarla temsil edilir. Ve veriler karar düğümlerinde ayrılır.

Bu iki modeli karşılaştırırken, iki katmanlı sinir ağı (aktivasyon fonksiyonu olmadan) iki katmanlı karar ağacından daha güçlüdür, çünkü iki katmanlı sinir ağı bir model oluştururken daha fazla özelliği dikkate alacaktır, oysa iki katmanlı karar ağaç yalnızca 2 veya 3 özelliği dikkate alacaktır.

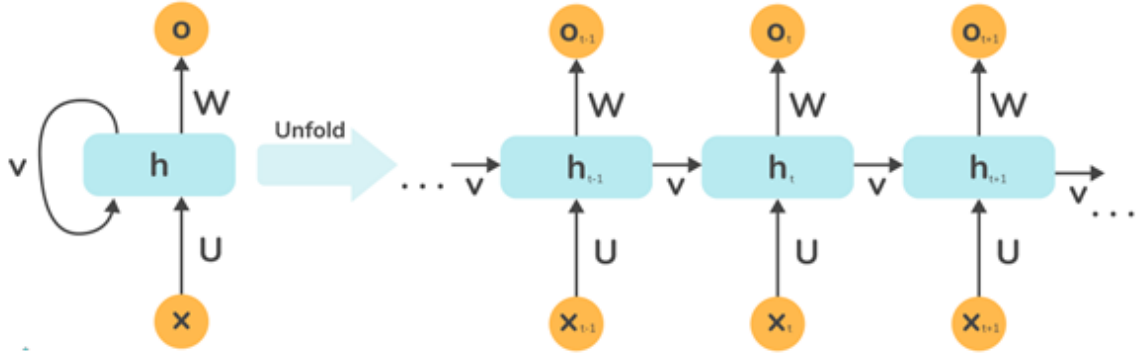


Soldaki şekil 2 katmanlı bir karar ağacını ve sağdaki şekil 2 katmanlı bir sinir ağını göstermektedir.

How does Recurrent Neural Network backpropagation vary from Artificial Neural Network backpropagation?

Tekrarlayan Sinir Ağı geri yayılımı, Yapay Sinir Ağı geri yayılımından nasıl farklıdır?

Tekrarlayan Sinir Ağlarında Geri Yayılım, aşağıdaki resimde gösterildiği gibi Tekrarlayan Sinir Ağlarındaki her düğümün ek bir döngüye sahip olması anlamında Yapay Sinir Ağlarından farklıdır:



Bu döngü, özünde, ağa geçici bir bileşen ekler. Bu, genel bir yapay sinir ağı ile imkansız olan, verilerden sıralı bilgilerin yakalanmasına izin verir.

What exactly do you mean by exploding and vanishing gradients?

Patlayan ve kaybolan gradyanlarla tam olarak ne demek istiyorsunuz?

Minimum değere doğru artan adımlar atarak, gradyan iniş algoritması hatayı en aza indirmeyi amaçlar. Bir sinir ağındaki ağırlıklar ve yanlılıklar bu işlemler kullanılarak güncellenir.

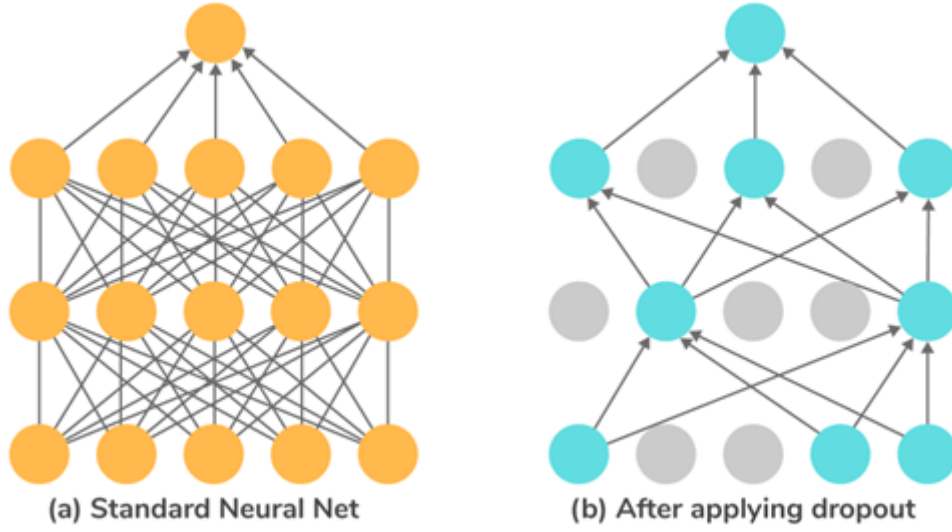
Bununla birlikte, bazen adımlar aşırı derecede büyür ve ağırlıkların ve yanlılık terimlerinin güncellemelerinin artmasıyla sonuçlanır - ağırlıkların taşıdığı (veya NaN, yani Sayı Değil) noktaya kadar. Patlayan bir gradyan bunun sonucudur ve kararsız bir yöntemdir.

Öte yandan, adımlar aşırı derecede küçükse, ağırlıklarda ve sapma terimlerinde küçük, hatta ihmal edilebilir değişikliklerle sonuçlanır. Sonuç olarak, her seferinde neredeyse aynı ağırlıklara ve yanlılıklara sahip, hiçbir zaman en az hata işlevine ulaşmayan bir derin öğrenme modeli eğitebiliriz. Kaybolan gradyan buna denir.

What do you know about Dropout?

Bırakma hakkında ne biliyorsunuz?

Bırakma, fazla uydurmayı önlemeye yardımcı olan ve dolayısıyla genellenebilirliği artıran bir düzenleme yaklaşımıdır (yani model, yalnızca eğitim veri seti ile sınırlı olmak yerine, genel olarak girdilerin çoğu için doğru çıktıyı tahmin eder). Genel olarak, %20 ile iyi bir başlangıç noktası olmak üzere nöronların yüzde 20 ile yüzde 50'si gibi düşük bir bırakma değeri kullanmalıyız. Çok düşük bir olasılığın etkisi yoktur, oysa çok yüksek bir sayı ağırlık yetersiz öğrenmesine neden olur.



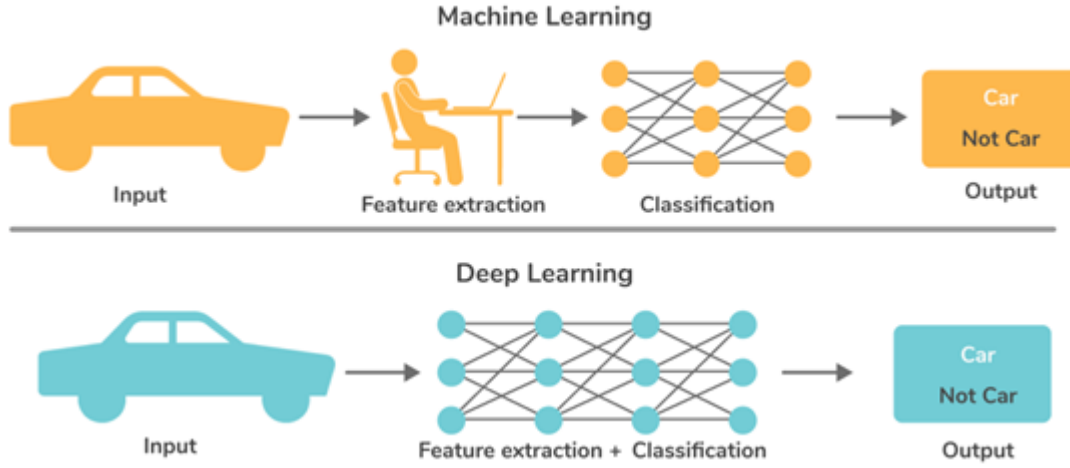
Daha büyük bir ağda bırakma yöntemini kullandığınızda, modelin bağımsız temsilleri öğrenmek için daha fazla fırsatı olduğundan daha iyi sonuçlar elde etme olasılığınız daha yüksektir.

Differentiate between Deep Learning and Machine Learning.

Derin Öğrenme ile Makine Öğrenimi arasında ayrım yapın.

Derin Öğrenme: Derin Öğrenme, tekrarlayan bir sinir ağının ve bir yapay sinir ağının bağlantılı olduğu bir Makine Öğrenimi alt sınıfıdır. Algoritmalar, makine öğrenimi algoritmalarıyla aynı şekilde oluşturulur, ancak daha birçok algoritma seviyesi vardır. Yapay sinir ağı, algoritmanın tüm ağlarının bir araya getirilmesini ifade eder. Çok daha basit bir ifadeyle derin öğrenme kavramı olan beyindeki tüm sinir ağlarını birbirine bağlayarak insan beynini taklit eder. Her türlü karmaşık problemin üstesinden gelmek için algoritmalar ve bir teknik kullanır.

Makine Öğrenimi: Makine öğrenimi, bir sistemin o seviyeye programlanmak zorunda kalmadan deneyimlerinden öğrenmesini ve büyümesini sağlayan bir Yapay Zeka (AI) alt kümesidir. Veriler, öğrenmek ve doğru sonuçlar elde etmek için Machine Learning tarafından kullanılır. Makine öğrenimi algoritmaları, daha fazla veri elde ederek öğrenme ve performanslarını iyileştirme yeteneğine sahiptir. Makine öğrenimi şu anda diğer uygulamaların yanı sıra sürücüsüz arabalarda, siber dolandırıcılık tespitinde, yüz tanımda ve Facebook arkadaş önerisinde kullanılmaktadır.



The following table illustrates the difference between them:

Derin Öğrenme:

- Derin Öğrenme, Makine Öğreniminin bir alt sınıfıdır.
- Derin Öğrenme, verileri temsil etmek için çok farklı bir veri temsili (YSA) olan sinir ağlarını kullanır.
- Bunda çıktı, sayısal değerlerden metin veya ses gibi serbest biçimli öğelere kadar değişir.
- Verileri işleme katmanlarından geçirerek veri özelliklerini ve ilişkilerini değerlendirmek için bir sinir ağı kullanır.
- Genellikle milyonlarca veri noktasıyla ilgilenir.
- Makine Öğrenimi, Derin Öğrenmeye dönüşmüştür. Esasen, makine öğreniminin derinliğini ifade eder.

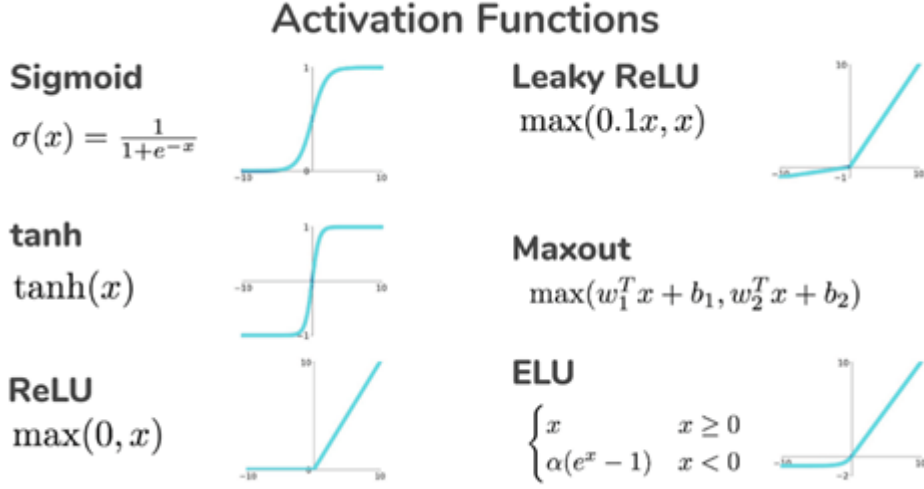
Makine öğrenme:

- Makine Öğrenimi, Derin Öğrenmenin bir süper sınıfıdır.
- Makine Öğrenimi, yapılandırılmış verileri kullandığından, verileri Derin Öğrenme'den farklı bir şekilde temsil eder.
- Bunda, çıktı sayısal değerlerden oluşur
- Girdileri model işlevlerine dönüştürmek ve gelecekteki eylemleri tahmin etmek için çeşitli otomatik teknikler kullanır.
- Genellikle binlerce veri noktasıyla ilgilenir.
- Yapay Zeka, Makine Öğrenimi'ne dönüştü.

Explain the different types of activation functions.

36. Farklı aktivasyon fonksiyon tiplerini açıklayın.

Aşağıda, farklı etkinleştirme işlevi türleri verilmiştir:



Sigmoid işlevi: Sigmoid işlevi, çoğunlukla ileri beslemeli sinir ağlarında kullanılan bir YSA'da doğrusal olmayan bir etkinleştirme işlevidir. Her yerde pozitif türevleri olan ve gerçek girdi değerleri için tanımlanmış belirli bir düzgünlük derecesine sahip türevlenebilir bir gerçek fonksiyondur. Sigmoid işlevi, derin öğrenme modellerinin çıktı katmanında bulunur ve olasılığa dayalı çıktıları tahmin etmek için kullanılır.

Hiperbolik Tanjant İşlevi (Tanh): Tanh işlevi, -1 ila 1 aralığına sahip daha yumuşak ve sıfır merkezli bir işlevdir. Çok katmanlı sinir ağları için daha yüksek eğitim performansı sağladığından, tanh işlevi sigmoid işlevinden çok daha yaygın olarak kullanılır. Tanh fonksiyonunun birincil avantajı, geri yayılıma yardımcı olan sıfır merkezli bir çıktı vermesidir.

Softmax işlevi: softmax işlevi, bir gerçek sayı vektöründen olasılık dağılımı oluşturmak için sinir ağlarında kullanılan başka bir etkinleştirme işlevi türüdür. Bu işlev, olasılıkların toplamı 1'e eşit olacak şekilde 0 ile 1 arasında bir sayı döndürür. Bu işlev en yaygın olarak çok sınıflı modellerde kullanılır ve hedef sınıf en yüksek olasılığa sahip olacak şekilde her sınıf için olasılıkları döndürür. DL mimarisinin hemen hemen tüm çıktı katmanlarında bulunabilir.

Softsign işlevi: Bu, en yaygın olarak regresyon hesaplama sorunları ve derin öğrenmeye dayalı metin okuma uygulamalarında kullanılır.

Düzeltilmiş Doğrusal Birim İşlevi: Doğrultulmuş doğrusal birim (ReLU) işlevi, üstün performans ve olağanüstü sonuçlar vermeyi vaat eden hızlı öğrenen bir yapay zekadır (AI). Derin öğrenmede, ReLU işlevi, performans ve genelleme açısından sigmoid ve tanh işlevleri gibi diğer AF'lerden daha iyi performans gösterir. İşlev, doğrusal modellerin özelliklerini

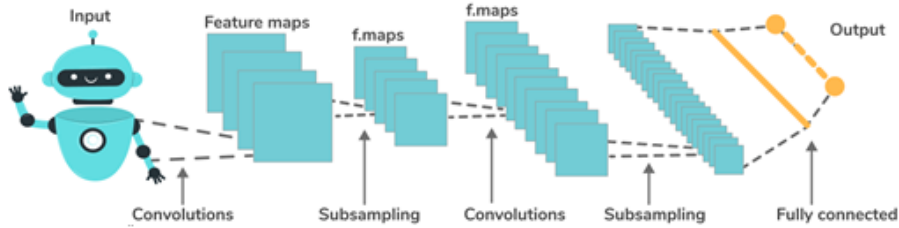
koruyan ve gradyan iniş yaklaşımlarını optimize etmeyi kolaylaştıran kabaca doğrusal bir işlevdir. Her giriş ögesinde, ReLU işlevi, sıfırdan küçük tüm değerleri sıfıra ayarlayarak bir eşik işlemi gerçekleştirir.

Üstel Doğrusal Birim İşlevi: Üstel doğrusal birimler (ELU'lar) işlevi, sinir ağı eğitimini hızlandırmak için kullanılabilen bir AF türüdür (tıpkı ReLU işlevi gibi). ELU fonksiyonunun en büyük avantajı, pozitif değerler için özdeşlik kullanarak ve modelin öğrenme özelliklerini artırarak kaybolan gradyan problemini çözebilmesidir.

Convolutional Neural Networks (CNNs):

Konvolüsyonel Sinir Ağları çoğunlukla bilgisayar görsellerde kullanılır. MLP'lerdeki tam bağlantılı katmanların aksine, bir veya daha fazla evrişim katmanı, CNN modellerinde evrişim işlemleri gerçekleştirilerek girdiden basit özellikler çıkarır. Her katman, önceki katmanın çıktılarının uzamsal olarak yakın alt kümelerinin çeşitli koordinatlarındaki ağırlıklı toplamaların doğrusal olmayan işlevlerinden oluşur ve ağırlıkların yeniden kullanılmasına izin verir.

AI sistemi, gerçek dünyadan bir dizi görüntü veya video verildiğinde resim sınıflandırma, yüz tanımlama ve görüntü anlamsal bölümlendirme gibi belirli bir görevi yerine getirmek için bu girdilerin özelliklerini otomatik olarak çıkarmayı öğrenir.



ConvNets olarak da bilinen CNN'ler, çoklu katmanlardan oluşur ve esas olarak görüntü işleme ve nesne algılama için kullanılır. Yann LeCun, ilk CNN'i 1988'de LeNet olarak adlandırıldığında geliştirdi. Posta kodları ve rakamlar gibi karakterleri tanımak için kullanıldı. CNN'ler, uydu görüntülerini tanımlamak, tıbbi görüntüleri işlemek, zaman serilerini tahmin etmek ve anormallikleri tespit etmek için yaygın olarak kullanılmaktadır.

CNN'ler Nasıl Çalışır?

CNN'ler, verilerden özellikleri işleyen ve çıkaran birden çok katmana sahiptir:

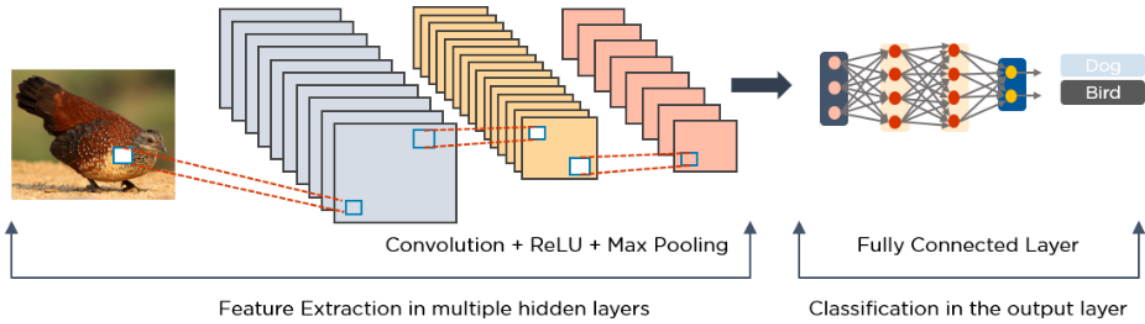
Evrişim Katmanı: CNN, evrişim işlemini gerçekleştirmek için birkaç filtreye sahip bir evrişim katmanına sahiptir.

Rektifiye Lineer Birim (ReLU): CNN'ler, ögeler üzerinde işlemleri gerçekleştirmek için bir ReLU katmanına sahiptir. Çıktı, düzeltilmiş bir özellik haritasıdır.

Havuzlama Katmanı: Düzeltilmiş özellik haritası daha sonra bir havuz katmanına beslenir. Havuzlama, özellik haritasının boyutlarını azaltan bir alt örnekleme işlemidir. Havuzlama katmanı daha sonra havuzlanmış özellik haritasından elde edilen iki boyutlu dizileri düzleştirerek tek, uzun, sürekli, doğrusal bir vektöre dönüştürür.

Tam Bağlantılı Katman: Havuzlama katmanından gelen düzleştirilmiş matris, görüntüleri sınıflandıran ve tanımlayan bir girdi olarak beslendiğinde, tam bağlantılı bir katman oluşur.

Aşağıda CNN aracılığıyla işlenen bir görüntü örneği bulunmaktadır.



In a Convolutional Neural Network (CNN), how can you fix the constant validation accuracy?

Bir Evrişimli Sinir Ağında (CNN), sabit doğrulama doğruluğunu nasıl düzeltebilirsiniz?

Herhangi bir sinir ağını eğitirken, sürekli doğrulama doğruluğu yaygın bir sorundur, çünkü ağ yalnızca örneği hatırlayarak aşırı uyum sorununa neden olur. Bir modele aşırı uydurma, sinir ağı modelinin eğitim örneğinde takdire şayan bir performans gösterdiğini, ancak modelin performansının doğrulama setinde bozulduğunu gösterir. CNN'nin sürekli doğrulama doğruluğunu iyileştirmenin bazı yolları aşağıda verilmiştir:

- Veri setini üç bölüme ayırmak her zaman iyi bir fikirdir: eğitim, doğrulama ve test etme.
- Sınırlı verilerle çalışırken, bu zorluk, sinir ağının parametreleriyle deneyler yapılarak aşılabılır.
- Eğitim veri setinin boyutunu artırarak.
- Toplu normalleştirme kullanarak.
- Düzenleştirme uygulayarak
- Ağın karmaşıklığını azaltarak

Long Short Term Memory Networks (LSTMs):

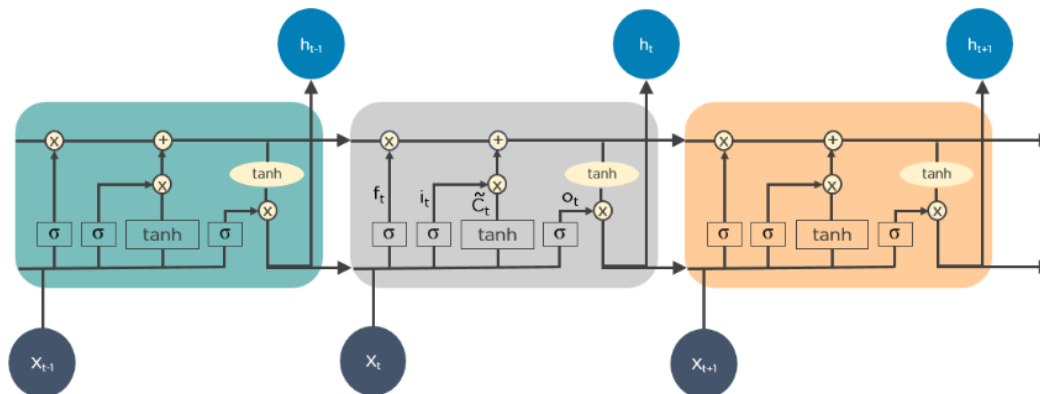
LSTMs are a type of Recurrent Neural Network (RNN) that can learn and memorize long-term dependencies. Recalling past information for long periods is the default behavior.

LSTMs retain information over time. They are useful in time-series prediction because they remember previous inputs. LSTMs have a chain-like structure where four interacting layers communicate in a unique way. Besides time-series predictions, LSTMs are typically used for speech recognition, music composition, and pharmaceutical development.

How Do LSTMs Work?

- First, they forget irrelevant parts of the previous state
- Next, they selectively update the cell-state values
- Finally, the output of certain parts of the cell state

Below is a diagram of how LSTMs operate:



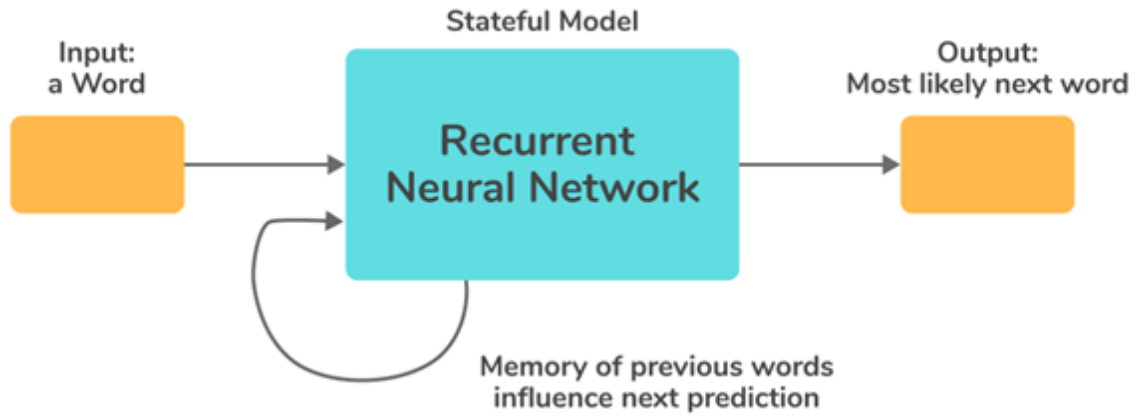
Recurrent Neural Networks (RNNs):

Recurrent Neural Network (RNN):

Tekrarlayan Sinir Ağları:

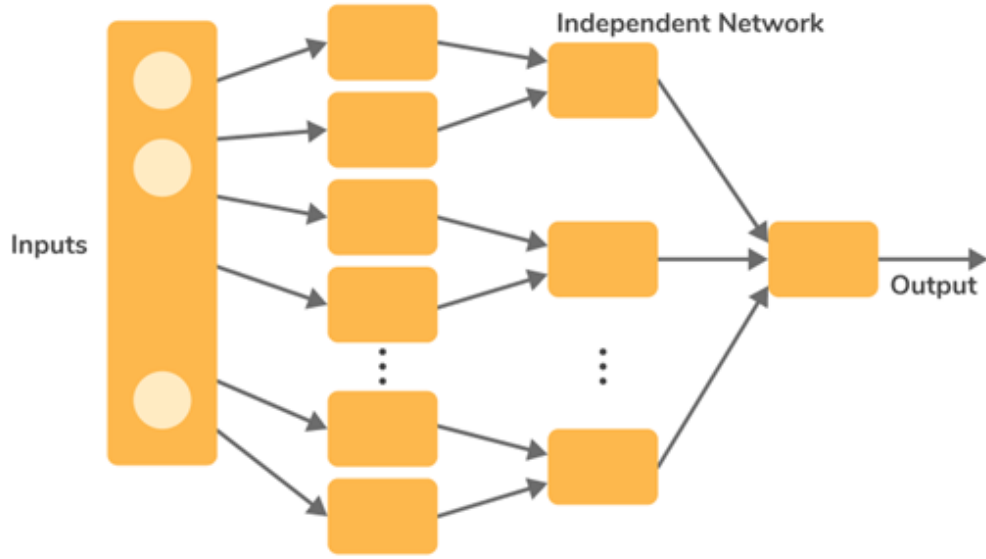
Sıralı giriş verisi zaman serisi problemini çözmek için Tekrarlayan Sinir Ağları oluşturuldu. RNN'nin girişi, mevcut giriş ve önceki örneklerden oluşur. Sonuç olarak, düğüm bağlantıları yönlendirilmiş bir grafik oluşturur. Ayrıca, bir RNN'deki her nöronun, önceki örneklerin hesaplamalarından elde edilen bilgileri depolayan bir dahili belleği vardır. RNN modelleri, değişken giriş uzunluğuna sahip verileri işlemedeki üstünlüklerinden dolayı, doğal dil işlemede (NLP) yaygın olarak kullanılır. Bu durumda AI'nın amacı, doğal dil modelleme, kelime yerleştirme ve makine çevirisi gibi insan tarafından konuşulan doğal dilleri anlayabilen bir sistem oluşturmaktır.

Bir RNN'deki her ardışık katman, ağırlıklı çıktı toplamalarının ve önceki durumun doğrusal olmayan fonksiyonlarından oluşur. Sonuç olarak, RNN'nin temel birimi "hücre" olarak adlandırılır ve her hücre katmanlardan ve tekrarlayan sinir ağı modellerinin sıralı olarak işlenmesine izin veren bir dizi hücreden oluşur.



Modular Neural Network:

Bu ağ, tek bir ağ olmaktan ziyade çok sayıda küçük sinir ağlarından oluşur. Alt ağlar, ortak bir hedefe ulaşmak için bağımsız olarak çalışan daha büyük bir sinir ağı oluşturmak için birleşir. Bu ağlar, büyük-küçük bir sorunu daha küçük parçalara bölmek ve sonra onu çözmek için son derece kullanışlıdır.



Sequence to Sequence Model:

Sıradan Sıraya Model:

Çoğu durumda, bu ağ iki RNN ağından oluşur. Ağ, kodlama ve kod çözmeye dayalıdır; bu, girdiyi işleyen bir kodlayıcıya ve çıktıyı işleyen bir kod çözücüye sahip olduğu anlamına gelir. Bu ağ türü, giren metnin uzunluğu, çıkan metnin uzunluğundan farklı olduğunda, genellikle metin işleme için kullanılır.

Deep Belief Networks (DBNs):

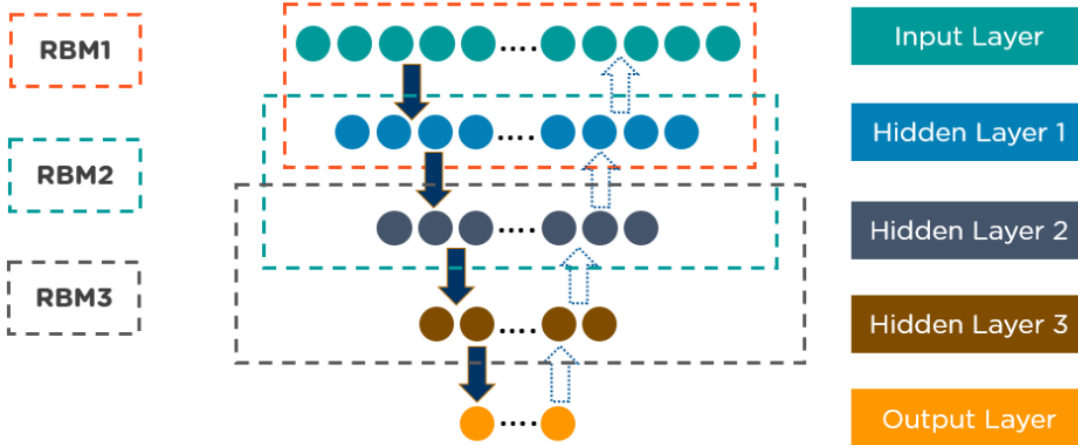
DBN'ler, birden çok stokastik, gizli değişken katmanından oluşan üretken modellerdir. Gizli değişkenler ikili değerlere sahiptir ve genellikle gizli birimler olarak adlandırılır.

DBN'ler, katmanlar arasında bağlantıları olan bir Boltzmann Makinesi yığınıdır ve her bir RBM katmanı, hem önceki hem de sonraki katmanlarla iletişim kurar. Derin İnanç (Kanı) Ağları (DBN'ler) görüntü tanıma, video tanıma ve hareket yakalama verileri için kullanılır.

DBN'ler Nasıl Çalışır?

- Açgözlü öğrenme algoritmaları DBN'leri eğitir. Açgözlü algoritma, her aşamada yerel olarak en uygun seçimi yapma problem çözme buluşsal yöntemini izleyen herhangi bir algoritmadır. Açgözlü öğrenme algoritması, yukarıdan aşağıya, üretken ağırlıkları öğrenmek için katman katman bir yaklaşım kullanır.
- DBN'ler, Gibbs örneklem adımlarını en üstteki iki gizli katmanda çalıştırır. Bu aşama, en üstteki iki gizli katman tarafından tanımlanan RBM'den bir örnek alır.
- DBN'ler, modelin geri kalanı boyunca tek bir atasal örneklem geçişini kullanarak görünür birimlerden bir örnek alır.
- DBN'ler, her katmandaki gizli değişkenlerin değerlerinin aşağıdan yukarıya tek bir geçişle çıkarılabileceğini öğrenir.

Below is an example of DBN architecture:



Autoencoders:

What are autoencoders? Explain the different layers of autoencoders.

Otomatik kodlayıcılar nelerdir? Otomatik kodlayıcıların farklı katmanlarını açıklayın.

Otomatik kodlayıcı, çıktı katmanının giriş katmanıyla aynı boyuta sahip olması koşuluyla bir tür sinir ağıdır. Başka bir deyişle, çıktı katmanındaki çıktı birimlerinin sayısı, girdi katmanındaki girdi birimlerinin sayısına eşittir. Otomatik kodlayıcı, girdiden çıktıya verileri denetimsiz bir şekilde çoğalttığı için çoğaltıcı sinir ağı olarak da bilinir.

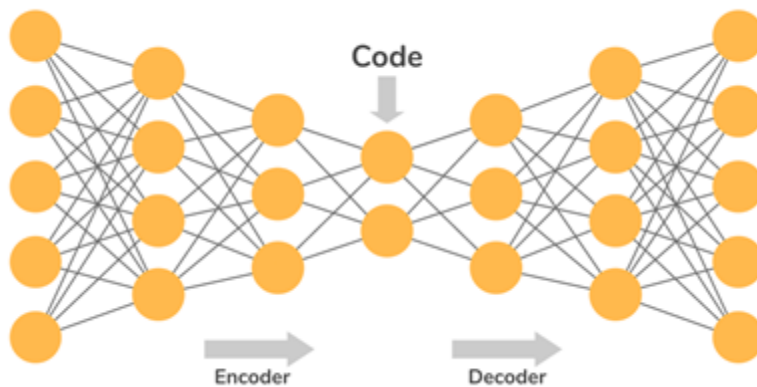
Giriş ağı üzerinden göndererek, otomatik kodlayıcılar girişin her bir boyutunu yeniden oluşturur. Bir girdiyi çoğaltmak için bir sinir ağı kullanmak basit görünebilir, ancak çoğaltma işlemi sırasında girdinin boyutu küçülür, bu da daha küçük bir temsille sonuçlanır. Giriş ve çıkış katmanlarına kıyasla, sinir ağının orta katmanları daha az birime sahiptir. Sonuç olarak, girdinin azaltılmış temsili orta katmanlarda depolanır. Girdinin bu azaltılmış temsili, çıktıyı yeniden oluşturmak için kullanılır.

Otomatik kodlayıcıların mimarisindeki farklı katmanlar şunlardır:

Kodlayıcı: Kodlayıcı, giriş görüntüsünü bir gizli uzay temsiline sıkıştıran ve onu daha düşük bir boyutta sıkıştırılmış bir temsil olarak kodlayan, tamamen bağlı, ileri beslemeli bir sinir ağıdır. Orijinal görüntünün deforme olmuş temsili sıkıştırılmış görüntüdür.

Kod: Kod çözücüye sağlanan girişin azaltılmış gösterimi ağı bu bölümünde saklanır.

Kod çözücü: Kodlayıcı gibi, kod çözücü de kodlayıcıyla aynı yapıya sahip ileri beslemeli bir ağıdır. Bu ağı, koddan gelen girdiyi orijinal boyutlarına yeniden monte etmekten sorumludur.



Yukarıdaki resimde gördüğümüz gibi, girdi kodlayıcıda sıkıştırılır, ardından Kodda saklanır ve ardından orijinal girdi kod çözücü tarafından koddan açılır. Otomatik kodlayıcının temel amacı, girdiyle aynı olan bir çıktı sağlamaktır.

Mention the applications of autoencoders.

Otomatik kodlayıcıların uygulamalarından bahsedin.

Otomatik kodlayıcıların uygulamaları şunlardır: -

Görüntü Gürültüsünü Giderme: Görüntüleri gürültüden arındırma, otomatik kodlayıcıların çok başarılı olduğu bir beceridir. Gürültülü bir görüntü, bozulmuş veya içinde az miktarda parazit (yani, görüntülerde rastgele parlaklık veya renk bilgisi değişimi) bulunan görüntüdür. Görüntü denoising, görüntünün içeriği hakkında doğru bilgi elde etmek için kullanılır.

Boyutsallık Azaltma: Giriş, kod adı verilen orta katmanda depolanan otomatik kodlayıcılar tarafından azaltılmış bir temsile dönüştürülür. Bu, girdiden gelen bilgilerin sıkıştırıldığı yerdir ve artık her düğüm, bu katman modelden çıkarılarak bir değişken olarak ele alınabilir. Sonuç olarak, kod çözücüyü kaldırarak, çıktı olarak kodlama katmanı ile boyut azaltma için bir otomatik kodlayıcının kullanılabileceği sonucuna varabiliriz.

Özellik Çıkarma: Otomatik Kodlayıcıların kodlama bölümü, yeniden yapılandırma hatasını azaltarak giriş verilerinde bulunan önemli gizli özelliklerin öğrenilmesine yardımcı olur. Kodlama sırasında, orijinal özellik kombinasyonlarından oluşan yeni bir koleksiyon oluşturulur.

Görüntü Renklendirme: Siyah beyaz bir görüntüyü renkli bir görüntüye dönüştürmek, otomatik kodlayıcıların uygulamalarından biridir. Renkli bir görüntüyü gri tonlamaya da dönüştürebiliriz.

Veri Sıkıştırma: Veri sıkıştırma için otomatik kodlayıcılar kullanılabilir. Yine de, aşağıdaki nedenlerden dolayı nadiren veri sıkıştırma için kullanılırlar: Kayıplı sıkıştırma: Otomatik kodlayıcının çıkışı girişle aynı değildir, ancak yakın fakat bozulmuş bir temsildir. Kayıpsız sıkıştırma için en iyi seçenek değildirler. Veriye özel: Otomatik kodlayıcılar, yalnızca eğitildikleri verilerle aynı olan verileri sıkıştırabilir. Sağlanan eğitim verileriyle ilgili özellikleri öğrenmeleri bakımından jpeg veya gzip gibi geleneksel veri sıkıştırma algoritmalarından farklıdır. Sonuç olarak, elle yazılmış rakamlarla eğitilmiş bir otomatik kodlayıcı tarafından sıkıştırılacak bir manzara fotoğrafını tahmin edemeyiz.

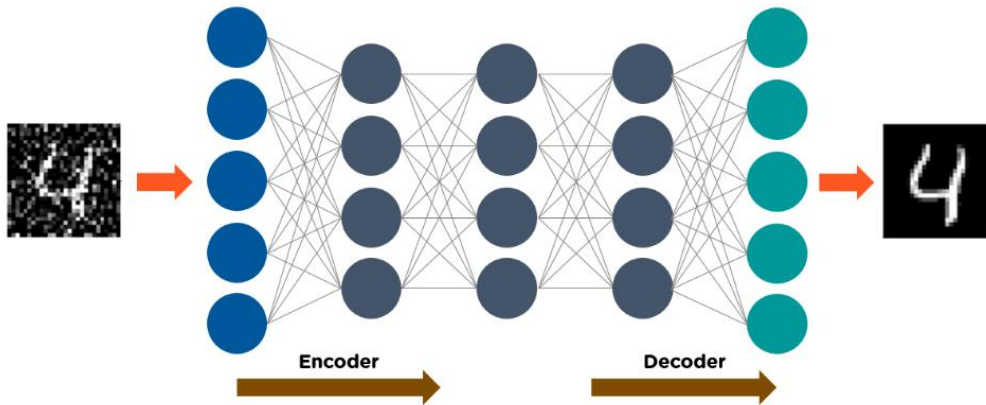
Otomatik kodlayıcılar, giriş ve çıkışın aynı olduğu belirli bir ileri beslemeli sinir ağı türüdür. Geoffrey Hinton, 1980'lerde denetimsiz öğrenme sorunlarını çözmek için otomatik kodlayıcılar tasarladı. Verileri girdi katmanından çıktı katmanına kopyalayan eğitilmiş sinir ağlarıdır. Otomatik kodlayıcılar, farmasötik keşif, popülerlik tahmini ve görüntü işleme gibi amaçlar için kullanılır.

Otomatik Kodlayıcılar Nasıl Çalışır?

Bir otomatik kodlayıcı üç ana bileşenden oluşur: kodlayıcı, kod ve kod çözücü.

- Otomatik kodlayıcılar, bir girdi alacak ve onu farklı bir temsile dönüştürecek şekilde yapılandırılmıştır. Daha sonra orijinal girdiyi mümkün olduğu kadar doğru bir şekilde yeniden oluşturmaya çalışırlar.
- Bir rakamın görüntüsü net olarak görülmediğinde, otomatik kodlayıcı sinir ağına beslenir.
- Otomatik kodlayıcılar önce görüntüyü kodlar, ardından girdinin boyutunu daha küçük bir temsile küçültür.
- Son olarak, otomatik kodlayıcı, yeniden oluşturulmuş görüntüyü oluşturmak için görüntünün kodunu çözer.

Aşağıdaki resim, otomatik kodlayıcıların nasıl çalıştığını gösterir:



7. Genetik Algoritmalar

Genetik algoritma (GA), belirli bir soruna iyi çözümler bulma umuduyla yeni genotipler üretmek için mutasyon ve çaprazlama gibi yöntemleri kullanarak doğal seleksiyon sürecini taklit eden bir arama algoritması ve sezgisel tekniktir. Makine öğrenmesinde, 1980'ler ve 1990'larda genetik algoritmalar kullanılmıştır. Tersine, genetik ve evrimsel algoritmaların performansını arttırmak için makine öğrenme teknikleri kullanılmıştır.

Genetik Algoritma yaklaşımının ortaya çıkışı 1970 'lerin başında olmuştur . 1975 'te John Holland'ın makine öğrenmesi üzerine yaptığı çalışmalarda canlılardaki evrimden ve değişimden etkilenecek, bu genetik evrim sürecini bilgisayar ortamına aktarması ve böylece bir tek mekanik yapının öğrenme yeteneğini geliştirmek yerine, çok sayıdaki böyle yapıların tamamını “ çoğalma, değişim gibigenetik süreçler sonunda üstün yeni bireylerin elde edilebileceğini gösteren çalışmasından çıkan sonuçların yayınlanmasından sonra geliştirdiği yöntemin adı “Genetik Algoritmalar” olarak tanınmıştır.

Genetik Algoritmalar, süreçleri model olarak kullanan problem çözme teknikleridir. Öğrenen Makina Algoritmaları ise tecrübe ile ortaya çıkan yeni duruma ya da belirsizliğe ilişkin parameterlerin otomatik olarak belirlendiği matematiksel modellerdir.

Genetik Algoritmalar mümkün olan çözümlerin bir popülasyonu üzerinde işlem yapan olasılıklı (stochastic) arama algoritmalarıdır. Geleneksel programlama teknikleriyle çözülmesi güç olan, özellikle sınıflandırma ve çok boyutlu optimizasyon problemleri, bunların yardımıyla daha kolay ve hızlı olarak çözüme ulaştırılmaktadır. Genetik algoritmalar doğada geçerli olan en iyinin yaşaması kuralına dayanarak sürekli iyileşen çözümler üretir. Bunun için “iyi”nin ne olduğunu belirleyen bir uygunluk fonksiyonu ve yeni çözümler üretmek için yeniden kopyalama, değiştirme gibi operatörleri kullanır. Görevler için programın ölçülen performansı tecrübe ile artıyor ise bu program tecrübe ile öğreniyor denilebilir. Genetik algoritmalar robot kontrolü için kural kümelerinin öğrenilmesi ve yapay sinir ağları için topoloji ve öğrenme parametrelerinin optimize edilmesi için kullanılmaktadır.

Göçmen kuşlar, her yıl çıktıkları yolculukta adreslerini hiç şaşırmadan, hiçbir duraklarını atlamadan, yüzyıllardır sürekli hareket halindeler; üstelik pusulasız ve haritasız... Dünyadaki on binlerce kuş türünün sadece sekiz bin kadarı her yıl binlerce kilometre kat ederek nesillerini sürdürmeye çalışır. Kimileri sürüler halinde, kimileri küçük gruplar halinde ömürlerinin sonuna kadar kendilerini bu uzun yolculuğa adanlar. Bu kuşların tek bir ortak amaçları vardır: Üremek ve beslenmek. Göçmen kuşların uzun uçuşlarına iç güdüsel olarak hazırlandıklarını da söyleyebiliriz. Öyle ki, genlerinde uzak diyarlara göç etme nitelikleri bulunan hayvanat bahçelerinde, ya da kafeste beslenen kuşlar göç vakti geldiklerinde kıpır kıpır oldukları gözlenir.

Göçmen kuşların her yıl binlerce kilometre süren yolculukları sırasında yönlerini kaybolmadan nasıl tayin edebildikleri üzerine yapılan araştırmalarda ortaya sadece mantıksal tahminler atılmış durumda. Örneğin, kuşların tanıdık kara parçalarını ezberledikleri, sürekli sahil şeridini takip ettikleri gibi var sayımlar bir yere kadar doğru ama uzun deniz yolculukları yapanlar bu varsayımları yerle bir ediyor. Diğer taraftan gece uçuşu yapanlar içinse yıldızların onlara yol gösterdiği düşünülüyor. Bir diğer var sayım ise göçmen kuşların, dünyanın manyetik alan çizgilerini takip ederek kaybolmadan yönlerini bulabildikleri şeklinde. Modern çağın nimetlerinden biri olan GPS sayesinde işaretlenen kimi göçmen kuşların izledikleri rotalar her yıl tıpa tıp aynı. Hatta inanılması güç ama, her yıl mola verdikleri yerler bile şaşmıyor.

Göçmen kuşların hayranlıkla izlenebilecek bir başka özellikleri de havada çizdikleri V şeklindeki düzen olsa gerek. Öyle ya, ancak bu şekilde onları diğer sürülerden net bir şekilde ayırt edebiliyoruz. Bu uçuş düzeninin iki önemli avantajı var. İlki, en öndeki lider kuşu görebilmek. İkincisi ve tabi ki daha önemli olanı ise hemen öndeki kuşun yarattığı hava akımından yararlanarak daha az enerji harcayarak uçmak. Kuşların bu şekilde uçmaları sayesinde tahminen yüzde 20'lik bir enerji tasarrufu sağladıklarına inanılıyor. Eğer havada bu tür bir V oluşumu yakalarsanız, kuşların yerlerini sürekli değiştirdiğine de tanık olacaksınız. Özellikle de en önde uçarak en fazla yorulan lider kuşa eşlik edenlerin onun yerini nasıl almaya çalıştıklarını göreceksiniz.

Geleneksel programlama teknikleriyle çözülmesi güç olan, özellikle sınıflandırma ve çok boyutlu optimizasyon problemleri, bunların yardımıyla daha kolay ve hızlı olarak çözüme ulaştırılmaktadır. Genetik algoritmalar doğada geçerli olan en iyinin yaşaması kuralına dayanarak sürekli iyileşen çözümler üretir. Bunun için "iyi"nin ne olduğunu belirleyen bir uygunluk fonksiyonu ve yeni çözümler üretmek için yeniden kopyalama, değiştirme gibi operatörleri kullanır. Görevler için programın ölçülen performansı tecrübe ile artıyor ise bu program tecrübe ile öğreniyor denilebilir. Genetik algoritmalar robot kontrolü için kural kümelerinin öğrenilmesi ve yapay sinir ağları için topoloji ve öğrenme parametrelerinin optimize edilmesi için kullanılmaktadır.

Genetik algoritmalar, evrim teorisinin dayandığı temel prensiplerinden olan doğal seçim ile en iyi bireylerin hayatta kalması ilkesini taklit eden bir tekniktir. Burada yapılan, en iyi çözümün pek çok çözüm seçeneği içinden arama yapılarak belirlenmesidir. Rassal arama teknikleri ile eldeki mevcut çözümlerden hareketle en iyi çözüme ulaşılmaya çalışılmaktadır. Basit bir genetik algoritmanın işlem adımları; problemin olası çözümlerinin dizilere (kromozomlar) kodlanarak çözüm yığınının oluşturulması, kromozomların çözüme yaklaşma başarısının uygunluk fonksiyonu ile değerlendirilmesi, genetik parametrelerin belirlenmesi, seçim stratejisi ve mekanizmaları, genetik operatörler ve durdurma kriteri olarak sıralanabilir.

Genetik algoritmalar, bilinen yöntemlerle çözülemeyen veya çözüm süresi problemin büyüklüğüne göre oldukça fazla olan problemlerde, kesin sonuca çok yakın sonuçlar verebilen bir yöntemdir. Bu özelliği ile, NP (Nonpolynomially-Polinomal olmayan) problemler yanında gezgin satıcı, karesel atama, yerleşim, atölye çizelgeleme, mekanik öğrenme, üretim planlama, elektronik, finansman ve hücresel üretim gibi konularda uygulanmaktadır.

GEN: Kendi başına anlamı olan ve genetik bilgi taşıyan en küçük genetik birimdir.

- Bir gen A, B gibi bir karakter olabileceği gibi 0 veya 1 ile ifade edilen bir bit veya bit dizisi olabilir. Örneğin bir cismin x koordinatındaki yerini gösteren bir gen 101 şeklinde ifade edilebilir

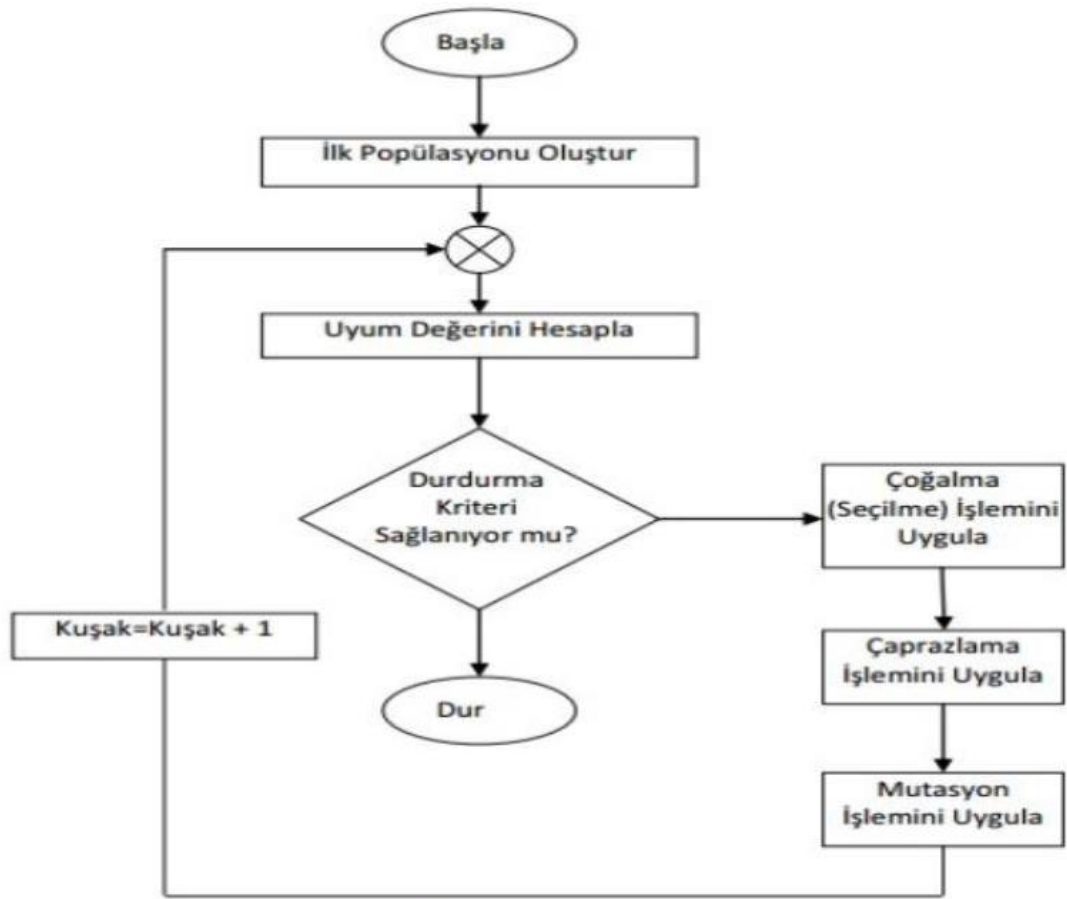
KROMOZOM: Bir ya da birden fazla genin bir araya gelmesiyle oluşurlar Probleme ait tüm bilgileri içerirler. Kromozomlar toplumdaki bireyler yada üyelere karşılık gelirler. Ele alınan problemde alternatif çözüm adaydır.

Örneğin, kromozom bir problemde aç, boyut ve koordinat değişkenlerinden veya bir dikdörtgen prizmasının ölçülerinden (genişlik, derinlik) oluşabilir 001 101 111 1 5 7 değerleri kromozomu oluşturan genlerdir. Genetik algoritma işlemlerinde kromozomları kullandığı için kromozom tanımları çok iyi ifade edilmelidir.

POPÜLASYON: Kromozomlar veya bireyler topluluğudur. Popülasyon üzerinde durulan problem için alternatif çözümler kümesidir. Aynı anda bir popülasyonda ki birey sayısı sabit ve probleme göre kullanıcı tarafından belirlenir. Zayıf olan bireylerin yerini kuvvetli yeniler almaktadır.

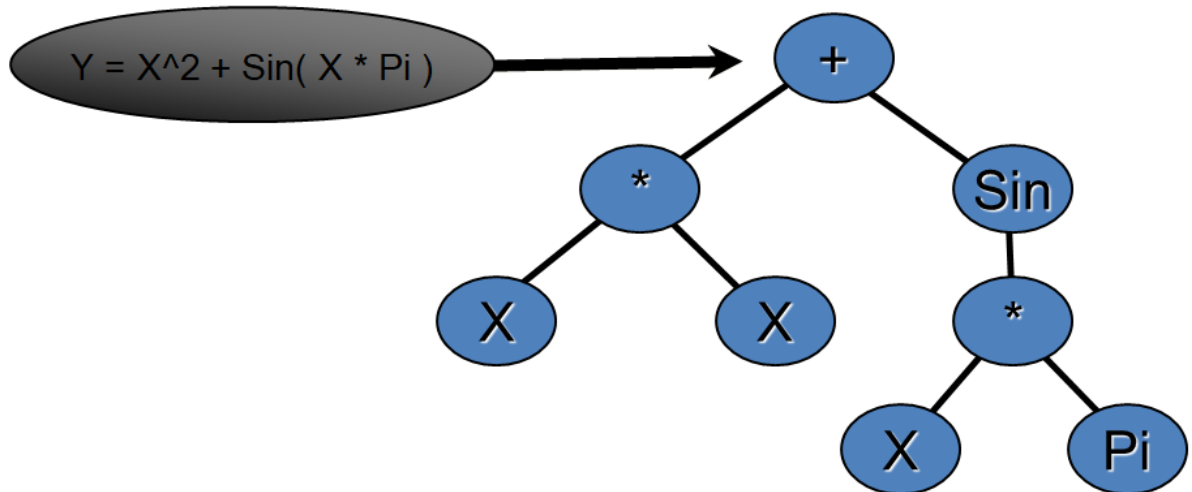
Genetik Algoritmalar Nasıl Çalışır?

1. Adım: Olası çözümlerin kodlandığı bir çözüm grubu oluşturulur. Çözüm grubuna biyolojideki benzerliği nedeniyle populasyon çözümlerin kodları da kromozom olarak adlandırılır. Bu adıma popülasyonda bulunan birey sayısı belirleyerek başlanır. Bu sayı için bir standart yoktur. Genel olarak önerilen 100 - 300 aralığında bir büyüklüktür. Büyüklük seçiminde yapılan işlemlerin karmaşıklığı ve aramanın derinliği önemlidir. Popülasyon bu işlemten sonra rasgele oluşturulur.
2. Adım: Her kromozomun ne kadar iyi olduğu bulunur Kromozomların ne kadar iyi olduğunu bulan fonksiyona uygunluk fonksiyonu denir Bu fonksiyon işletilerek kromozomların uygunluklarının bulunmasına ise hesaplama(evalution adı verilir Bu fonksiyon genetik algoritmanın beynini oluşturmaktadır GA da probleme özel çalışan tek kısım bu fonksiyondur. Çoğu zaman GA nın başarısı bu fonksiyonun verimli ve hassas olmasına bağlı olmaktadır



Genetik algoritmanın akış diyagramı.

Genetic Programming Background:



Crossover: bir üst programdaki bir makinedeki işlem sırasını başka bir ana makinedeki başka bir makinedeki işlem dizisiyle birleştirin.

Example 1.

Cut Point



Example 2. Partially Mapped Crossover

Cut Point 1

Cut Point 2



Örnek: P1'den alınan işlerin mutlak konumlarını ve P2'den alınanların göreceli konumlarını korur.

Cut Point 1



Örnek: Örnek 3'e benzer ancak 2 geçiş noktası vardır.

Cut Point 1

Cut Point 2



Örnek:

Aşağıdaki ifadeyi karşılayan GA'yı kullanarak a ve b'nin optimal değerlerini tahmin edelim.

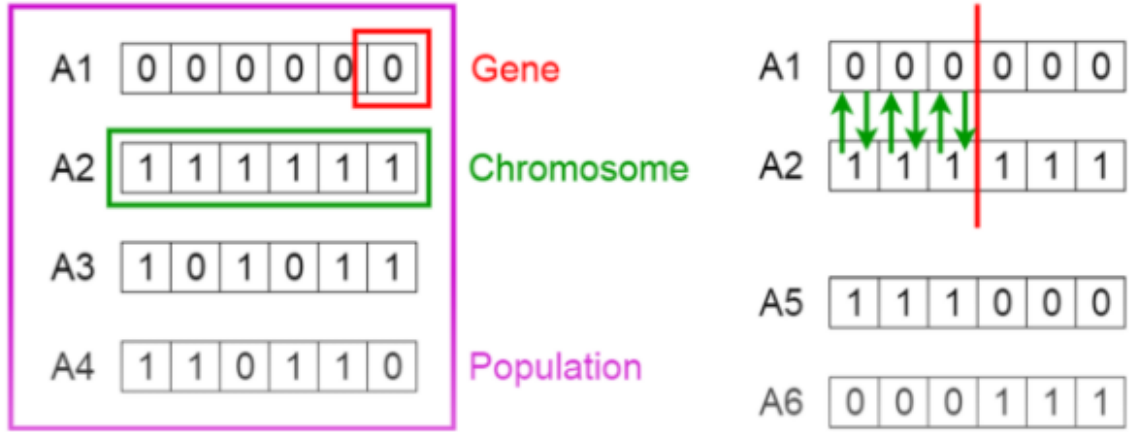
$$2a^2 + b = 57$$

Herhangi bir optimizasyon problemi, bir amaç fonksiyonuyla başlar. Yukarıdaki denklem şu şekilde yazılabilir:

$$f(a,b) = 2a^2 + b - 57$$

Fonksiyonun değerinin 0 olduğu anlaşılmaktadır. Bu fonksiyon bizim amaç fonksiyonumuzdur ve amaç a ve b değerlerini tahmin etmektir, öyle ki objektif fonksiyonun değeri sıfıra indirilir.

Örnek:



Örnek:

Adil bir jetonu 60 kez atıyoruz ve aşağıdaki ilk popülasyonu elde ediyoruz:

$s_1 = 1111010101$	$f(s_1) = 7$
$s_2 = 0111000101$	$f(s_2) = 5$
$s_3 = 1110110101$	$f(s_3) = 7$
$s_4 = 0100010011$	$f(s_4) = 4$
$s_5 = 1110111101$	$f(s_5) = 8$
$s_6 = 0100110000$	$f(s_6) = 3$

Seçimi yaptıktan sonra aşağıdaki popülasyonu elde ettiğimizi varsayalım

$s_1' = 1111010101$	(s_1)
$s_2' = 1110110101$	(s_3)
$s_3' = 1110111101$	(s_5)
$s_4' = 0111000101$	(s_2)
$s_5' = 0100010011$	(s_4)
$s_6' = 1110111101$	(s_5)

Sonra crossover için dizeleri eşleştiriyoruz. Her çift için crossover olasılığına (örneğin 0.6) göre crossover yapıp yapmamaya karar veririz. Sadece çiftler için (s_1', s_2') ve (s_5', s_6') crossover yapmaya karar verdiğimiz varsayalım. Her çift için rastgele bir geçiş noktası çıkarırız, örneğin birinci için 2 ve ikinci için 5.

Before crossover:

$s_1' = 1111010101$

$s_2' = 110110101$

$s_5' = 0100010011$

$s_6' = 111011101$

After crossover:

$s_1'' = 110110101$

$s_2'' = 1111010101$

$s_5'' = 0100011101$

$s_6'' = 1110110011$

Son adım, rastgele mutasyon uygulamaktır: yeni popülasyona kopyalayacağımız her bit için küçük bir hata olasılığına izin veriyoruz (örneğin 0.1)

Mutasyonu uygulamadan önce:

$s_1'' = 1110110101$

$$s_2'' = 1111010101$$

$$s_3'' = 1110111101$$

$$s_4'' = 0111000101$$

$$s_5'' = 0100011101$$

$$s_6'' = 1110110011$$

After applying mutation:

$$s_1''' = 1110100101 \quad f(s_1''') = 6$$

$$s_2''' = 1111110100 \quad f(s_2''') = 7$$

$$s_3''' = 1110101111 \quad f(s_3''') = 8$$

$$s_4''' = 0111000101 \quad f(s_4''') = 5$$

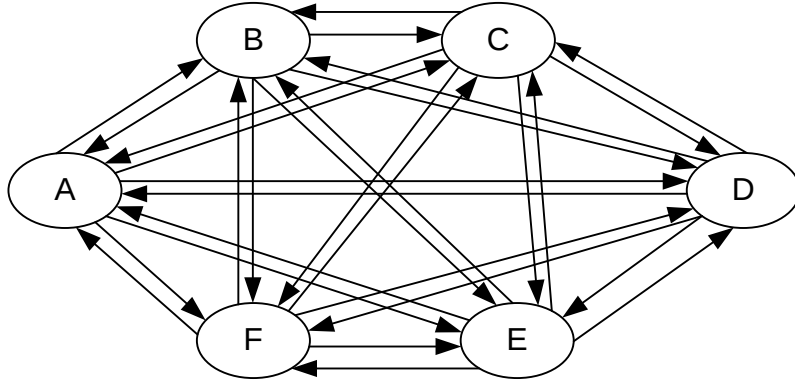
$$s_5''' = 0100011101 \quad f(s_5''') = 5$$

$$s_6''' = 1110110001 \quad f(s_6''') = 6$$

Bir nesilde, toplam popülasyon uygunluğu 34'ten 37'ye değişti, böylece ~% 9 arttı. Bu noktada, bir durdurma kriteri karşılanana kadar aynı süreci yeniden yaşıyoruz.

Örnek:

A, B, C, D, E, F olmak üzere 6 durumum olsun. Her bir durumdan(genden) diğer bütün durumları (genlere) geçiş mümkün olsun. Tüm durumlar bir defa muhakkak ziyaret edilecek. O halde $5!=120$ tur mümkündür. Tur, her zaman A'dan başlayacak bütün durumlar dolaşılacak sonunda yeniden A'ya dönülecek. Durumlar arasında rasgele bir rota oluşturalım. ABDFECA.



Bu problem Genetik Algoritma ile çözülürken,

- Uygunluk fonksiyonu seçilir. Uygunluk fonksiyonu mesafeler olacak, mesafeyi en aza indirilmesi gerekiyor.
- Popülasyon büyüklüğü, kaç çözümden oluşacak, mesela 10 seçilebilir.
- Ebeveyn seçimi, çözüm uzayındaki farklı durumları kullanarak yeni bir rota oluşturulacak. (Rulet tekerleği, sıralı yöntem)
- Çaprazlama, iki turdan yeni bir tur oluşturulurken çarpazlama iki noktalı mı yoksa tek noktalı mı?
- Mutasyon oranı, yeni tur oluşturulduktan sonra bir veya iki durumun (gen) rasgele değiştirilmesi
- Bitirme Şartı, iterasyon sayısının belirli bir sayıya ulaşması mı, uygunluk fonksiyonun belirli bir değerinin altında veya üstünde olması mı karar verilir.

Öncelikle durumlar (genler) arasında rasgele 2 çözüm turu oluşturalım. 6 durum arasında oluşturulacak tur sayısı= $5!=120$ farklı tur oluşturulabilir.

Tur-1: A B D F E C A

Tur-2: A C E D F B A

Bu iki tur kendi aralarında çaprazlamaya tabi tutulur. Çaprazlama oranı %50 olsun ve tek noktalı olarak yapılsın.

Tur-1: A B D **F E C A**

Tur-2: A C E **D F B A**

Yeni tur oluşturulurken birinci turdan ilk kısım ikinci turdan ise ikinci kısım alınır. İkinci kısımda olanlar birinci kısımda var ise alınmaz, yerine ikinci turundan başından itibaren olmayanlar alınır.

Tur-3: A B D ~~D~~ F ~~B~~ A = A B D F C E A olur.

Mutasyonu yaparken tek noktalı bir durumun (genin) değiştirilsin. Yeni oluşan Turda rasgele bir durum (gen) değiştirilsin. Sözelimi B'yi değiştirelim ve E olsun. Yeni tur (kromozom),

Tur-4: A E D **F C B A** olur. E durumu iki adet olamayacağı için sağ taraftaki E B ile değiştirilir. Böylece mutasyondan sonraki turda elde edilmiş olur.

Bu aşamadan sonra uygunluk fonksiyonu hesaplanır. Popülasyon içerisindeki diğer çözümlerden daha kısa mesafe ise diğer çözümlerden daha iyi çözüm olarak kabul edilir. Eğer değilse öldürülür.

Adımları sıralarsak,

- 10 tane başlangıç çözümünü oluşturulur.
- Herbir çözüm için uygunluk fonksiyonu değerini hesaplanır.
- Başlangıç çözümlerinden iki tane ebeveyn seçilir.
- Ebeveynler üzerinde çaprazlama yapılır.
- Mutasyona tabi tutulur.
- Yeni bireyin uygunluk fonksiyon değeri hesaplanır.
- Yeni birey öldürülecek mi yoksa popülasyona katılacak mı karar vermek için seleksiyon işlemi yapılır.
- Bitirme şartı sağlandı mı diye kontrol edilir, sağlandıysa en iyi çözüm al ve çık.

8. Python Yükleme

- 1- python.org
Download
All Releases
Python releases by version number
Python 3.9.13
Not: setup kurulumunda path seçme dahil iki kutucuğu da onaylayın.
- 2- <https://www.anaconda.com/products/distribution>
Download(uzun sürer)
Setup install edilirken, kutucukların seçimine dikkat edin, ortak path seçimi yapın
Anaconda Navigator kurulumu yapın
- 3- Editör
<https://code.visualstudio.com/>
Download for Windows
- 4- Çalıştırılırken
Anaconda Navigator, başlat menüsünden yazılarak enter yapılır
VS Code seçilir. Launch yapılır.

Pandas

Pandas, veri işlemesi ve analizi için Python programlama dilinde yazılmış olan bir yazılım kütüphanesidir. Bu kütüphane temel olarak zaman etiketli serileri ve sayısal tabloları işlemek için bir veri yapısı oluşturur ve bu şekilde çeşitli işlemler bu veri yapısı üzerinde gerçekleştirilebilir olur. Yazılım ücretsizdir ve bir çeşit lisansına sahiptir. Yazılım ismini bir ekonometri terimi olan veri panelinden almıştır. Bir veri paneli birçok zaman aralığı içinde farklı gözlemlerin işlenebildiği yapıyı tarif eder.

Kütüphane özellikleri

- İndeksli DataFrame (veri iskeleti) objeleri ile veri işlemesi yapabilmek.
- Hafızadaki veya farklı türlerde bulunan veriyi okuyabilmek ve yazabilmek için araçlar sağlamak.
- Veri sıralama ve bütünleşik kayıp veri senaryolarına karşı esnek imkanlar sunması.
- Veri setlerinin tekrar boyutlandırılması veya döndürülmesi.
- Etiket bazlı dilimleme, özel indeksleme ve büyük veri setlerini ayrıştırma özelliği.
- Veri iskeletine sütun ekleme veya var olan sütunu çıkarma.
- Veri gruplama özelliği ile ayırma-uygulama-birleştirme uygulamalarının yapılabilmesi.
- Veri setlerinin birleştirilmesi ve birbirine eklenmesi.

- Hiyerarşik eksenleri indeksleme özelliğiyle birlikte çok boyutlu veriden, daha az boyutlu veri elde edilebilmesi.
- Zaman serisi özelliği: Zaman aralığı oluşturma[4] ve sıklık çevrimleri yapma, hareketli aralık istatistik fonksiyonları, tarih öteleme ve geciktirme.
- Veri filtrelemesi yapabilmek.

Kütüphane performans konusunda yüksek derecede en iyilenmiştir. Bu yüzden kütüphanenin önemli parçaları CPython ve C üzerinde yazılmışlardır.

Dataframes (Veri İskeletleri):

Pandas temel olarak makine öğrenmesi uygulamalarında kullanılmaktadır. Bu uygulamalarda en öne çıkan özelliği de veri isketleridir. Pandas ayrıca birçok farklı formattan (csv, excel gibi) veri içe aktarması gerçekleştirebilir. Pandas çok farklı veri işleme yöntemlerini uygulayabilir; örneğin grupta, ekleme, birleştirme, kaynaştırma, bir araya getirme. Ayrıca bu kütüphane veri temizleme için veri doldurma, değiştirme ve varsayma özelliklerine de sahiptir.

9. Yararlanılan Kaynaklar

1. Machine Learning (ML) Algorithms For Beginners with Code Examples in Python.
<https://medium.com/towards-artificial-intelligence/machine-learning-algorithms-for-beginners-with-python-code-examples-ml-19c6afd60daa>
2. https://www.w3schools.com/python/python_ml_getting_started.asp
3. https://www.tutorialspoint.com/machine_learning_with_python/knn_algorithm_finding_nearest_neighbors.htm
4. <https://brilliant.org/wiki/feature-vector/>
5. <https://stanford.edu/~shervine/l/tr/teaching/cs-229/>
6. <https://stanford.edu/~shervine/l/tr/teaching/cs-221/>
7. <https://stanford.edu/~shervine/l/tr/teaching/cs-230/>