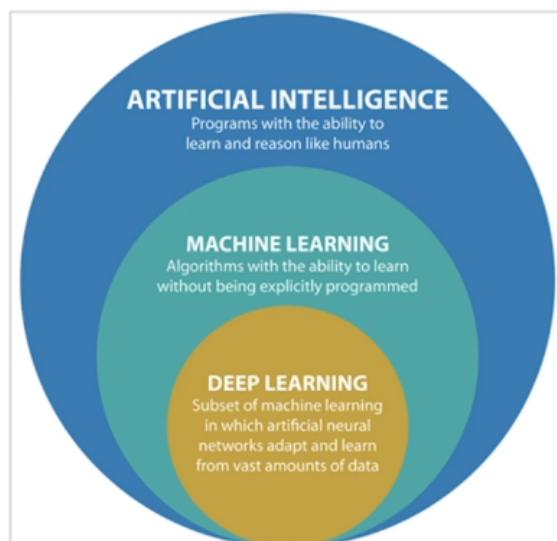
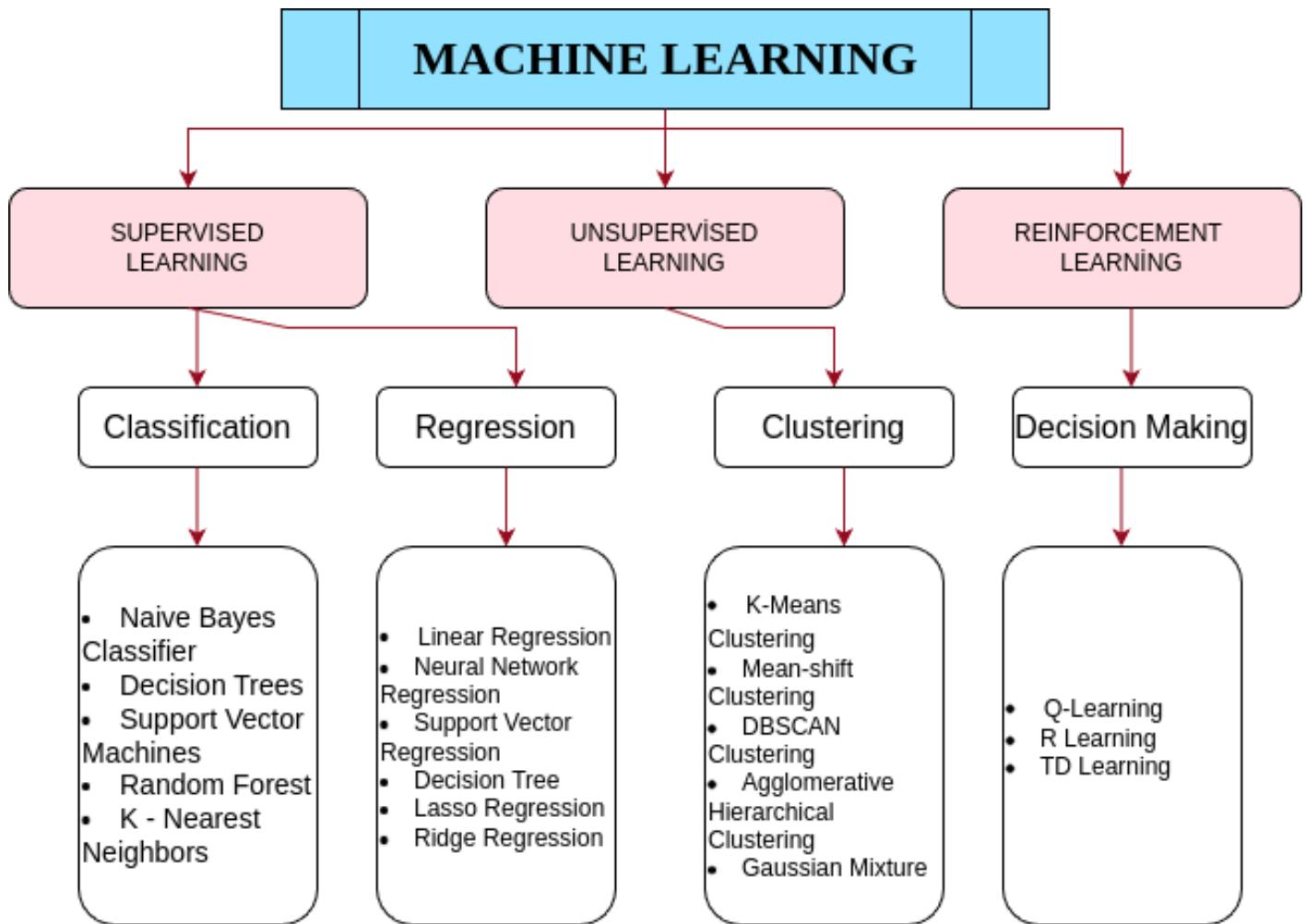


# **MACHINE LEARNING**



## k fold cross validation



1. Eğitim verilerini K eşit parçaya bölün
2. Modeli  $k-1$  parçaya uydurun ve  $k$ 'inci parçadaki uydurulmuş modeli kullanarak test hmasını hesaplayın
3. Her veri alt kümesini bir kez test kümesi olarak kullanarak  $k$  kez tekrarlayın.

## Cost (maliyet) and Loss (kayıp) Function

**Cost function :** Tüm veri kümesine sahip ortalama hata

Maliyet Fonksiyonu:

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2$$

Kuralların güncellenmesi:

$$\theta_0 := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$$

$$\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$

Türevi:

$$\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})$$

$$\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) \cdot x^{(i)}$$

### 1) Regresyon Problemleri için (Loss) Kayıp Fonksiyonları:

- Ortalama Karesel Hata (Mean Squared Error - MSE):

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

Bu fonksiyon, büyük hataları daha ağır cezalandırır çünkü farkın karesi alınır.

- **Ortalama Mutlak Hata (Mean Absolute Error - MAE):**

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

Bu fonksiyon, hataları eşit olarak cezalandırır.

- **Root Mean Squared Error (RMSE) - Kök Ortalama Karesel Hata**

$$RMSE = \sqrt{\frac{1}{n} \sum_{t=1}^n e_t^2}$$

Hataları daha kolay yorumlamayı sağlar ve tahmin edilen değerin biriminde bir sonuç verir.

## 2) Sınıflandırma Problemleri için Kayıp Fonksiyonları:

- **Binary Cross-Entropy (İkili Çapraz Entropi):**

$$= -\frac{1}{n} \sum_{i=1}^n [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)]$$

Burada  $y_i$  gerçek etiket (0 veya 1),  $\hat{y}_i$  ise modelin tahmin ettiği olasılıktır.

- **Categorical Cross-Entropy (Kategorik Çapraz Entropi):**

$$= -\sum_{i=1}^n \sum_{j=1}^k y_{ij} \log(\hat{y}_{ij})$$

Burada  $y_{ij}$  gerçek sınıfın ikili değeri (yani, doğru sınıf için 1, diğerleri için 0) ve  $\hat{y}_{ij}$  modelin sınıf  $j$  için tahmin ettiği olasılıktır.

- **Log Loss (Logaritmik Kayıp)**

$$LogLoss = \frac{1}{m} \sum_{i=1}^m (-y_i \cdot \log(p(\hat{y}_i)) - (1 - y_i) \log(1 - p(\hat{y}_i)))$$

Modelin kesin olasılık tahminlerini değerlendirmek için kullanılır. Düşük bir Log Loss, modelin daha iyi performans gösterdiğini gösterir. 1 ve 0'a yakın doğru tahminler daha az cezalandırılırken, modelin yanlış tahmin ettiği olasılıklar daha fazla cezalandırılır.

---

### Mean Absolute Percentage Error (MAPE) - Ortalama Mutlak Yüzde Hatası

Regresyon problemleri için kullanılan bir performans ölçümüdür.

Hatayı sabit bir şekilde cezalandırır, yani küçük ve büyük hatalar arasında eşit bir ağırlık verir. MSE'nin aksine, büyük hataları abartmaz.

$$MAE = \frac{1}{n} \sum_{t=1}^n |e_t|$$

---

### Mini-batch Gradient Descent

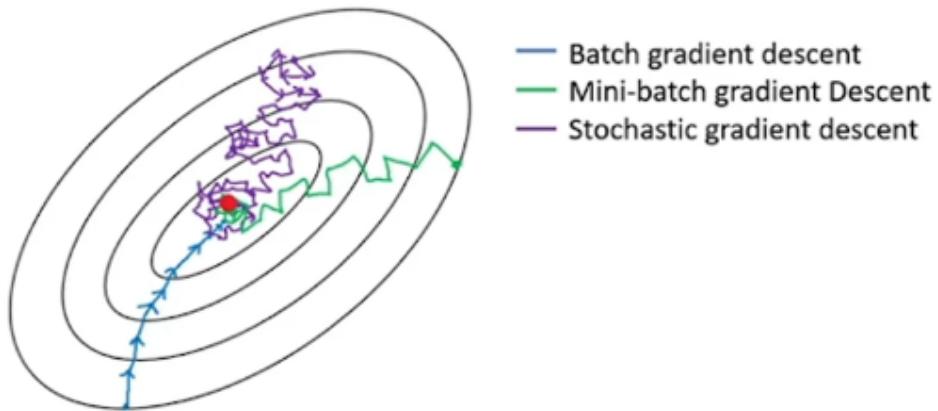
Optimizasyon algoritmasıdır.

Modelin parametrelerini güncellerek maliyet fonksiyonunu minimize etmeyi amaçlar.

- Veri setini mini-batch'lere bölünür 32, 64, 128
- Her mini-batch üzerinde maliyet fonksiyonunun gradyanı hesaplanır. Bu, parametrelerin mevcut durumda nasıl değişmesi gerektiğini gösterir.

$$\theta := \theta - \eta \cdot \nabla_{\theta} J(\theta; x^{(i:i+k)}, y^{(i:i+k)})$$

- Hesaplanan gradyanlar, modelin parametrelerini güncellemek için kullanılır. Bu adım her mini-batch için tekrarlanır.
- Tüm mini-batch'ler işlendiğten sonra bir **epoch** tamamlanmış olur. Epoch, tüm veri setinin model tarafından bir kez işlenmesi anlamına gelir. Model birden fazla epoch boyunca eğitilir.



## Lasso ve Ridge regularizasyonu

Makine öğrenmesi ve istatistiksel modellerde, aşırı öğrenme (overfitting) problemini azaltmak ve daha genel modeller elde etmek için kullanılan iki yaygın **düzenleme (regularizasyon)** yöntemidir. Bu yöntemler, modelin parametrelerini cezalandırarak, modelin aşırı karmaşık hale gelmesini engeller ve daha basit modeller oluşturur. İkisi de regresyon problemlerinde yaygın olarak kullanılır.

### Ridge Regularizasyonu (L2 Regularizasyonu):

Ridge regularizasyonunda, modelin hata fonksiyonuna (maliyet fonksiyonuna) parametrelerin (koefisiyentlerin) karesinin toplamı eklenir. Bu, parametrelerin büyük değerler almasını engeller ve modelin daha dengeli bir şekilde genelleştirme yapmasını sağlar.

$$J(\theta) = \frac{1}{m} \sum_{i=1}^m (y^{(i)} - h_{\theta}(x^{(i)}))^2 + \lambda \sum_{j=1}^n \theta_j^2$$

**Ridge regularizasyonunun temel amacı**, parametrelerin büyüklüğünü sınırlayarak aşırı öğrenmeyi (overfitting) önlemektir. Parametrelerin karelerinin toplamı ne kadar büyükse, maliyet fonksiyonu o kadar artar ve bu parametreler küçültülmeye çalışılır. Ridge regresyonunda, tüm parametreler küçütülür, ancak sıfıra eşitlenmez.

### Lasso Regularizasyonu (L1 Regularizasyonu):

Lasso regularizasyonunda, maliyet fonksiyonuna parametrelerin mutlak değerlerinin toplamı eklenir. Lasso, parametreleri küçültmenin yanı sıra, bazı parametrelerin sıfıra eşitlenmesini sağlar. Bu, parametre seçimi (feature selection) ve daha sade modeller için kullanışlıdır.

$$J(\theta) = \frac{1}{m} \sum_{i=1}^m (y^{(i)} - h_\theta(x^{(i)}))^2 + \lambda \sum_{j=1}^n |\theta_j|$$

Lasso, bazı parametreleri tam olarak sıfıra indirerek, modeldeki gereksiz veya az etkili özelliklerini elimine edebilir. Bu, Lasso'nun **değişken seçimi (feature selection)** özelliğine sahip olmasını sağlar. Özellikle çok fazla özelliğin bulunduğu veri setlerinde Lasso, daha sade ve etkili modeller oluşturabilir.

### **Elastic Net Regularizasyonu:**

Lasso ve Ridge düzenlemelerinin bir kombinasyonu olarak, **Elastic Net** adı verilen bir düzenleme yöntemi de vardır. Elastic Net, hem L1 hem de L2 düzenlemelerini kullanarak, Lasso'nun değişken seçimi yeteneği ile Ridge'in daha dengeli parametre küçültme özelliğini birleştirir. Elastic Net'in maliyet fonksiyonu şu şekildedir:

$$J(\theta) = \frac{1}{m} \sum_{i=1}^m (y^{(i)} - h_\theta(x^{(i)}))^2 + \lambda_1 \sum_{j=1}^n |\theta_j| + \lambda_2 \sum_{j=1}^n \theta_j^2$$

Burada hem  $\lambda_1$  (L1 düzenleme) hem de  $\lambda_2$  (L2 düzenleme) terimleri kullanılarak her iki düzenleme bir arada yapılır.

### **Confusion matrix (karmaşıklık matrisi)**

Sınıflandırma problemlerinde modelin performansını değerlendirmek için kullanılan bir tablodur.

		True Class	
		Positive	Negative
Predicted Class	Positive	TP	FP
	Negative	FN	TN

TP (True Positive): doğru pozitif sayısı

TN (True Negative): doğru negatif sayısı

FP (False Positive): yanlış pozitif sayısı

FN (False Negative): yanlış negatif sayısıdır.

**Accuracy (Doğruluk):** Modelin doğru sınıflandırdığı örneklerin oranı.

$$\frac{TP + TN}{TP + FP + TN + FN}$$

**Precision (Kesinlik):** Pozitif tahminlerin ne kadarının gerçekten pozitif olduğunu gösterir.

$$\frac{TP}{TP + FP}$$

**Recall (Duyarlılık ya da Sensitivite):** Gerçek pozitiflerin ne kadarının doğru bir şekilde pozitif olarak sınıflandırıldığını gösterir.

$$\frac{TP}{TP + FN}$$

**F1-Score:** Precision ve recall'un harmonik ortalamasıdır. Dengeli bir değerlendirme sunar.

$$2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

**Specificity (Özgüllük):** Gerçek negatiflerin ne kadarının doğru bir şekilde negatif olarak sınıflandırıldığını gösterir.

$$\frac{TN}{TN + FP}$$

## ROC eğrisi (Receiver Operating Characteristic Curve)

Bir sınıflandırma modelinin performansını farklı eşik değerlerinde değerlendirmek için kullanılan bir yöntemdir. ROC eğrisi, modelin doğru pozitif oranı (True Positive Rate - TPR) ile yanlış pozitif oranı (False Positive Rate - FPR) arasındaki ilişkiyi gösterir. Bu eğri

sayesinde, bir modelin her türlü karar eşliğinde (threshold) ne kadar iyi performans gösterdiği değerlendirilebilir.

**Y Ekseninde:** True Positive Rate (TPR) bulunur, yani modelin pozitif sınıfı doğru tahmin etme oranı (duyarlılık).

**X Ekseninde:** False Positive Rate (FPR) bulunur, yani negatif sınıfı yanlış tahmin etme oranı.

**True Positive Rate (TPR)** veya **Recall (Duyarlılık)**:

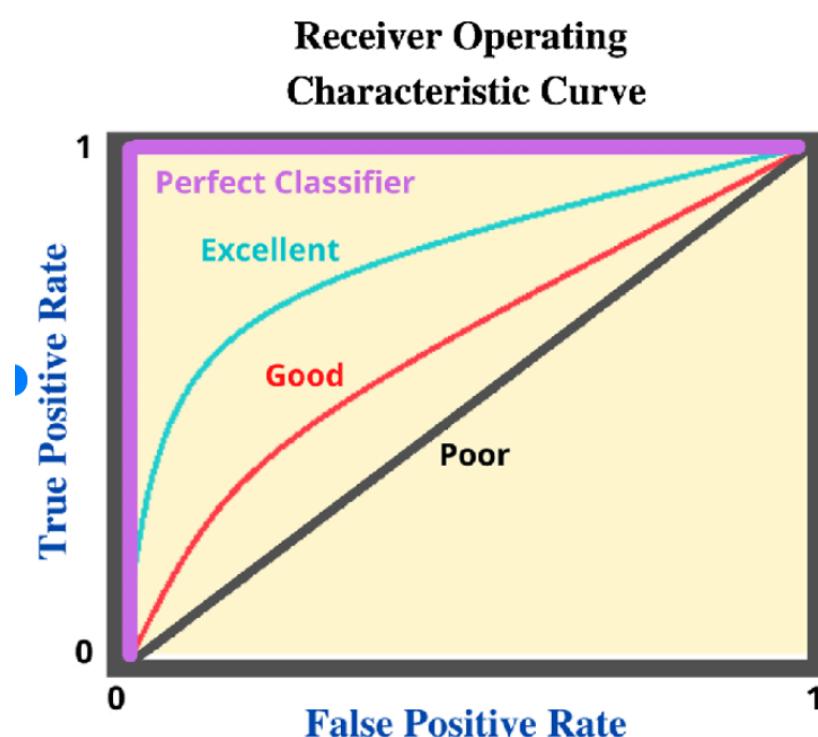
$$TPR = \frac{TP}{TP + FN}$$

Gerçek pozitiflerin ne kadarının doğru tahmin edildiğini gösterir.

**False Positive Rate (FPR)**:

$$FPR = \frac{FP}{FP + TN}$$

Gerçek negatiflerin ne kadarının yanlış pozitif olarak sınıflandırıldığını gösterir. FPR, hatalı bir şekilde pozitif tahmin edilenlerin oranını gösterir.

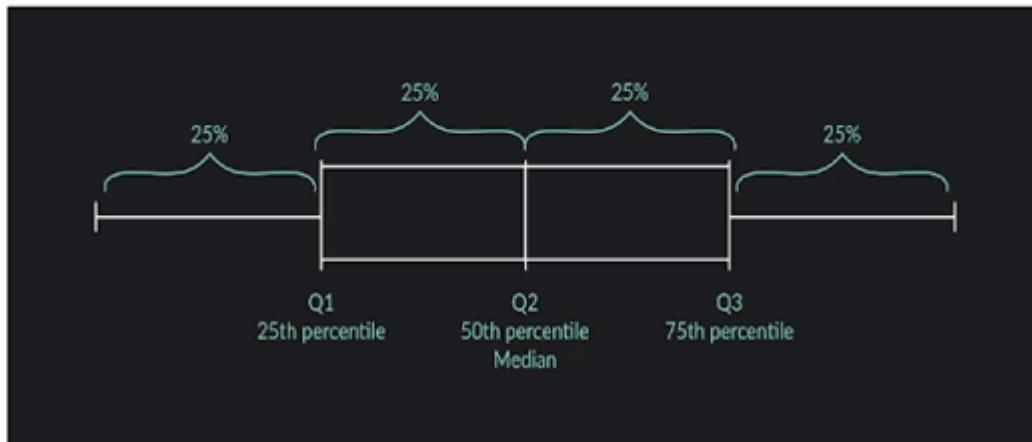


Eğri ne kadar sol üst köşeye yakınsa, modelin performansı o kadar iyidir, çünkü bu, düşük yanlış pozitif oranıyla yüksek doğru pozitif oranına sahip olduğunu gösterir.

**Rastgele sınıflandırma:** Eğer bir model tamamen rastgele sınıflandırma yapıyorrsa, ROC eğrisi **45 derece açılı** bir çizgi olacak şekilde X ve Y eksenin boyunca ilerler. Bu eğrinin altında kalan alan (AUC - Area Under the Curve) %50 olur.

**Mükemmel model:** Mükemmel bir modelde, ROC eğrisi sol üst köşeye dokunur ve sonra yukarı çıkar. Bu modelin AUC skoru %100 olur.

## Aykırı değerlerin (outlier) tespiti



### 1) Boxplot ve Çeyreklikler:

- Boxplot grafiği, veri setinin merkezi eğilimi, yayılımı ve potansiyel aykırı değerleri görselleştirmek için kullanılan bir yöntemdir.
- Grafik, veri setini dört eşit parçaaya böler:
  - Q1 (Birinci Çeyrek - 25th percentile): Verilerin %25'inin altındaki değerler. Boxplot'ta kutunun alt kenarını temsil eder.
  - Q2 (Medyan - 50th percentile): Verilerin %50'sinin altındaki değerler. Bu, kutunun ortasında yatay bir çizgi ile gösterilen medyandır.
  - Q3 (Üçüncü Çeyrek - 75th percentile): Verilerin %75'inin altındaki değerler. Bu, kutunun üst kenarını temsil eder.

### 2) IQR (Interquartile Range - Çeyrekler Arası Aralık):

- IQR: Üçüncü çeyrektan birinci çeyreğin çıkarılması ile elde edilir:

$$IQR = Q3 - Q1$$

- IQR, veri setindeki orta %50'lik dilimi gösterir ve veri yayılımını ölçmek için kullanılır.

### 3) Aykırı Değerlerin Belirlenmesi (Outliers):

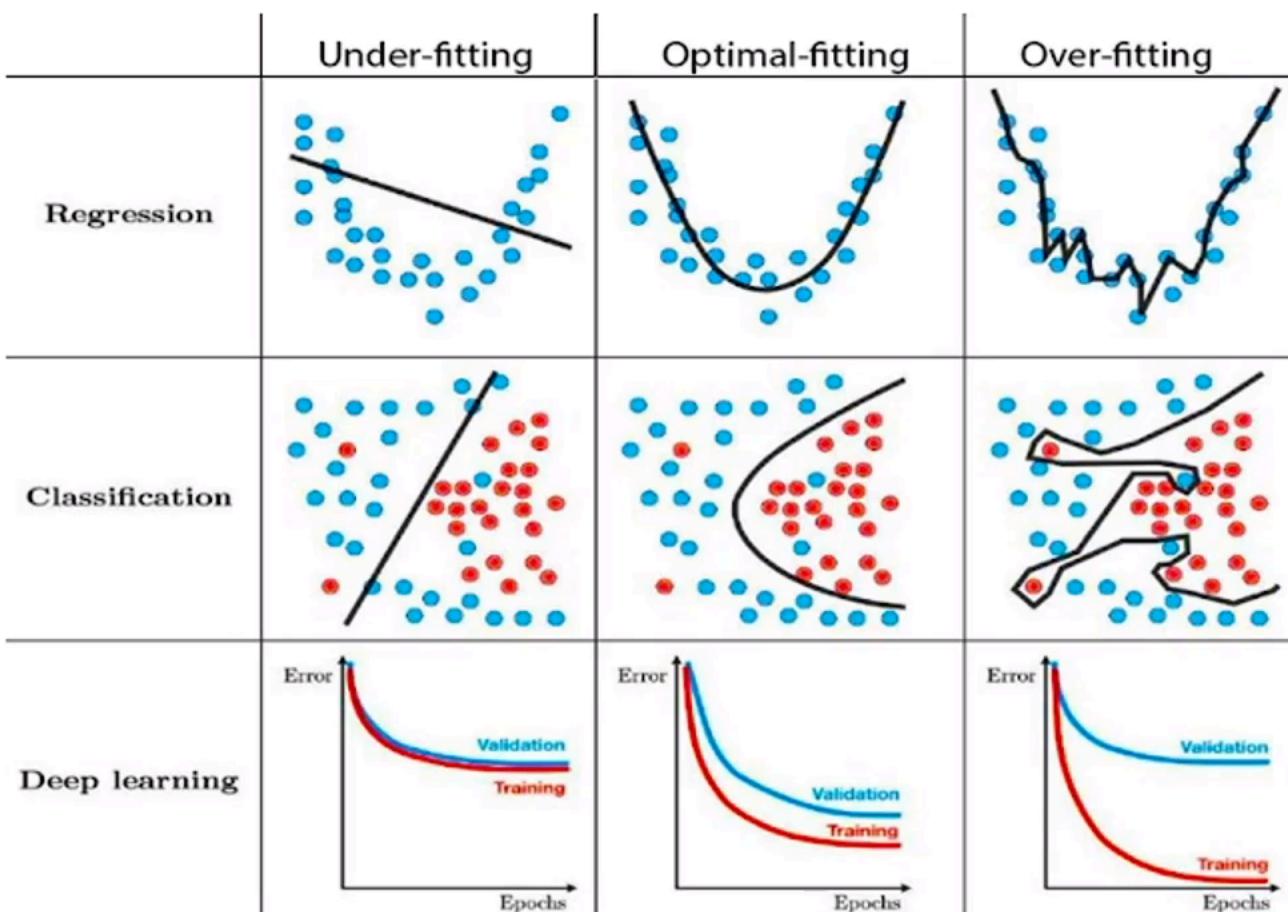
- Aykırı değerleri belirlemek için şu formüller kullanılır:
  - Lower Outlier (Alt Sınır Aykırı Değeri): Lower

$$\text{Lower Outlier} = Q1 - (1.5 \times IQR)$$

- Upper Outlier (Üst Sınır Aykırı Değeri): Upper  

$$\text{Upper Outlier} = Q3 + (1.5 \times IQR)$$
- Eğer bir veri noktası bu sınırların dışında kalıyorsa, aykırı bir değer (outlier) olarak kabul edilir.

## Overfit Underfit Optimal Fit



Makine öğrenmesinde sayısal verilerin tahmin edilmesi için Prediction Algorithms ( Tahmin algoritmaları ) kullanılır. Sayısal olmayan yani kategorik verilerin tahmini için ise Classification ( Sınıflandırma ) kullanılır.

## KNN (K-Nearest Neighbors) Algoritması

KNN (K-Nearest Neighbors) Algoritması iki temel değer üzerinden tahmin yapar;

- Distance (Uzaklık): Tahmin edilecek noktanın diğer noktalara uzaklığı hesaplanır. Bunun için Minkowski uzaklık hesaplama fonksiyonu kullanılır.
- K (komşuluk sayısı): En yakın kaç komşu üzerinden hesaplama yapılacağını söyleziz. K değeri sonucu direkt etkileyecektir. K 1 olursa overfit etme olasılığı çok yüksek olacaktır. Çok büyük olursa da çok genel sonuçlar verecektir. Bu sebeple optimum K değerini tahmin etmek problemin asıl konusu olarak karşımızda durmaktadır.

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”. Bir tahmin yapmak istediğimizde, tüm veri setinde en yakın komşuları arar.

Uzaklık hesaplama işleminde genelde **Öklid fonksiyonu** kullanılır. Öklid fonksiyonuna alternatif olarak **Manhattan**, **Minkowski** ve **Hamming** fonksiyonlarında kullanılabilir. Uzaklık hesaplandıktan sonra sıralanır ve gelen değer uygun olan sınıfa atanır.

KNN (K-Nearest Neighbors) Algoritması ile üretilmiş bir modelin başarısını ölçmek için genel olarak kullanılan 3 adet indikatör vardır.

Jaccard Index: Doğru tahmin kümesi ile gerçek değer kümesinin kesişim kümesinin bunların bireşim kümesine oranıdır. 1 ile 0 arası değer alır. 1 en iyi başarım anlamına gelir.

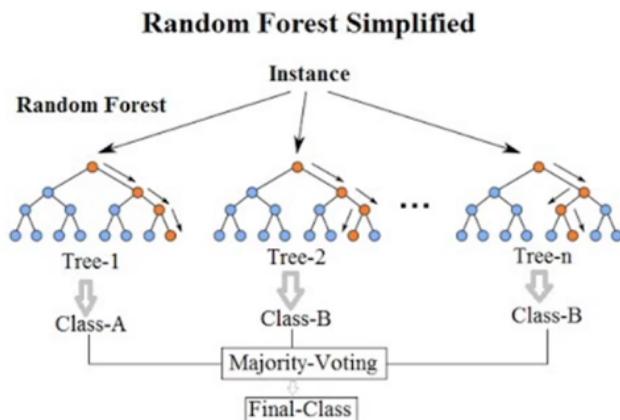
F1-Score: Confusion Matriks üzerinden hesaplanan Precision ve Recall değerlerinden hesaplanır.  $Pre=TP/(TP+FP)$   $Rec=TP/(TP+FN)$   $F1-Score= 2((Pre*Rec)/(Pre+Rec))$  1 ile 0 arası değer alır. 1 en iyi başarım anlamına gelir.

LogLoss: Logistic Regresyon sonunda tahminlerin olasılıkları üzerinden LogLoss değeri hesaplanır. 1 ile 0 arası değer alır. Yukarıdaki iki değerden farklı olarak 0 en iyi başarım anlamına gelir.

KNN gibi noktalar arası mesafe temelli algoritmaların daha başarılı sonuç elde etmek için veriler normalize edilir.

## Decision Tree — Random Forest

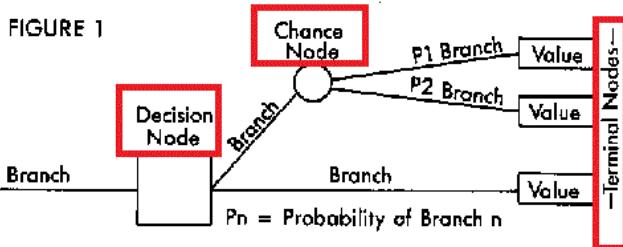
Karar Ağaçları (DT), hem classification ( Sınıflandırma ) hem de [regression](#) ( regresyon ) problemlerinde kullanılan makine öğrenmesinin en popüler algoritmalarından biridir. Amacı veri özelliklerinden basit kurallar çıkarıp bu kuralları öğrenerek bir değişkenin değerini tahmin eden modeli oluşturmaktr. Algoritma eksik değerleri desteklemez makineyi eğitmeden önce [eksik değerleri \( missing value \) çözmemiz](#) gereklidir.



Random Forest algoritması decision tree ( Karar ağacı ) gibi hem classification ( Sınıflandırma ) hem de [regression](#) ( Regresyon ) problemlerinde kullanılabilir. Çalışma mantığı birden fazla karar ağacı oluşturur. Bir sonuç üreteceği zaman bu karar ağaçlarındaki ortalama değer alınır ve sonuç üretilir.

3 çeşit düğüm çeşidi vardır.

1. Chance Node ( Şans Düğümü ): *Daire ile gösterilir. Birden çok olası yol belirtir.*
2. Decision Node ( Karar Düğümü ): *Dikdörtgen ile gösterilir. Bir karar vereleceğini belirtir.*
3. End Node ( Uç Düğümü ): *Üçgen ile gösterilir. Bir sonuç belirtir.*



## Support Vector Machine

Support Vector Machine (SVM), **denetimli öğrenme** algoritmalarından biridir ve **sınıflandırma** ve **regresyon** görevlerinde kullanılır. Ancak genellikle sınıflandırma problemlerinde daha yaygındır. SVM, bir veriyi farklı sınıflara ayırırken iki sınıf arasındaki en iyi ayırcı çizgisi (hiper düzlem) bulmayı amaçlar.

### Temel Çalışma Mantığı:

SVM'in temel amacı, veriyi iki farklı sınıfa ayıran **hiper düzlemi** bulmaktır. Bu hiper düzlem, iki sınıf arasındaki **maksimum marjinal ayımı** sağlar. Başka bir deyişle, sınıfları ayıran çizginin her iki sınıfa da olabildiğince uzak olmasını hedefler.

### Örnek:

2 boyutlu bir veri kümesinde, SVM iki sınıfı ayırmak için bir doğru (1 boyutlu hiper düzlem) bulur. Daha yüksek boyutlu (örneğin 3 boyutlu) bir veri kümesinde ise bu hiper düzlem bir düzlem ya da bir yüzey olabilir.

### SVM'in Önemli Bileşenleri:

- Hiper Düzlem:** İki sınıfı ayıran doğrusal bir sınırdır. Bu sınıflandırma çizgisi, iki sınıfın arasındaki mesafeyi en büyük yapacak şekilde seçilir.
- Destek Vektörleri:** Hiper düzleme en yakın olan veri noktalarıdır. Bu veri noktaları, hiper düzlemi belirlemek için kullanılır. Destek vektörleri, hiper düzlemi etkileyen kritik noktalardır ve sınıflandırma sürecinde önemli bir rol oynarlar.
- Marj:** Hiper düzlem ile destek vektörleri arasındaki mesafedir. SVM, bu marji maksimize etmeye çalışır. İki sınıf arasındaki marj ne kadar büyükse, sınıflandırma o kadar iyi olur.

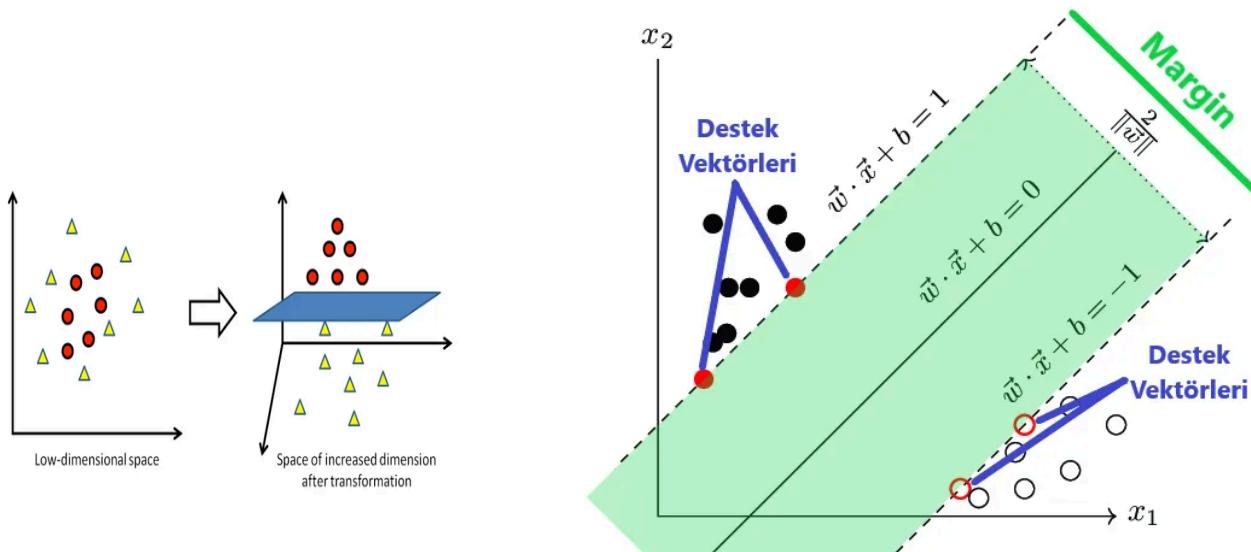
### Çekirdek Fonksiyonları (Kernel Trick):

SVM, veriyi **doğrusal olmayan** ayırmalar için de kullanabilir. Bu durumda veriler doğrusal olarak ayıramayabilir. SVM bu verileri daha yüksek boyutlu bir alana projelendirir ve orada

doğrusal olarak ayırmayı dener. Bu projelendirme işlemi için **çekirdek (kernel) fonksiyonları** kullanılır.

En sık kullanılan çekirdek fonksiyonları şunlardır:

- **Doğrusal (Linear) Kernel:** Veri doğrusal olarak ayrılabilirse kullanılır.
- **Polinomial (Polynomial) Kernel:** Veriler arasında polinomsal bir ilişki varsa kullanılır.
- **RBF (Radial Basis Function) Kernel:** Veriler doğrusal olarak ayrılmıyorsa kullanılır.
- **Sigmoid Kernel:** Genellikle yapay sinir ağlarıyla benzer bir yapı gösterir.



**PCA (Principal Component Analysis)**, Türkçe karşılığıyla **Temel Bileşen Analizi**, veri setindeki yüksek boyutluluğu azaltmak ve veriyi daha sade bir hale getirmek için kullanılan bir **boyut indirgeme** yöntemidir. PCA, verideki değişkenlerin (özelliklerin) çoğunun açıklanmasını sağlayan, daha az sayıda yeni değişken (bileşen) oluşturur. Bu yeni bileşenler, veri setinin **ana özelliklerini** kaybetmeden, veriyi daha düşük boyutlu bir hale getirir.

**Grid Search**, makine öğrenmesinde kullanılan bir **hiperparametre optimizasyonu** teknigidir. Bir modelin performansını artırmak için en iyi hiperparametre kombinasyonunu bulmayı amaçlar. Hiperparametreler, modelin öğrenme sürecini etkileyen, model eğitimi başlamadan önce belirlenen ayarlardır ve modelin doğrudan öğrenme sürecinde optimize edilemezler. Bu yüzden **Grid Search**, belirlenen hiperparametrelerin her bir kombinasyonunu dener ve en iyi sonucu veren kombinasyonu seçer.

**XGBoost (Extreme Gradient Boosting)**, yüksek performanslı bir **makine öğrenmesi algoritmasıdır** ve sınıflandırma ve regresyon problemlerinde yaygın olarak kullanılır.

**Gradient boosting** algoritmasının optimize edilmiş ve daha hızlı versiyonudur. Özellikle büyük ve karmaşık veri setlerinde güçlü sonuçlar verdiği için popülerdir.

XGBoost, her bir yeni modelin önceki modellerin hatalarını düzelterek daha güçlü tahminler yapmasını sağlar. Bu yöntemle zayıf modellerin (genellikle karar ağaçları) bir araya getirilmesiyle güçlü bir topluluk modeli (ensemble model) oluşturulur.

## Naive Bayes Classifier

Naive Bayes sınıflandırıcısının temeli Bayes teoremine dayanır. **lazy** ( tembel ) bir öğrenme algoritmasıdır aynı zamanda dengesiz veri kümelerinde de çalışabilir. Algoritmanın çalışma şekli bir eleman için her durumun olasılığını hesaplar ve olasılık değeri en yüksek olana göre sınıflandırır. Az bir eğitim verisiyle çok başarılı işler çıkartabilir. Test kümelerindeki bir değerin eğitim kümesinde gözlemlenemeyen bir değeri varsa olasılık değeri olarak 0 verir yani tahmin yapamaz. Bu durum genellikle **Zero Frequency ( Sıfır Frekans )** adıyla bilinir. Bu durumu çözmek için düzeltme teknikleri kullanılabilir. En basit düzeltme tekniklerinden biri Laplace tahmini olarak bilinir.

$$P(A|B) = \frac{P(B|A) P(A)}{P(B)}$$

THE PROBABILITY OF "B"  
BEING TRUE GIVEN THAT  
"A" IS TRUE

↓

THE PROBABILITY  
OF "A" BEING  
TRUE

↑  
THE PROBABILITY  
OF "A" BEING  
TRUE  
GIVEN THAT "B" IS  
TRUE

↑  
THE PROBABILITY  
OF "B" BEING  
TRUE

Bayes Formülü

$P(A | B)$  = B olayı gerçekleştiğinde A olayın gerçekleşme olasılığı

$P(A)$  = A olayın gerçekleşme olasılığı

$P(B | A)$  = A olayı gerçekleştiğinde B olayın gerçekleşme olasılığı

$P(B)$  = B olayın gerçekleşme olasılığı

Kullanım alanlarına örnek olarak gerçek zamanlı tahmin, çok sınıflı tahmin, metin sınıflandırması, spam filtreleme, duyarlılık analizi ve öneri sistemleri verilebilir.

# Linear Regression

Linear regression ( Doğrusal regresyon ), temel ve yaygın olarak kullanılan bir tahmin analizidir. **Sayısal girdi ve çıktı değerleri ( Kategorik verilerde kullanılamaz )** arasında ilişki kurmayı sağlar yani kısaca amacımız yukarıdaki grafikte gördüğünüz karmaşık biçimde bulunan gerçek değerlerleri tek bir doğru şeklinde göstermektir.

Bir bağımlı ve bir bağımsız değişken ile regresyon denkleminin en basit şekli,  $y = ax + b$  formülü ile tanımlanır, burada

$y$  = bağımlı değişken,  $a$  = katsayı,  $x$  = bağımsız değişken,  $c$  = sabittir.

Hata miktarı gerçek değerdeki bir nokta ile doğru arasındaki mesafedir.

# Multiple Linear Regression

Multiple linear regression ( Çoklu Doğrusal Regresyon) en yaygın kullanılan linear regression analizidir.

Önceki yazımmda anlattığım [Linear Regression](#) 1 tane bağımlı ve 1 tane de bağımsız değişkenle çalışıyordu. ( $y = ax + b$  ,  $y$  : bağımsız,  $x$  : bağımlı ) Multiple linear regression birden fazla bağımsız değişkenle çalışabilir.

$$y = b_0 + a_1 x_1 + a_2 x_2 + a_3 x_3 + d$$

Örneğin Linear Regression'da aylara göre satış tahmini ( bağımsız değişken) yapmıştık. Multiple linear regression 'da da kilo, yaş ve boy verisinden ayakkabı numarasını tahmin ettirebiliriz. Burada kilo, yaş, boy bağımsız değişkendir yani birden fazladır.

Lineer regresyonda amaç, veriler üzerinde **en iyi doğrusal ilişkiyi** bulmak, yani veri noktalarına en iyi uyan bir doğru çizmektir. Bunun için genellikle **hata fonksiyonu** (error

function) olan **Ortalama Kare Hatası (Mean Squared Error, MSE)** minimize edilir. MSE, her veri noktasının tahmin edilen değerle olan farklarının karesinin ortalamasıdır:

## Dummy Variable ( Kukla Değişken )

OneHotEncoder			
yas	boy	kilo	cinsiyet
20	175	82	e
35	182	65	k
45	168	73	k
32	176	42	e

Dummy variable bir değişkeni ifade eden başka bir değişken olarak tanımlanabilir. Örneğin yukarıdaki cinsiyet kolonu OneHotEncoder ile kategorik veriden sayısal veriye dönüştürülmüştür. Dönüşüm işlemi yapıldıktan sonra elimizdeki kolon sayısı 4'ten 6'ya çıkmıştır. OneHotEncoder sonucu *e* ve *k* kolonlarında dahil olmuştur. Bu veri setini doğrudan makine öğrenme algoritmasına vericek olursak sonucumuzun hatalı çıkma ihtimali yüksektir çünkü bu 6 kolondan 3'ü ( cinsiyet, e, k) özünde aynıdır yani birinin değişmesi diğer kolon değerlerininide etkilemektedir. ( bağımlı ) Bu duruma **Dummy variable trap ( Kukla değişken tuzağı )** denir.

Bu durumdan kurtulmak için 3 kolondan ( cinsiyet, e, k) ikisini çıkarmalıyız ve makine öğrenme algoritmasına veri setini öyle vermeliyiz.

OneHotEncoder

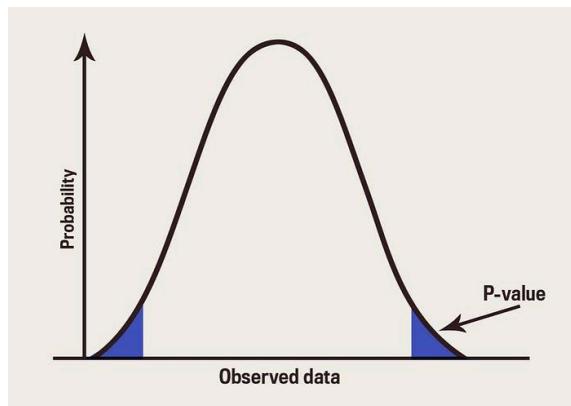
yas	boy	kilo	cinsiyet
20	175	82	e
35	182	65	k
45	168	73	k
32	176	42	e

	e	k
1	1	0
0	0	1
1	0	1
1	1	0

Not : Bazı makine öğrenme algoritmaları dummy variable trap olayına karşı bağılıklıdır.

## P Value ( Olasılık Değeri )



P değeri bir karşılaştırmada istatiksel anlamlılık düzeyine işaret eder. Olası hata miktarını gösterir. Ünlü bir istatistikçi olan [Fisher](#) tarafından bu hatanın maksimum kabul edilebilir düzeyi 0,05 olarak önerilmiş ve kabul görmüştür.

Bir test sonucunda bulunan P değeri 0,05'in altında ise karşılaştırma sonucunda anlamlı farklılık vardır.

## Polynomial Regression

$$y = \alpha + \beta x, \quad y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \epsilon. \quad Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \dots + \beta_h X^h + \epsilon,$$

$$y_i = \alpha + \beta x_i + \epsilon_i.$$

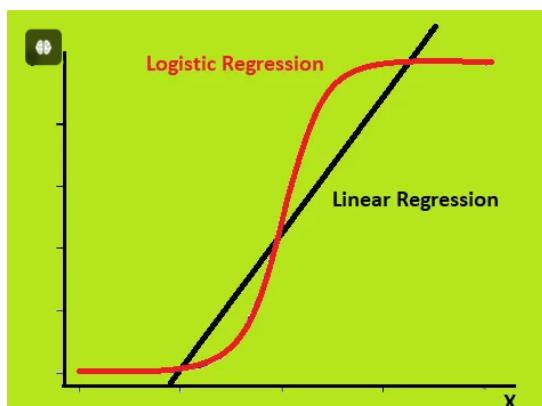
**Linear Regression**

**Multiple Linear Regression**

**Polynomial Regression**

## Logistic Regression

Logistic Regression ( Lojistik Regresyon ) sınıflandırma işlemi yapmaya yarayan bir regresyon yöntemidir. Kategorik veya sayısal verilerin sınıflandırılmasında kullanılır. Bağımlı değişkenin yani sonucun sadece 2 farklı değer alabilmesi durumda çalışır. ( Evet / Hayır, Erkek / Kadın, Şişman / Zayıf vs. )



## Association Rule Mining ( Birliktelik Kural Çıkarıımı ) — APriori Algorithm — Eclat Algorithm

Olayların birlikte gerçekleşme durumlarını çözümleyip verileri arası ilişkiler kuran bir makine öğrenmesi yöntemidir. Birçok makine öğrenme algoritması sayısal verilerle çalışır ve bu algoritmaların çok matematiksel olma eğilimindedir (Logistic Regression, Support Vector Machines ). Ancak, Association Rule Mining algoritmaları kategorik veriler ile çok başarılı bir şekilde çalışabilirler.

Association Rule Mining' e ( Birliktelik Kural Çıkarımı ) örnek olarak genellikle **market sepeti uygulaması** verilir. Bu işlem, müşterilerin yaptıkları alışverişlerdeki ürünler arasındaki birliktelikleri bularak müşterilerin satın alma alışkanlıklarını çözümler.( Örneğin kola alan müşteri ekmekte alır ya da yumurta alan müşteri gofrette alır vs. ) Bu tip birlikteliklerin keşfedilmesi, müşterilerin hangi ürünleri bir arada aldıkları bilgisini ortaya çıkarır ve market yöneticileri de bu bilgi ışığında raf düzenlerini belirleyerek satış oranlarını artırabilir ve etkili satış stratejileri geliştirebilirler.

**Destek ( Support ):** Bir varlığı içeren eylem sayısının toplam eylem sayısına oranıdır. ( $A / \text{Tüm eylem sayısı}$ )

**Güven ( Confidence ):** İki varlığı içeren eylem sayısının birine oranıdır. ( $((A+B) / A)$ )

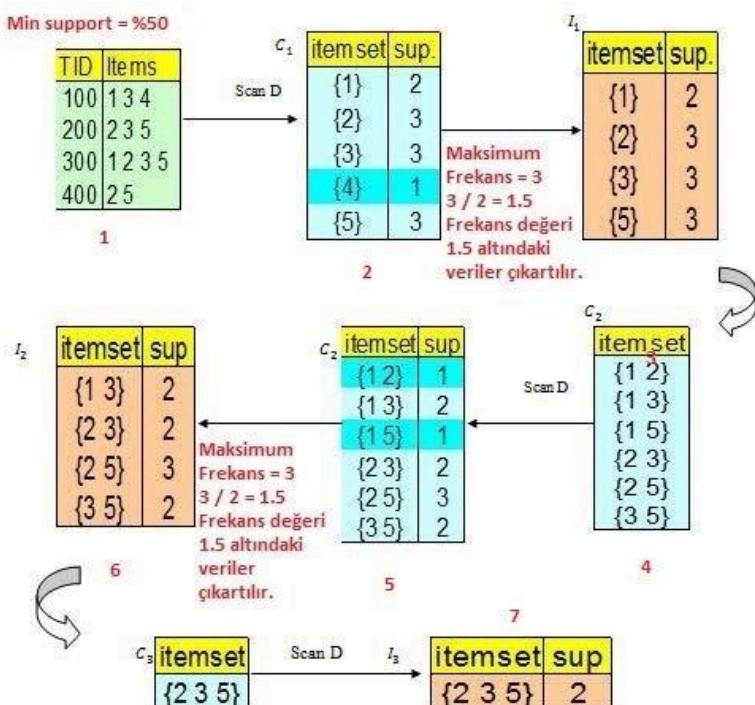
Bir birliktelik kuralı oluşturmak için **destek** ve **güven** değerlerine bakılır. Kural için **minimum destek** ve **minimum güven** şartı aranır.

Birliktelik kuralı için **diger ölçüler** Lift ( Kaldırmak ), Conviction ( Kanaat ), All-Confidence ( Tam Güven ), Collective Strength ( Kolektif güç ) ve Leverage'tır. ( Kaldıraç )

Yayın olarak kullanılan iki tane Association Rule Mining algoritması vardır.

- **APriori Algoritması**
- **Eclat Algoritması**

### **APriori Algoritması**



## Örnek APrriori algoritma çalışması

Makine öğrenmesinde veriler arasındaki ilişkiyi çıkarmak için geliştirilmiş bir algoritmadır. Algoritma aşağıdan yukarıya (bottom-up) yaklaşımı kullanmakta olup, her seferinde tek bir veriyi incelemekte ve bu veriyle diğerleri arasında bir ilişki aramaktadır.

Örneğin yukarıdaki şenin bir marketteki müşterilerin alışveriş sepetleri olduğunu düşünelim. 1. tabloya baktığımızda alınan ürünleri görüyoruz. ( 1 3 4 – 2 3 4 vs. ) Algoritma ilk olarak bu ürünlerin frekansını bulur yani toplam alınma sayısını. ( 1. ürün 2 defa alınmış, 3. ürün 3 defa vs. ) Bu değerleri bulduktan sonra en yüksek frekansının minimum support değerini alır (  $\%50 - 3 * 50 / 100 = 1.5$  ) ve frekansı bu değerden küçük olanlar elenir. Kalan değerlerin kombinasyonu alınarak aynı işlem tekrarlanır ve tablo daha da küçültülür. Bu durum bir ilişki bulununcaya kadar devam eder.

## Eclat Algoritması

Eclat algoritmasının adı Equivalence Class Transformation ( Eşdeğerlik Sınıf Dönüşümü ) kelimelerinin birleşiminden gelmektedir. Apriori algoritmasından en büyük farkı Breadth First Search yerine **Depth First Search** kullanmasıdır. Apriori algoritması ile benzer çalışır. Apriori algoritmasında eleman ( Alışveriş sepetindeki ürün 1,2,3 vs. ) bazlı işlem yapılır ancak Eclat algoritmasında ise elemanların geçtiği transaction ( Alışveriş sepeti 100,200 vs. ) baz alınır.

# Clustering

**Clustering (Kümeleme)**, bir makine öğrenmesi ve veri madenciliği tekniği olup, verileri benzer özelliklerine göre gruptara ayırmayı amaçlar. Bu gruplar, veri noktalarının birbirine olan benzerliği veya uzaklısına dayalı olarak oluşturulur. Kümeleme, denetimsiz öğrenme yöntemlerinden biridir, yani veriler üzerinde herhangi bir önceden tanımlanmış etiket (sınıf) bulunmaz.

## Clustering (Kümeleme) Nasıl Çalışır?

Kümeleme algoritmaları, veri noktalarını benzerlik veya mesafe ölçütlerine göre grupperler. Amaç, aynı gruptaki veri noktalarının birbirine mümkün olduğunda yakın, farklı gruppardaki veri noktalarının ise birbirine mümkün olduğunda uzak olmasını sağlamaktır.

## Kümeleme Algoritmalarının Türleri

### 1. K-Means Clustering:

- En popüler kümeleme algoritmalarından biridir.
- Kullanıcı, küme sayısını ( $K$ ) önceden belirler.
- Algoritma, her veri noktasını en yakın olduğu kümeye atar ve kümeler arasındaki merkezleri (centroid) hesaplar. Merkezler güncellenerek en iyi kümeler elde edilene kadar bu işlem tekrar eder.
- 

### 2. Hierarchical Clustering (Hiyerarşik Kümeleme):

- Veri noktaları arasında hiyerarşik bir ilişki kurar.
- **Agglomerative** (birleştirici) ve **Divisive** (bölgücü) olmak üzere iki türü vardır.
- Agglomerative hiyerarşik kümeleme, her veri noktasını bir küme olarak başlatır ve benzer kümeleri birleştirir. Divisive ise tüm verileri tek bir küme olarak başlatır ve en uzak noktaları ayırarak kümeler oluşturur.

### 3. DBSCAN (Density-Based Spatial Clustering of Applications with Noise):

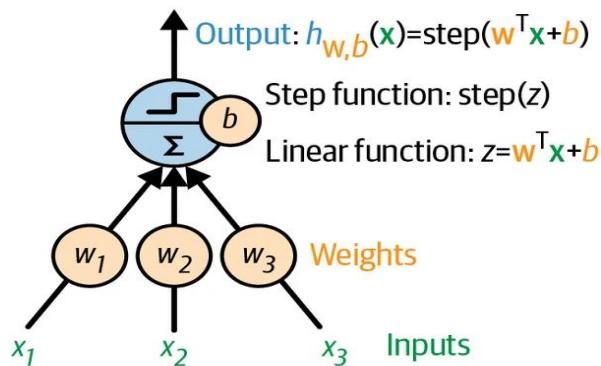
- Yoğunluğa dayalı bir kümeleme yöntemidir.
- Verilerin yoğun olduğu bölgelerdeki noktaları kümelendirir. Yoğun olmayan, seyrek bölgeler gürültü (noise) olarak kabul edilir.
- Küme sayısını önceden belirtmeye gerek yoktur, bu sayede **K-Means**'den farklıdır.
- Gürültü noktalarını da bulabilir.

### 4. Mean Shift:

- Veri noktalarının yoğun olduğu alanları bulmak için kullanılır.
- Merkezleri veri noktalarının yoğun olduğu bölgelerde kaydırarak kümeler oluşturur.
- Küme sayısını önceden belirtmeye gerek yoktur.

# Artificial Neural Networks - ANN

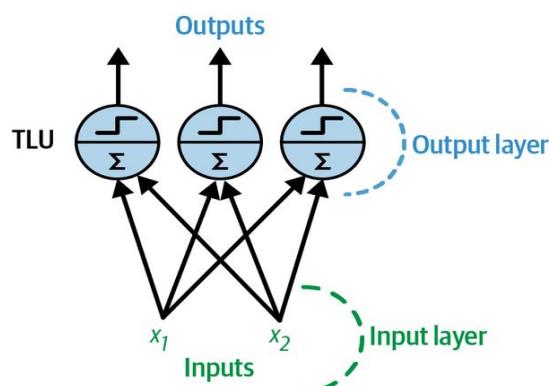
## Perceptron



Yukarıda gördünüz gibi, bu mimaride, girdiler ve çıktı sayılarından oluşur ve her bir girdinin bir ağırlığı vardır. Bu ağırlıklı girdiler öncelikle toplanır daha sonra bir bias eklenir. Bu toplam, bir step fonksiyonundan geçirilir. Bu fonksiyon, örneğin bir işaret fonksiyonu olabilir.

$$\text{sgn } (z) = \begin{cases} -1 & \text{if } z < 0 \\ 0 & \text{if } z = 0 \\ +1 & \text{if } z > 0 \end{cases}$$

İşaret fonksiyonu bildiğiniz gibi sıfırdan küçük değerleri -1'e, sıfır değerlerini sıfıra ve sıfırdan büyük değerleri 1'e götürür. Basit bir perceptron'u, lineer bir ikili sınıflandırma için kullanabilirsiniz. Örneğin önce girdilerin lineer bir fonksiyonu hesaplanır, ardından çıkan sonuç bir threshold yani bir eşik değerini geçerse, sonuç pozitif olur. Belirli bir threshold geçmezse sonuç negatif olur. Bu bahsettiğim örnek, threshold logic unit, kısaca TLU denen, tek bir lineer fonksiyondan oluşuyor. Bir perceptron, bir layerda yani katmanda, birden çok birbiri ile bağlantılı lineer fonksiyondan oluşabilir.



Yukarıdaki görselde iki tane girdi ve üç tane çıktı var. Girdilerin bulunduğu layer'a input layer, çıktıları üreten TLU'ların bulunduğu katmana output layer denir. Dikkat ederseniz girdilerin

hepsi bütün nöronlarla bağlantılı. Bu şekildeki bir katmana *tam bağlantılı layer* ya da *dense layer* denir.

Bu perceptron 'un üç farklı çıktısı var. Hatırlarsanız binary yani ikili sınıflandırma da etiket sayısı ikiydi. Burada üç çıktı olduğu için bu sınıflandırma multilabel denen çok etiketli sınıflandırmadır. Tabi bu mimariyi multiclass denen çok sınıflı sınıflandırma için de kullanabilirsiniz.

Öncelikle her bir girdiye rasgele bir ağırlık verilerek girdilerin toplamı bulunur ve daha sonra bu toplama bir bias eklenir. Unutmayın, bir nöron diğer bir nöronu sık sık tetiklediğinde, aralarındaki bağlantı daha çok güçlenir. Nöronlardan geçen girdiler bir çıktı üretir. Bu çıktı bir tahmindir. Tahminin ne kadar hata yaptığına bakılır. Daha az hata ile tahmin yapmak için ağırlıklar güncellenir.

Perceptron, XOR gibi bazı basit problemleri çözemedi.

Perceptron 'un eksikliklerini ortadan kaldırmak için çok katmanlı multilayer perceptron geliştirildi.

## Çok Katmanlı Perceptron

Multilayer perceptronlar bir girdi katman, bir gizli katman ve birde çıktı katmandan oluşur. Bu basit bir çok katmanlı perceptronudur.

Bu yapay sinir ağı mimarisinde, bir gizli katman var. Eğer birden fazla gizli katman olursa buna *derin sinir ağı* denir.

Tamam girdiler sinir ağlarından geçiyor ve bir tahmin yapılıyor. İşte tam burada backpropagation teknigi ortaya çıktı. Bu teknik ile hata geriye doğru dağıtılarak ağırlıklar güncellenir. Ve en iyi tahmin elde edilinceye kadar bu döngü devam ediyor.

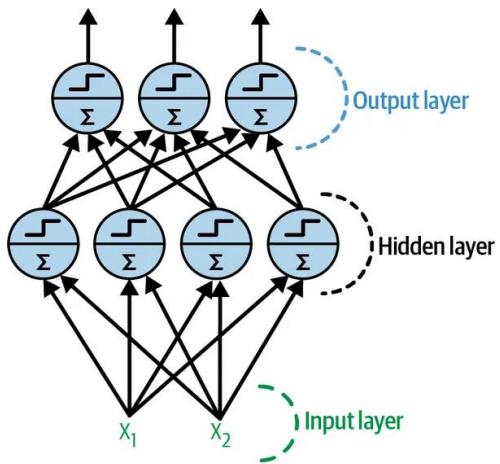
Önce *batch* denilen bir grup veri alınır. Bu batch sayısı örneğin 32 olabilir. Bu veriler eğitim esnasında bir çok kez bütün sinir ağıdan geçer. Bu her bir geçişe *epoch* deniyor.

Bir batch veri, girdi katmanından sinir ağına girer. Bu girdi katmanından geçen veriler, gizli katmandan geçerken çıktı hesaplanır. Bu çıktılar diğer katmana ilettilir. Bu şekilde girdiler bütün katmanlardan geçerek bir sonuç ortaya çıkar. Bu geçişe forward pass yani ileriye doğru geçiş denir. Sinir ağıının çıktısı bir tahmindir. Her bir katmanlardan geçen sonuçlar saklanır.

Bunun için loss fonksiyonu kullanılır. Loss fonksiyonu, gerçek değer ile tahmin değerini karşılaştırır ve bir tahmin hatasını bulur. Her bir batch sinir ağından geçer ve bir tahmin hesaplanır. Ve bu tahminin hatası ölçülür. Bu hata geriye doğru girdi katmanına kadar dağıtılr ve ağırlıklar güncellenir. Güncellenen ağırlıklara göre diğer batch tahmin edilir ve yine tahmin hatası hesaplanır. Bu hataya göre ağırlıklar güncellenir. Bu şekilde bütün veriler sinir ağından geçer. Unutmayın başlangıçtaki ağırlıklar rasgele verilir. Diğer türlü yapay sinir ağıının eğitimi başarısız olur.

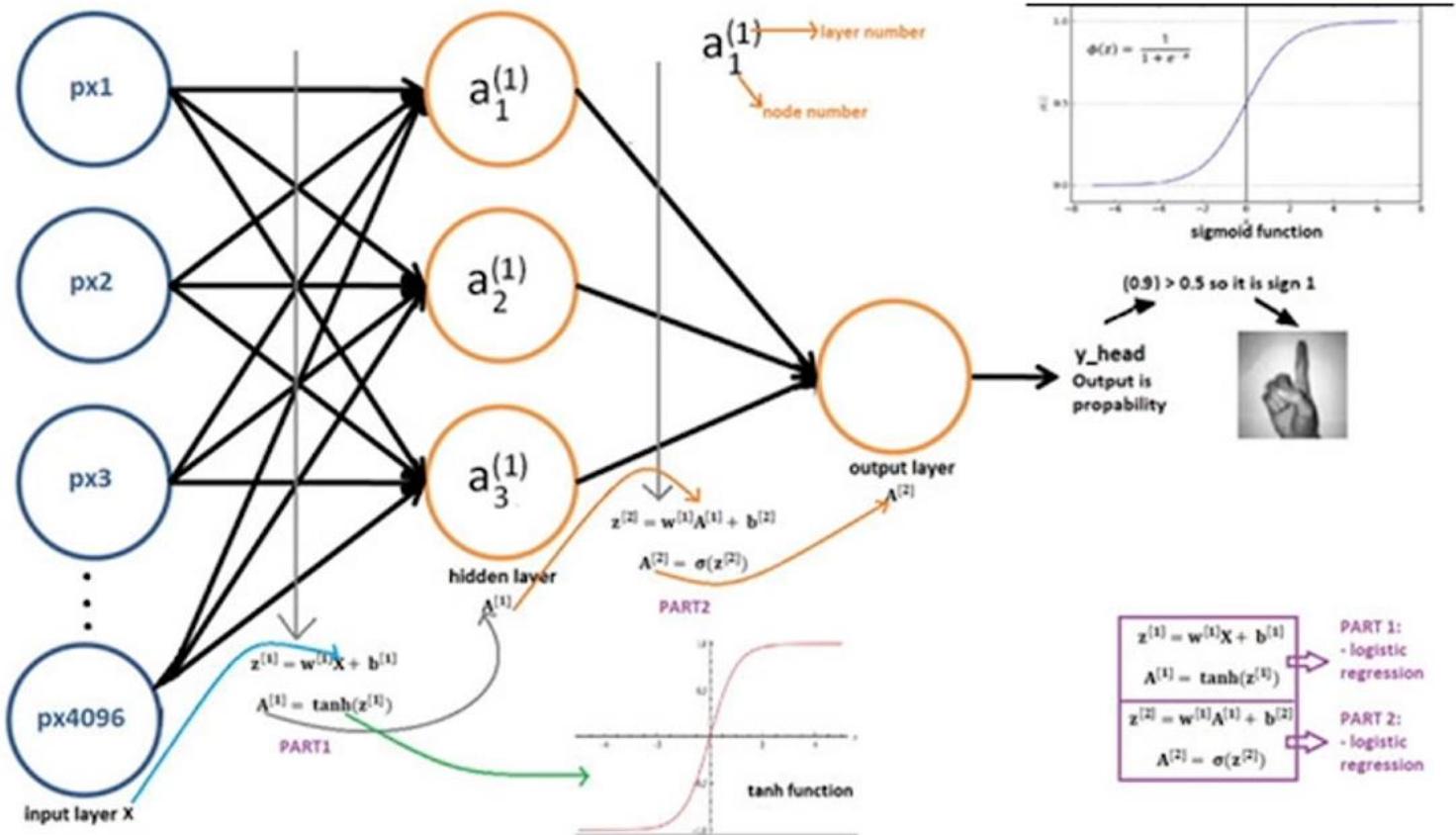
Perceptronlarda kullanılan aktivasyon, step fonksiyonuydu. Multilayer perceptronlarda kullanılan aktivasyon fonksiyonu ise sigmoid, hiperbolik tanjant ya da ReLU olabilir. En çok kullanılan fonksiyon ReLU'dur.

Aktivasyon fonksiyonunu kullanmamızın sebebi, mimarimize nonlineerlik kazandırmak istememizdir. Böylece mimarımız daha esnek olur. Eğer aktivasyon fonksiyonu kullanmazsak kompleks problemleri çözemeyiz.



## ANN

Lojistik regresyona dayalı olarak inşa edilmiştir.



### Hidden Layer (Gizli Katman):

- Girdiler, gizli katmandaki nöronlara bağlıdır. Bu nöronlar üzerinde işlemler gerçekleştirilir.
- Gizli katmandaki her nöron, girdi değerlerinin ağırlıklı toplamı ve bias kullanılarak hesaplanır:

$$z^{[1]} = W^{[1]}X + b^{[1]}$$

- Aktivasyon fonksiyonu olarak burada **tanh fonksiyonu** kullanılmıştır:

$$A^{[1]} = \tanh(z^{[1]})$$

Bu fonksiyonun grafiği de görselde sol alt köşede yer almaktadır. Tanh fonksiyonu, çıktı değerlerini -1 ile 1 arasında sınırlar.

### Output Layer (Çıktı Katmanı):

- Gizli katmandaki aktivasyon değerleri çıktı katmanına gönderilir.
- Çıktı katmanında, her bir nöron yine girdi değerlerinin ağırlıklı toplamını alır ve bir aktivasyon fonksiyonuna tabi tutulur:  $z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$
- Çıktı katmanında kullanılan aktivasyon fonksiyonu **sigmoid fonksiyonudur**:  
 $A^{[2]} = \sigma(z^{[2]})$ , Burada  $\sigma$  sigmoid fonksiyonudur ve sonucu 0 ile 1 arasında sınırlar. Grafiği sağ üst köşede yer alır.
- **Sigmoid Fonksiyonu**:

$$\sigma(x) = \frac{1}{1+e^{-x}}$$
 Bu fonksiyon, çıktıların olasılık olarak yorumlanması sağlar.

# AKTİVASYON FONKSİYONLARI

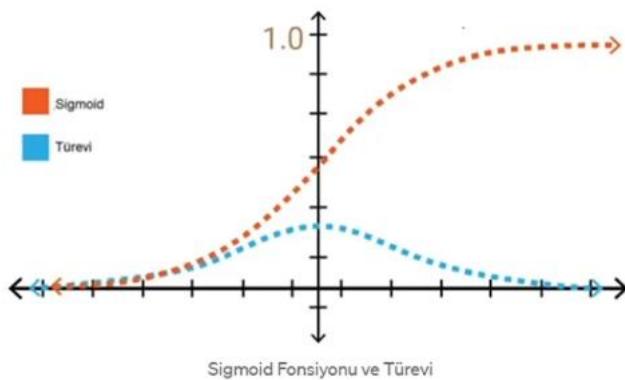
Aktivasyon fonksiyonlarının amacı weight(ağırlık) ve bias değerlerini ayarlamaktır. Mimarimize nonlineerlik kazandırmak istememizdir.

## 1. Sigmoid Fonksiyonu

Aldığı değerleri 0 ve 1 arasına hapseder. Yüksek bir değer geldiğinde 1'e yakın olurken düşük

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

bir değer geldiğinde 0'a yakın olur.

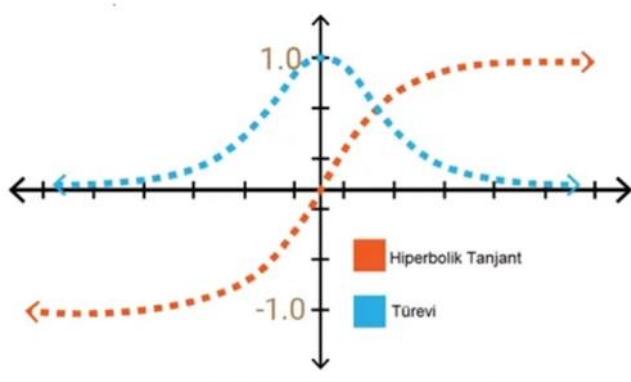


Sigmoid Fonksiyonu ve Türevi

## 2. tanh (Hiperbolik Tanjant) Fonksiyonu

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

Tanh fonksiyonu, girişleri -1 ile 1 arasında sınırlar.



Hiperbolik Tanjant Fonksiyonu ve Türevi

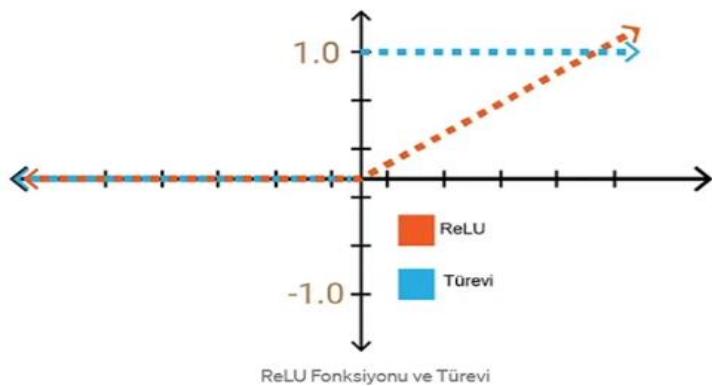
## 3. ReLu Fonksiyonu

Çalışma mantığı olarak gelen değerlerin pozitif mi negatif mi olduğuna bakar sonrasında eğer gelen değer negatif bir değerse işlem sonucunu 0 verir. Ancak gelen değer pozitifse herhangi bir sıkıştırma ya da değiştirmeye işlemi uygulamaz olduğu gibi geçer.

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

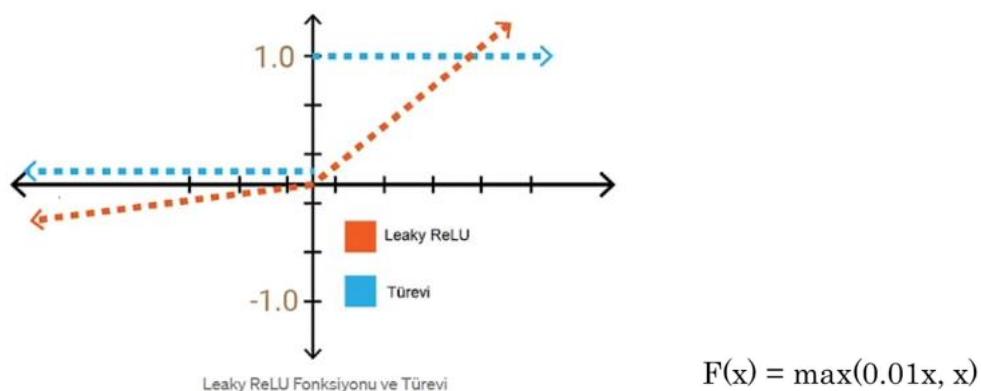
Bilgisayar bu denklemi 6 kat daha hızlı hesaplar.

bu fonksiyonda 0 odaklı olmadığından bazı nöronlar (%40'a kadar) ölebiliyor.



#### 4. Leaky ReLU Fonksiyonu

Ölü nöron sonunu ortadan kaldırmak için geliştirilmiş bir fonksiyondur diyebiliriz. Negatif bir değer geldiğinde çok küçük bir sayı döndürür ve bu sayede nöronların ölmesinin önüne geçer. Hesaplama sayısının artmasına rağmen diğer fonksiyonlara göre oldukça hızlı çalışır.



## 5. Softmax Fonksiyonu

Softmax, birden fazla sınıfın olduğu durumlarda her bir sınıfa ait olasılığı hesaplar. Çıktılar,

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}}$$

toplamı 1 olan olasılık dağılımı şeklinde olur.

class sayısı  $\leq 2$  ise sigmoid class sayısı  $> 2$  ise softmax

## 6. Swish Fonksiyonu

Swish, sigmoid ve ReLU fonksiyonlarının bir karışımı gibidir. Negatif değerler için sıfıra yakın

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

sonuçlar verirken, pozitif değerler için lineer gibi davranışır.

Fonksiyon	Çıktı Aralığı	Avantajları	Dezavantajları	Kullanım Alanı
Sigmoid	0 ile 1	Olasılık temelli çıktı, ikili sınıflandırmada kullanılır.	Gradyan kaybolması	Çıktı katmanları (binary classification)
Tanh	-1 ile 1	Negatif ve pozitif değerleri ifade edebilir.	Gradyan kaybolması	Gizli katmanlar
ReLU	0 ile sonsuz	Hesaplama basit, daha hızlı öğrenme	Negatif bölgelerde ölü nöronlar	Gizli katmanlar
Leaky ReLU	$-\infty$ ile sonsuz	Ölü nöron problemine çözüm	Yüksek negatif eğim riskleri	Gizli katmanlar
Softmax	0 ile 1 (her sınıf için)	Çok sınıflı sınıflandırmada kullanılır.	Sigmoid'in çoklu sınıflı hali	Çıktı katmanları (multiclass classification)
Swish	$-\infty$ ile sonsuz	Yüksek performanslı, derin öğrenmede etkili	Hesaplama maliyeti daha yüksek	Derin sinir ağları

AKTİVASYON FONKSİYONU	DENKLEM	ARALIK
Doğrusal Fonksiyon	$f(x) = x$	$(-\infty, \infty)$
Basamak Fonksiyonu	$f(x) = \begin{cases} 0 & \text{için } x < 0 \\ 1 & \text{için } x \geq 0 \end{cases}$	$\{0, 1\}$
Sigmoid Fonksiyon	$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}}$	$(0, 1)$
Hiperbolik Tanjant Fonksiyonu	$f(x) = \tanh(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})}$	$(-1, 1)$
ReLU	$f(x) = \begin{cases} 0 & \text{için } x < 0 \\ x & \text{için } x \geq 0 \end{cases}$	$[0, \infty)$
Leaky (Sızıntı) ReLU	$f(x) = \begin{cases} 0.01 & \text{için } x < 0 \\ x & \text{için } x \geq 0 \end{cases}$	$(-\infty, \infty)$
Swish Fonksiyonu	$f(x) = 2x\sigma(\beta x) = \begin{cases} \beta = 0 & \text{için } f(x) = x \\ \beta \rightarrow \infty & \text{için } f(x) = 2\max(0, x) \end{cases}$	$(-\infty, \infty)$

## Callbacks (geri çağrıclar)

### Early stopping

Düzenleme (regularization) tekniğidir.

Early stopping, modelin doğruluğunu veya kaybını (loss) **doğrulama veri seti (validation set)** üzerinde izler.

- Doğrulama setindeki performans iyileşmeyi durdurduğunda veya kötüleşmeye başladığında, eğitim süreci durdurulur.



### Model Checkpoint

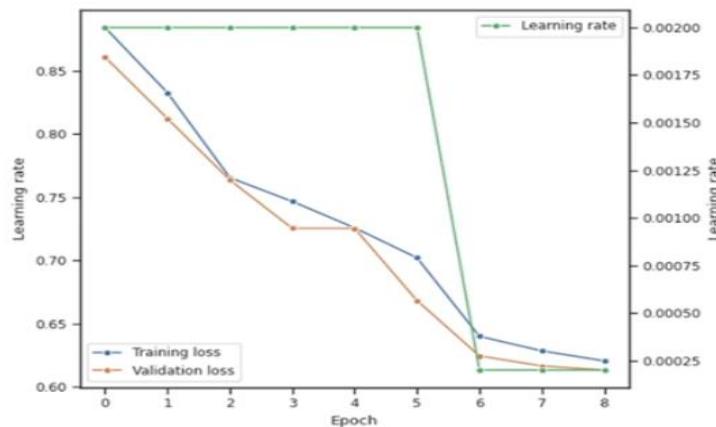
(Model Kontrol Noktası), makine öğrenmesi modellerinin eğitim sürecinde belirli aralıklarla veya belirli koşullar sağlandığında modelin **durumu (ağırlıkları ve parametreleri)** kaydedilen bir yöntemdir.

## ReduceLROnPlateau

Derin öğrenme modellerini eğitirken kullanılan bir **öğrenme oranı (learning rate)** düzenlemeye stratejisidir. Bu teknik, modelin doğrulama performansı bir süre boyunca iyileşmezse, öğrenme oranını otomatik olarak azaltarak modelin daha küçük adımlarla öğrenmesini sağlar. Özellikle doğrulama kaybı (validation loss) veya doğrulama doğruluğu (validation accuracy) gibi metrikler "plato" durumuna geldiğinde, yani bir süre boyunca iyileşme göstermezse devreye girer.

Öğrenme oranını, **doğrulama performansında bir plato (iyileşmenin durduğu nokta)** gözlemlendiğinde değiştirir.

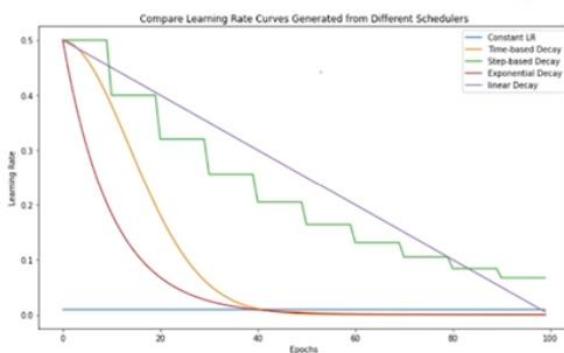
Bir özelliğe göre değişir bir özelliğe bakılır.



## Learning Rate Scheduler

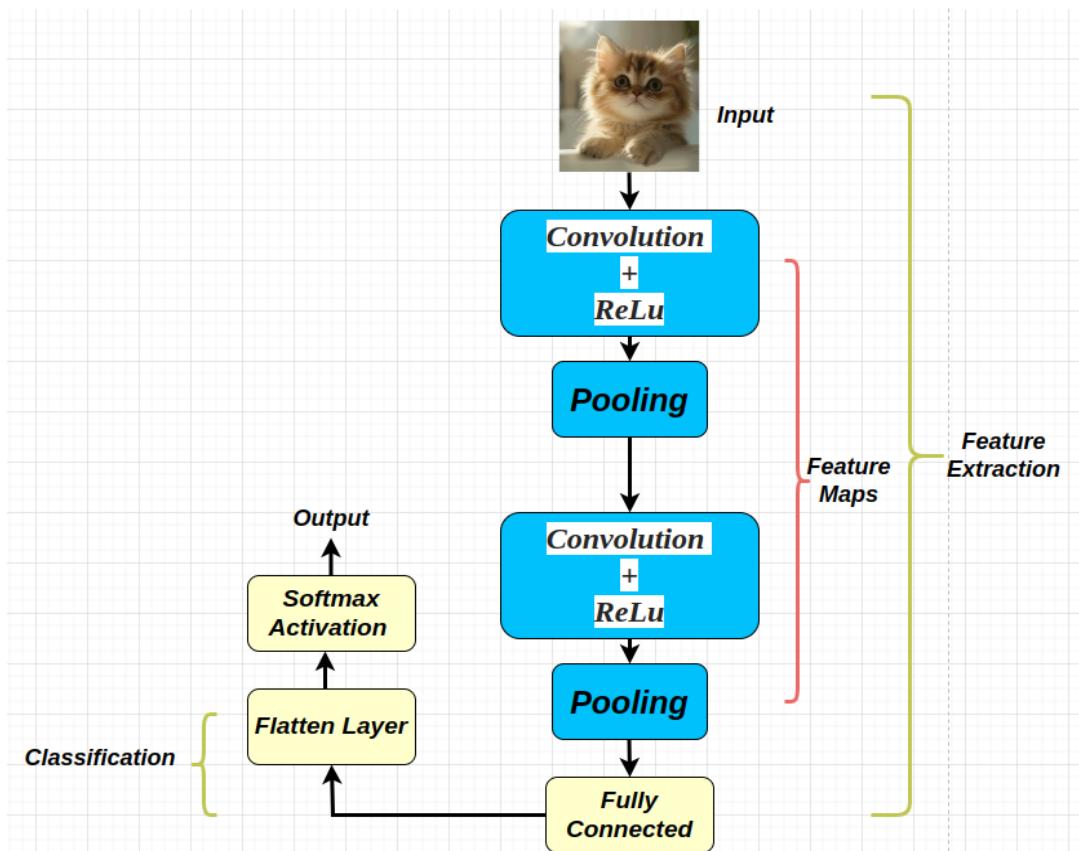
(Öğrenme Oranı Zamanlayıcısı), makine öğrenimi modellerinin eğitim sürecinde **öğrenme oranını (learning rate)**, eğitim ilerledikçe belirli bir stratejiye göre dinamik olarak değiştiren bir tekniktir.

Öğrenme oranını **eğitim süresi boyunca sabit bir plana göre** değiştirir. Burda ise bir çok değişken etkilidir.



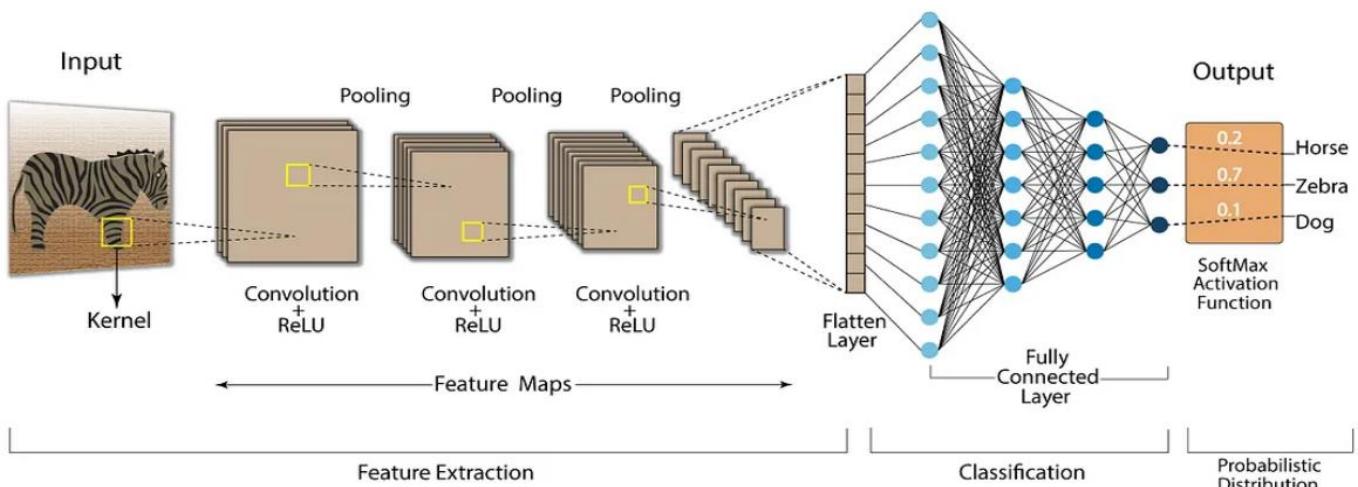
Özellik	Learning Rate Schedulers	ReduceLROnPlateau
<b>Çalışma Prensibi</b>	Epoch veya step bazlı sabit bir plana göre değişir	Performans takibine dayalı, doğrulama setindeki plateau'da tetiklenir
<b>Tetikleyici</b>	Belirli epoch/step sayısı	Doğrulama performansında iyileşme olmadığından
<b>Kullanım Senaryoları</b>	Sabit oranlı bir planla öğrenme oranını kademeli olarak azaltmak için kullanılır	Performans plato yaparsa öğrenme oranını dinamik olarak azaltmak için kullanılır
<b>Kontrol Mekanizması</b>	Eğitimde belirli bir epoch veya step'ten sonra devreye girer	Performans izleme ve duraksama durumunda devreye girer
<b>Düzenleme Tipi</b>	Sabit bir plan veya matematiksel formül	Dinamik, performansa bağlı

**CSVLogger**, makine öğrenmesi modellerinin eğitim sürecinde kaydedilen eğitim metriklerini (örneğin, kayıp değerleri, doğruluk oranları) **CSV dosyasına** yazdırın bir callback fonksiyonudur. Bu sayede modelin eğitim performansı, epoch bazında kayıt altına alınarak daha sonra analiz edilebilir veya görselleştirilebilir.



# CNN - Convolutional Neural Networks

CNN'ler, verinin uzamsal ilişkilerini ve paternlerini öğrenmek için kullanılan özel katmanlarla çalışır. Bu yapı, geleneksel yapay sinir ağlarından farklıdır ve görüntülerin piksel düzeyinde işlenmesini sağlar.



CNN yapısı iki ana kısımdan ve bu kısımlara bağlı bazı katmanlardan oluşuyor.

- **Feature Extraction**
- **Classification**

Feature extraction yani nitelik çıkarımı kısmına bakacak olursak ilk katman olarak **Convolution** katmanını görürüz. Bu katman ağa verilen giriş değerleri üzerinde bazı filtreleme işlemlerinin uygalandığı yerdir.

Giriş değerleri sonrasında **ReLU** aktivasyon fonksiyonunun kullanıldığı ReLU katmanından geçerek **Pooling** katmanına verilir

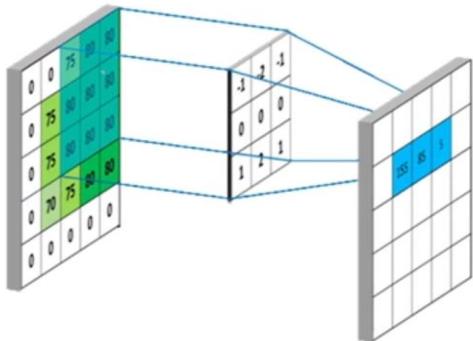
Pooling katmanı Convolution katmanından gelen değerler için bir boyut indirgeme işlemi uygulayarak 2D olan matrisleri lineer bir vektöre çevirir.

Classification için hazırlanan giriş değerlerimiz bir sinir ağına verilir. Bu ağa **Fully-Connected** diyoruz. Kendisi aslında bildiğimiz [Yapay Sinir Ağları](#)'ndaki hidden layer fakat fully connected (Kısaca girişteki bir nöron öndeğindeki her bir nöronla bağlantılı). Bu katmanın sonunda da sınıflandırma işlemini gerçekleştiren bir **Softmax** regresyonu bulunur.

# Convolution Katmanı

Bu katman, giriş verisi üzerinde kayan bir **filtre (kernel)** uygular.

Filtre, giriş verisi (örneğin bir görüntü) üzerinde belirli bir boyutta (genellikle  $3 \times 3$ ,  $5 \times 5$ , vb.) kaydırılır ve her adımda girişteki pikseller ilefiltredeki ağırlıkların çarpımı alınarak bir **özellik haritası (feature map)** üretilir.



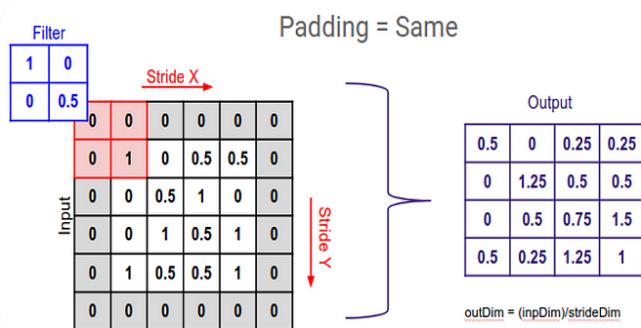
## ReLU - Activation Function

Özellik haritalarına doğrusal olmayanlık ekleyerek daha karmaşık desenlerin öğrenilmesini sağlar.

CNN içerisinde de Convolution çıktısındaki tüm hücrelere uygulanıyor.

Belirli bir evrişim işlemi 0 veya negatif bir değerle sonuçlanırsa bu aranan niteliğin orada bulunmadığı manasına geliyor ve o kısım 0 ile ifade ediliyor. Diğer tüm durumlardaki değerler ise korunuyor.

## Matris çarpımında ufak bir optimizasyon

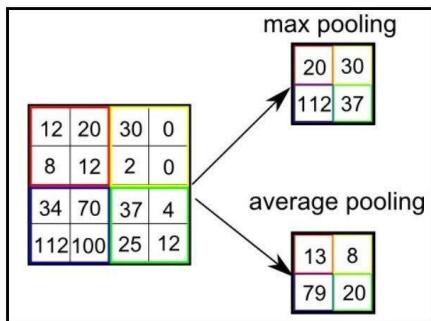


Resmin köşelerindeki pikseller ,filtre matrisinin iterasyonu esnasında ortadaki piksellerden daha az kaplanıyor. Bu da sinir ağında ortadaki piksellerin kenarlara nazaran daha fazla ağırlıklandırılmasına sebep oluyor. Bu da resmin kenarları için yapılacak tespitlerde doğruluk oranının kötü olacağı anlamına gelir. Bunun için matrisimize Padding uyguluyoruz. Padding ile hedef matrisimizin kenar tarafları 0 veya 1 olarak kaplanıyor ve bu da filtre matrisinin orjinal resmin kenarlarında daha fazla alan kaplayabilmesini sağlıyor. Kıyılarda gezen filtrenin sayesinde resim hakkında daha fazla bilgi gidiyor sinir ağına ve daha fazla bilgi eşittir daha yüksek accuracy :)

**Striding :** Convolution esnasında filtre matrisini 1 satır 1 sütun kaydırırmaktansa 2 veya 3 satır 3 sütun şeklinde gezdirelim demişler. Bu da yüksek boyutlu resimlerde hesaplama maliyetini düşürmüştken herhangi bir veri kaybına da sebep olmamış.

## Pooling Katmanı

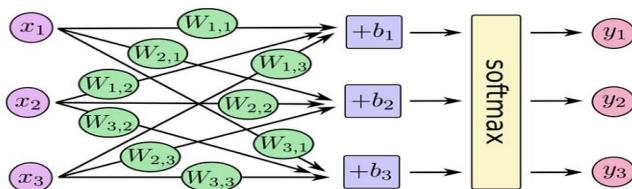
Havuzlama yani Pooling katmanındaki amaç, çıktı boyutunu azaltıp özelligin yalnızca varlığına ilişkin olan kısımlarını almaktır. Bu parametre sayısını düşürür ve hesaplama sayısını azaltır.



Pooling işlemi için matris belli ayrılır. Ayrılan parçalardaki hücre değerlerinin ortalaması veya maksimum olanı alınarak yeni ve daha küçük bir matris oluşturulur. CNN mimarisinde genellikle ardışık evrişim blokları arasına birer Pooling katmanı eklenir.

## Fully Connected Katmanı

CNN'in sonunda yer alan ve tüm özellik haritalarını bir vektöre çevirerek, sınıflandırma görevini gerçekleştiren katmandır. Sınıflandırma işleminin gerçekleştiği bu katmanda sınıflandırma algoritması için genellikle Softmax regression kullanılır. Softmax regression ise genelleştirilmiş Logistic regression olarak bilinir.



## Dropout Layer (Bırakma Katmanı)

Aşırı öğrenmeyi (overfitting) önlemek için bazı nöronların geçici olarak devre dışı bırakılmasını sağlar.

Örneğin, **Dropout(0.5)** ifadesi, o katmandaki nöronların %50'sinin her eğitim adımında devre dışı bırakılacağı anlamına gelir.

**Normalization Layer (Normalizasyon Katmanları)** Eğitim sırasında modelin daha stabil hale gelmesini sağlamak için aktivasyonları normalize eder.

Çeşitleri:

### Batch Normalization:

Her minibatch'te yer alan aktivasyonları normalize eder. Böylece modelin öğrenme süreci hızlanır ve gradyanlar daha stabil hale gelir.

Batch normalization, her batch'teki aktivasyonların ortalamasını 0, standart sapmasını 1 olacak şekilde ayarlar.

### Layer Normalization:

Batch yerine her bir katman içerisindeki aktivasyonları normalize eder. Özellikle NLP gibi sıralı verilerde yaygın olarak kullanılır.

## Öğrenme Aktarımı/Transfer Learning

bir problemi çözerken elde ettiği bilgiyi saklayıp, başka bir problem ile karşılaşlığında o bilgiyi kullanmasıdır. Öğrenme aktarımı ile önceki bilgiler kullanılarak daha az eğitim verisi ile daha yüksek başarı gösteren ve daha hızlı öğrenen modeller elde edilir.

- Eğitim Süresini Azaltmak:
- Az Veriye Sahip Olma Durumu:
- Daha İyi Performans

### 1. Tümevarımsal Öğrenme Aktarımı (Inductive Transfer Learning):

- **Kaynak Görev ve Hedef Görev Farklı:** Hedef görev, kaynak görevden farklıdır, ancak öğrenme aktarımı yapılır.
- **İki Duruma Ayrılır:**
  - **Etiketli Veri Durumu:** Kaynak görevde bol miktarda etiketli veri varsa, bu yöntem **çok görevli öğrenmeye** (multi-task learning) benzer. Ancak buradaki amaç, sadece hedef görevde başarı elde etmektir. Örneğin, kedi-köpek sınıflandırma modeli, araç sınıflandırma görevi için başlangıç noktası olarak kullanılır.
  - **Etiketsiz Veri Durumu:** Kaynak veri etiketsiz olduğunda, bu yöntem **kendi kendine öğrenme** (self-taught learning) gibi çalışır. Etiketsiz verilerden öğrenilen özellikler, hedef görev için kullanılır. Örneğin, bilinmeyen resimlerden öğrenilen bilgilerin, araç sınıflandırımda kullanılması.

### 2. Dönüştürücü Öğrenme Aktarımı (Transductive Transfer Learning):

- **Görev Aynı, Veri Farklı:** Kaynak ve hedef görev aynıdır, ancak veriler farklıdır.
- **Hedef Veri Etiketsiz:** Hedef görevde etiketli veri bulunmaz, ancak kaynak görevde bol miktarda etiketli veri mevcuttur. Örneğin, bir mobil operatörün eski bir kampanya verilerini kullanarak yeni bir kampanya analizi yapması.
- **Özellik Uzayı Durumları:**
  - **Özellik Uzayı Aynı:** Eğer kaynak ve hedef veri kümelerinin özellik uzayı aynıysa (örneğin, kullanıcı bilgileri aynıysa), model aynı özellikler üzerinde öğrenme yapar.

- **Özellik Uzayı Farklı:** Kaynak ve hedef veri kümelerinin özellik uzayı farklıysa (örneğin, yeni bilgiler eklenirse), öğrenme aktarımı farklı özellik uzaylarına uyarlanarak yapılır.

### 3. Gözetmensiz Öğrenme Aktarımı (Unsupervised Transfer Learning):

- **Etiketsiz Görevler:** Tümevarımsal öğrenme aktarımına benzer, ancak etiketsiz görevler üzerinde çalışır.
- **Kaynak ve Hedef Veri Benzer:** Kaynak ve hedef veri kümeleri benzer olabilir, ancak görevler farklıdır.
- **Kullanım Alanları:** Bu yöntem, **kümeleme**, **boyut küçültme** gibi etiketsiz öğrenme problemlerinde kullanılır.

transfer learning, genellikle iki temel şekilde uygulanır:

#### 1. Feature Extraction (Özellik Çıkarma)

Bu yöntemde, önceden eğitilmiş bir modelin ilk katmanları (özellikle convolutional katmanlar) kullanılır ve modelin ağırlıkları sabitlenir. Son katmanlar (fully connected katmanlar) çıkarılır veya değiştirilir ve yalnızca bu son katmanlar yeniden eğitilir.

#### 2. Fine-Tuning (İnce Ayar)

Bu yöntemde, modelin tüm katmanları veya belirli bir kısmı serbest bırakılır ve model yeni veriyle yeniden eğitilir. Ancak modelin önceki görevde öğrendiği bilgileri koruyarak yeni görevde uyum sağlama hedeflenir.

İşte öğrenme aktarımında neyin aktarılacağına dair dört temel yaklaşımın özetleri:

#### 1. Örnek Aktarımı (Instance Transfer):

- **Tanım:** Kaynak verideki bazı örneklerin, hedef veri setinde kullanılmasıdır.
- **Nasıl Çalışır:** Kaynak veri doğrudan kullanılamaz, ancak kaynak verideki örneklerin uygun ağırlıklarla hedef veride kullanılması başarayı artırabilir.
- **Kullanım Durumu:** Kaynak ve hedef veri setlerinin ortak özelliklere sahip olduğu durumlarda geçerlidir. Örneğin, kitap önerisi için kullanılan bir modelin örnekleri, film önerisi yapmak üzere kullanılabilir.

#### 2. Özellik Temsili Aktarımı (Feature Representation Transfer):

- **Tanım:** Kaynak verideki özellik temsillerinin hedef veri setinde kullanılmasıdır.
- **Nasıl Çalışır:** Ortak özelliklerin yanı sıra, sadece kaynakta ya da hedefte bulunan özellikler de kullanılabilir. Bu, daha iyi özellik gösterimleri sağlayarak hedef görevdeki performansı artırır.
- **Kullanım Durumu:** Örneğin, yüz tanıma modelinde göz ile ilgili özellikler, yorgunluk tespitinde kullanılabilir.

#### 3. Parametre Aktarımı (Parameter Transfer):

- **Tanım:** Benzer görevlerde ortak parametre ve hiper-parametrelerin transfer edilmesidir.
- **Nasıl Çalışır:** Kaynak görevdeki öğrenilmiş parametreler ve hiper-parametreler, hedef görev'e aktarılır.
- **Kullanım Durumu:** Benzer veya yakın görevlerde kullanılır. Örneğin, iki benzer sınıflandırma görevi için aynı parametreler kullanılabilir.

#### 4. İlişki Kurma Tecrübesinin Aktarımı (Relational Knowledge Transfer):

- **Tanım:** Kaynak ve hedef veri arasındaki ilişkilerin benzer olması durumunda, bu ilişkilerin transfer edilmesidir.
- **Nasıl Çalışır:** Veri kaynakları arasındaki ilişkiler eşleştirilir ve ilişkisel bilginin aktarımı yapılır.
- **Kullanım Durumu:** Kaynak ve hedef veri arasında ilişkilerin olduğu varsayıldığında kullanılır.

### AlexNet (2012)

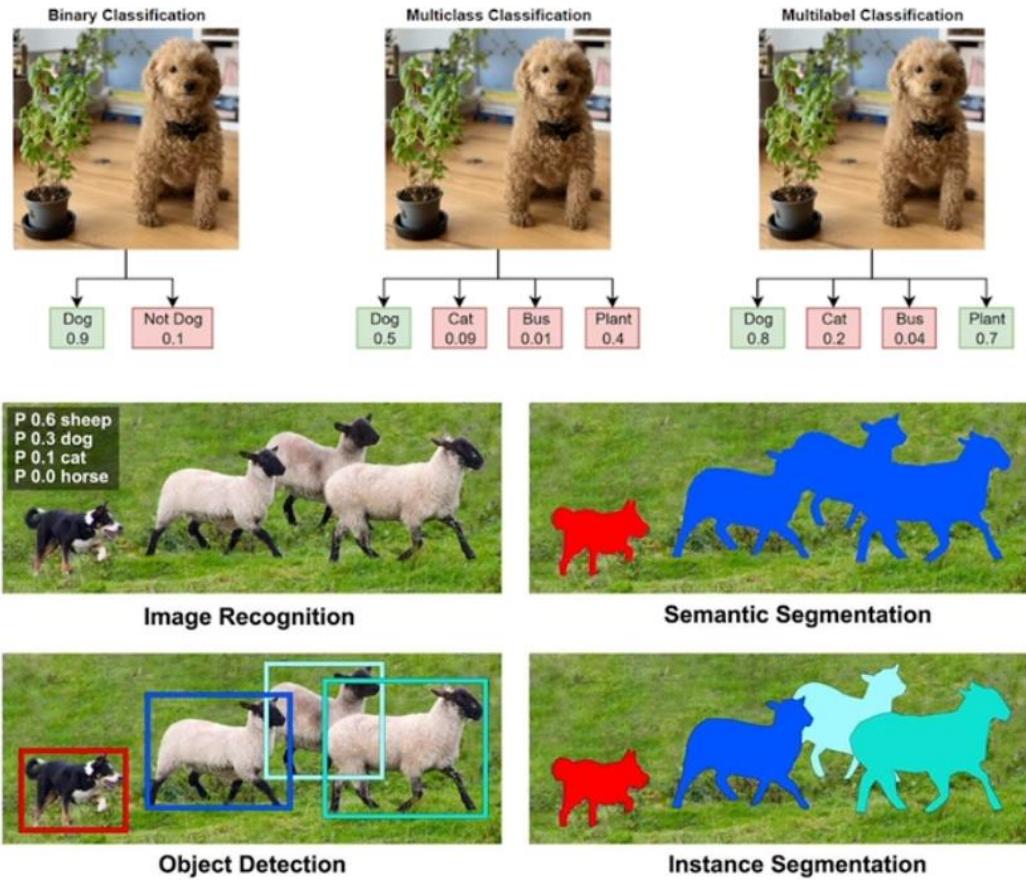
- Derin ve büyük bir CNN mimarisidir. ImageNet gibi büyük veri kümeleri üzerinde başarı gösteren ilk CNN'dir.
- ReLU aktivasyon fonksiyonunu tanıtarak, eğitim hızını büyük ölçüde artırmıştır.
- Dropout kullanımı ile aşırı öğrenmeyi (overfitting) azaltmıştır.
- GPU kullanımı ile eğitim süreci hızlandırılmıştır.

### VGGNet (2014)

- 3x3 evrişim filtreleri ile daha küçük ancak derin katmanlar kullanır.
- Derinliği artırarak daha iyi özellik çıkarımı sağlamıştır.
- Kernel size

### GoogLeNet (Inception v1, 2014)

- Inception modülü ile farklı boyutlarda filtrelerin aynı katmanda kullanılmasına olanak tanıyan yenilikçi bir mimari.
- Bu mimari, derin ve geniş yapılar yerine daha verimli ve optimize edilmiş bir ağ sunar.
- Parametre sayısı daha düşük ve daha hızlı bir yapı sağlar.



### Temel Farklar:

Görev	Amaç	Çıktı	Bilgi Düzeyi
<b>Classification</b>	Görüntünün hangi sınıfı ait olduğunu belirlemek	Tek bir sınıf etiketi	Tüm görüntüye genel sınıflandırma
<b>Object Detection</b>	Görüntüdeki nesnelerin yerlerini ve sınıflarını tespit etmek	Sınıf etiketi ve bounding box'lar	Nesnelerin yerlerini belirler, şekillerini vermez
<b>Segmentation</b>	Nesnelerin piksellerinin sınıfını belirlemek	Her piksel için sınıf etiketi	Nesnelerin tam sınırlarını belirler

## MATRİS İŞLEMLERİ

### 1. Matris Çarpımı

İki matrisin çarpımı, ilk matrisin satırları ile ikinci matrisin sütunlarının skaler çarpımlarından oluşur.

$$\begin{aligned} & \begin{bmatrix} 2 & 5 \\ 7 & 3 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 \\ 4 & -1 \end{bmatrix} \\ &= \begin{bmatrix} 2 \cdot 1 + 5 \cdot 4 & 2 \cdot 0 + 5 \cdot (-1) \\ 7 \cdot 1 + 3 \cdot 4 & 7 \cdot 0 + 3 \cdot (-1) \end{bmatrix} \\ &= \begin{bmatrix} 22 & -5 \\ 19 & -3 \end{bmatrix} \end{aligned}$$

### 2. Transpoz (Transpose)

Bir matrisin satırları ile sütunlarının yer değiştirmesi işlemine **transpoz** denir. Matrisin transpozu, özellikle sinir ağlarında ağırlık güncellemeleri sırasında veya optimizasyon işlemlerinde kullanılır.

$$\begin{aligned} & (A + B)^T = A^T + B^T \\ & (A - B)^T = A^T - B^T \\ & (k \cdot A)^T = k \cdot A^T \\ & (A \cdot B)^T = B^T \cdot A^T \\ & (A \cdot B \cdot C)^T = C^T \cdot B^T \cdot A^T \\ & (A^T)^T = A \quad (A^{-1})^T = (A^T)^{-1} \end{aligned}$$

### 3. Ters Matris

Bir matrisin tersi, o matris ile çarpıldığında birim matrisi elde etmenizi sağlayan matristir. Ters matris, genellikle doğrusal sistemlerin çözümünde ve çeşitli optimizasyon problemlerinde kullanılır. Bir matrisin tersi sadece kare matrisler için hesaplanabilir ve determinantının sıfır olmaması gereklidir.

### 4. Determinant

Determinant, bir kare matrisin büyüklüğünü veya özelliklerini ifade eden bir sayıdır. Özellikle bir matrisin tersinin olup olmadığını belirlemek için kullanılır. Determinant sıfırsa, matrisin tersi yoktur.

$A = [3]$  olmak üzere,

$$\det(A) = 3$$

$$\det(A \cdot B) = \det(A) \cdot \det(B)$$

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \text{ olmak üzere,}$$

$$\det(A) = 1 \cdot 4 - 3 \cdot 2 = -2$$

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \text{ olmak üzere,}$$

$$\det(A) = 1 \cdot 5 \cdot 9 + 2 \cdot 6 \cdot 7 + 3 \cdot 4 \cdot 8 - (7 \cdot 5 \cdot 3 + 8 \cdot 6 \cdot 1 + 9 \cdot 4 \cdot 2)$$

$$= 45 + 84 + 96 - 105 - 48 - 72$$

$$= 0$$

$\det(A) \neq 0$  olmak üzere,

$$A = \begin{bmatrix} 3 & 0 & 0 \\ 5 & -2 & 0 \\ 2 & -4 & 6 \end{bmatrix}$$

$$\det(A^{-1}) = \frac{1}{\det(A)}$$

$$\det(A) = 3 \cdot (-2) \cdot 6 = -36$$

**5. Eigenvektör ve Eigenvalue (Özvektör ve Özdeğer)** Bir matrisin özvektörü, matrisin lineer dönüşümü altında yönü değişmeyen bir vektördür. Bu özvektöre karşılık gelen skaler değer ise özdeğer (eigenvalue) olarak adlandırılır. Özdeğerler, özellikle PCA (Principal Component Analysis) ve spektral kümeleme gibi algoritmalarında kullanılır. **karakteristik polinom**

$$A = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} \quad \det(A - \lambda I) = 0 \quad \det \begin{pmatrix} 2 - \lambda & 1 \\ 1 & 2 - \lambda \end{pmatrix} = 0$$

$$(2 - \lambda)(2 - \lambda) - 1 = 0 \Rightarrow \lambda^2 - 4\lambda + 3 = 0$$

Özdeğer  $\lambda_1 = 3$  için:

$$(A - 3I) = \begin{pmatrix} 2 - 3 & 1 \\ 1 & 2 - 3 \end{pmatrix} = \begin{pmatrix} -1 & 1 \\ 1 & -1 \end{pmatrix}$$

Bu matrisle ilgili denklem şudur:

$$\begin{pmatrix} -1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

Bu sistemi çözerek  $v_1 = v_2$  olduğunu buluruz, yani eigenvektör  $v = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$  olacaktır.

## 6. Olasılık Dağılımları

Olasılık dağılımları, rastgele bir değişkenin nasıl dağıldığını gösteren fonksiyonlardır. En yaygın olanları **Bernoulli**, **Binomial**, **Poisson**, **Normal (Gaussian)** ve **Uniform** dağılımlardır.

## 7. Bayes Teoremi

Bayes teoremi, olasılık teorisinde, bir olayın koşullu olasılığını hesaplamada kullanılan bir formüldür. Özellikle makine öğreniminde, Naive Bayes gibi algoritmalar bu teorem yaygın olarak kullanılır.

## 8. Koşullu Olasılık

Bir olayın, başka bir olayın gerçekleşmesi durumundaki olasılığıdır. Olasılık hesaplamaları sırasında önemli bir yere sahiptir.

## 10. Kovaryans ve Korelasyon

Kovaryans, iki değişkenin nasıl birlikte değiştiğini ölçer. Korelasyon ise bu ilişkinin ne kadar güçlü olduğunu gösterir. Korelasyon, özellikle değişkenler arası ilişkiyi anlamak için önemlidir.

- **Sorulabilecek Sorular:**

- Kovaryans ve korelasyon arasındaki fark nedir?
- Korelasyon katsayısı nasıl yorumlanır?

## 11. Markov Zincirleri

Bir sistemin bir duruma geçiş olasılığının, yalnızca mevcut duruma bağlı olduğu bir süreçtir. Reinforcement learning ve stokastik modellemelerde kullanılır.

- **Sorulabilecek Sorular:**

- Markov zincirlerinin temel özellikleri nelerdir?
- Markov zincirleri hangi problemleri çözmek için kullanılır?

## 12. Monte Carlo Simülasyonu

Monte Carlo simülasyonu, rastgele örnekler üreterek olasılık hesaplamaları ve optimizasyon problemleri çözmek için kullanılır. Özellikle, karmaşık sistemlerde olasılık dağılımlarını simüle etmek için etkilidir.

## 13. Olasılık Yoğunluk Fonksiyonu (Probability Density Function - PDF)

**DF (Probability Density Function)**, yani **Olasılık Yoğunluk Fonksiyonu**, sürekli rastgele değişkenlerin olasılık dağılımlarını tanımlamak için kullanılan bir fonksiyondur. Bir PDF, bir sürekli rastgele değişkenin belirli bir aralıkta bulunma olasılığını verir. Genellikle normal dağılımlarda kullanılır.

PDF'in hesaplanması, hangi olasılık dağılımını kullandığınıza bağlıdır. Genellikle en sık kullanılan PDF'lerden biri **normal dağılımdır**. Diğer olasılık dağılımları için de PDF'ler farklı sekillerde hesaplanır. Aşağıda en yaygın kullanılan normal dağılım için PDF örneği verilmiştir:

### 1. Normal Dağılım (Gaussian Distribution) PDF

Normal dağılımı niteleyen iki parametre vardır. Bunlar ortalama ve standart sapmadır.

- Normal dağılım, ortalama  $\mu$  ve standart sapma  $\sigma$  ile tanımlanır. Normal dağılıminin PDF'i şu şekilde hesaplanır:

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

$f(x)$  = olasılık yoğunluk fonksiyonu

$\sigma$  = standart sapma

$\mu$  = ortalama

## Makine Öğrenmesinde Veri Normalizasyonu ve Standardizasyonu

Bazı gicik makine öğrenmesi algoritmaları özellikle mesafe/uzaklık (distance) üzerine kurgulanmış olanları doğası gereği girdi değişkenlerinin birbirleri ile aynı aralıkta ve mümkünse küçük sayılarından oluşmasını tercih ediyorlar.

**Normalizasyon teknikleri**, farklı ölçeklere sahip verileri ortak bir ölçüye getirmek için kullanılan yöntemlerdir. Veriler farklı ölçüm birimlerine veya genişliklere sahip olabilir ve bu durum, makine öğrenmesi algoritmalarının performansını olumsuz etkileyebilir. Normalizasyon, verilerin ölçeklerini belirli bir aralığa veya dağılıma indirerek modelin daha tutarlı ve doğru çalışmasını sağlar.

- KNN: En Yakın Kapı Komşusu
- SVM: Destekli Atan Vektör Makinemsileri
- K-Means: Kayda Değer Ortalamalar

Normalizasyon, verileri 0 ve 1 arasında yeniden ölçekler. Bu işlem tüm parametrelerin aynı pozitif ölçüye sahip olması gereken bazı durumlarda yararlı olabilir. Ancak aykırı değerlerin (outliers) kaybolmasına yol açar.

En sık kullanılan normalizasyon yöntemlerinden biri olan Feature Scaling, (Özellik Ölçekleme veya Min-Max Normalizasyonu) normal dağılımı esas alır ve her bir değerin ortalamadan olan uzaklığının standart sapmaya oranı ile bulunur. Bu metot ile değişkenlerin farklı ortalamalarda ve standart sapmaya sahip olmasına izin verilir ancak aynı aralıkta olmaları şartıyla.

Standardizasyon ise veriyi aynı ortalama (0) ve aynı standart sapmaya (1) sahip olması için yeniden ölçeklendirilmesidir.

Çok değişkenli analizlerde değişkenlerin standartlaştırılması doğru analiz için önemli rol oynamaktadır. Farklı ölçeklerde ölçülen değişkenler analize eşit katkıda bulunamaz. Örneğin 0–100 aralığında ve 0–1 aralığında yer alan iki feature üzerinde standardizasyon gerçekleştirilmemezse 0–100 aralığındaki değişkenin modeldeki ağırlığı daha fazla olacaktır. Verileri karşılaştırılabilir ölçeklere dönüştürmek bu sorunu önleyebilir. Veri standardizasyonu ile bu ölçeklendirmeyi sağlayabiliyoruz.

En sık kullanılan standardizasyon türlerinden olan Z Score Scaling, veri setindeki tüm değişkenlerin ortalamasının sıfıra standart sapmasının ise bire eşitlendiği bir standardizasyon yöntemidir.

Hiperparametreleri optimum şekilde ayarlanmış modellerde dahi veri dağılıminin gerektirdiği standardizasyon veya normalizasyon yöntemi varsa model doğruluğunu bu preprocessing işlemleri ile artıtabiliriz.

Support Vector Machine, K-NN, lojistik regresyon gibi algoritmalar ile birlikte PCA (Temel Bileşen Analizi) işleminde de önemli rol oynayan standardizasyonun etkisini aşağıdaki grafiklerde ve kod çıktısında prediction accuracy değerini %81.48'den %98.15'e çıkardığını görebiliyoruz.

### 1. Min-Max Normalizasyonu

- **Tanım:** Bu teknik, verileri belirli bir aralık (genellikle 0 ile 1 arasında) içine ölçekler. Bu, verilerin minimum değerini 0 ve maksimum değerini 1 olacak şekilde dönüştürür.

$$X_{\text{norm}} = \frac{X - X_{\text{min}}}{X_{\text{max}} - X_{\text{min}}}$$

## 2. Z-Score Standardizasyonu (Standart Normalizasyon)

Bu teknik, verileri ortalaması 0 ve standart sapması 1 olacak şekilde dönüştürür. Z-score normalizasyonu, veri noktalarının ortalamadan kaç standart sapma uzaklıkta olduğunu gösterir.

$$X_{\text{norm}} = \frac{X - \mu}{\sigma}$$

Verilerin bir normal dağılıma sahip olduğunu düşündüğünüzde ya da verilerin geniş bir aralıkta dağıldığı durumlarda kullanılır. Lineer regresyon, lojistik regresyon, SVM ve k-ortalama gibi algoritmaların performansını artırabilir.

## Boxplot

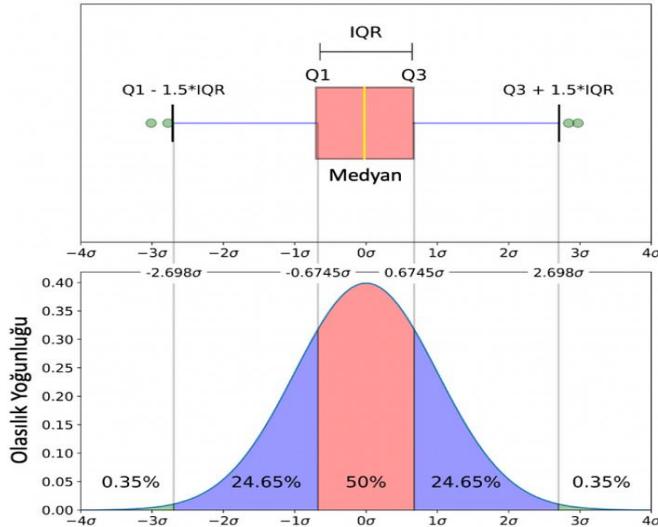
Bir kutu grafiği (Boxplot), veri çeyreklerini (veya yüzdelikleri) ve ortalamaları görüntüleyerek sayısal verilerin ve değişkenliğin görsel olarak dağılımını göstermek için kullanılır. Veri analizinde sıkılıkla kullanılan bir grafik türüdür.

Kutu grafikleri, bir veri kümesinin beş özelliğini gösterir: minimum değer, ilk (%25) çeyrek, medyan, üçüncü (%75) çeyrek ve maksimum değer. Bu veri setini çeyrekler olarak da bilinen dört eşit parçaya bölün beş değerden oluşur.

- Minimum değer:** Veri kümesindeki en küçük değer.
- İlk çeyrek (Q1):** Verilerin en düşük %25'ini veri setinin geri kalanından ayıran değer.
- Medyan (Q2):** Verilerin en düşük %50'sini en yüksek %50'sinden ayıran değer.
- Üçüncü çeyrek (Q3):** Verilerin en düşük %75'ini en yüksek %25'inden ayıran değer.
- Maksimum değer:** Veri setindeki en büyük değer.

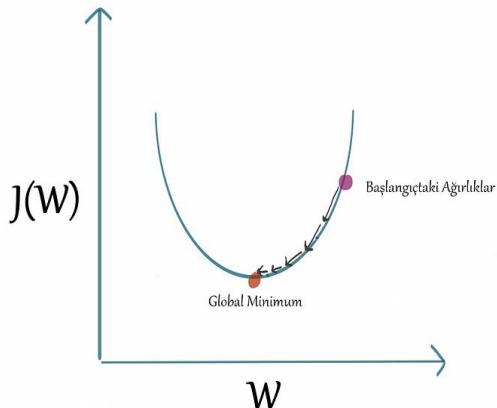


Normal dağılım eğrisiyle kutu grafiğini bir arada görmek isterseniz aşağıdaki grafik bunu gösteriyor. IQR değeri dağılım eğrisindeki tepe noktasını ve bunun etrafındaki %50'lik dilimi gösterir.



Aykırı değerler iç çitlere düşmeyen çok uç değerlerdir. Bunlar, değerleri kutuların yüksekliğinin üç katından daha fazla olan durumları/satırları temsil eder.

### Gradient Descent (Gradyan Azalma)



Gradient Descent (Gradyan Azalma), rastgele alınan değişkenlerle başlayarak global minimum değerine ulaşmayı amaçlar. Adım adım anlatmak gerekirse:

1-) Her parametre için Kayıp Fonksiyonun (loss function) türevini al.

Maliyet Fonksiyonu:

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2$$

Kuralların güncellenmesi:

$$\begin{aligned}\theta_0 &:= \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) \\ \theta_1 &:= \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)\end{aligned}$$

Türevi:

$$\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})$$

$$\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) \cdot x^{(i)}$$

2-) Parametreler için rastgele değerler topla ve toplanan değerleri parametrelere gönder.

3-) Adım büyüklüğü (step size) hesapla.

$$\text{Adım Büyüklüğü} = \text{Eğim} \times \text{Öğrenme Oranı}$$

$$\text{Step Size} = \text{slope} \times \text{learning rate}$$

4-) Yeni parametreyi hesapla.

$$\text{Yeni Parametre} = \text{Eski Parametre} - \text{Adım Büyüklüğü}$$

$$\text{New Parameter} = \text{Old Parameter} - \text{Step Size}$$

5-) Minimum seviyeye ulaşışıya kadar 2. adıma dön.

**y kesişim için:**

$$\text{MSE}_{\text{y kesişim}} = (-2(1,4 - (\text{y kesişim} + \text{eğim} \times 0,5))) + (-2(1,9 - (\text{y kesişim} + \text{eğim} \times 2,3))) + (-2(3,2 - (\text{y kesişim} + \text{eğim} \times 2,9)))$$

**Eğim için:**

$$\text{MSE}_{\text{eğim}} = (-2 \times 0,5(1,4 - (\text{y kesişim} + \text{eğim} \times 0,5))) + (-2 \times 2,3(1,9 - (\text{y kesişim} + \text{eğim} \times 2,3))) + (-2 \times 2,9(3,2 - (\text{y kesişim} + \text{eğim} \times 2,9)))$$

Bu modelde 1000. adımdan sonra modelimiz durur. Bir dakika! Neden?

Gradyan Azalmanın durması için ya adım büyüklüğünün sıfıra çok yaklaşması gereklidir (adım büyülüklüğü < 0,001 olmalı) ya da daha önceden adım sayısını sınırlayabiliriz (genellikle 1000 sayısı kullanılır). Örnekte adım büyülüklüğü 0'a çok yaklaşıyor.



Buradaki 3 veri ile bu işlemleri yapmak oldukça kolaydı. Peki ya oldukça büyük bir veri seti ile işlem yapmak istediğimizde bütün işlemleri çok uzun sürmeyecek mi?

-Evet, çok uzun sürecek, ama onunda bir çözümü var.

Şuana kadar anlattığım Gradient Descent, **Batch Gradient Descent** olarak geçiyor ve tüm veri setinde işlem yapıyor. Bu durum, veri setimiz büyükse bizi yavaşlatır. Bunun yerine **Stochastic Gradient Descent** veya **Mini-Batch Gradient Descent** kullanabiliriz.

**Stochastic Gradient Descent:**

Her adımda rastgele alınan 1 veri üzerinde işlem yapar. Noktalar sürekli değişerek optimum noktaya ulaşmayı amaçlar. Batch Gradient Descent'e göre daha hızlıdır.

Stochastic Gradient Descent ile iyi bir sonuca ulaşılır ama optimum sonuca ulaşamayabilir.

### Mini-Batch Gradient Descent:

Veri seti içerisinde rastgele alınan örneklemeler ile öğrenir. Diğer ikisinden daha hızlıdır. Stochastic Gradient Descent'e göre daha dengeli ilerler, onun kadar çok savrulmaz.

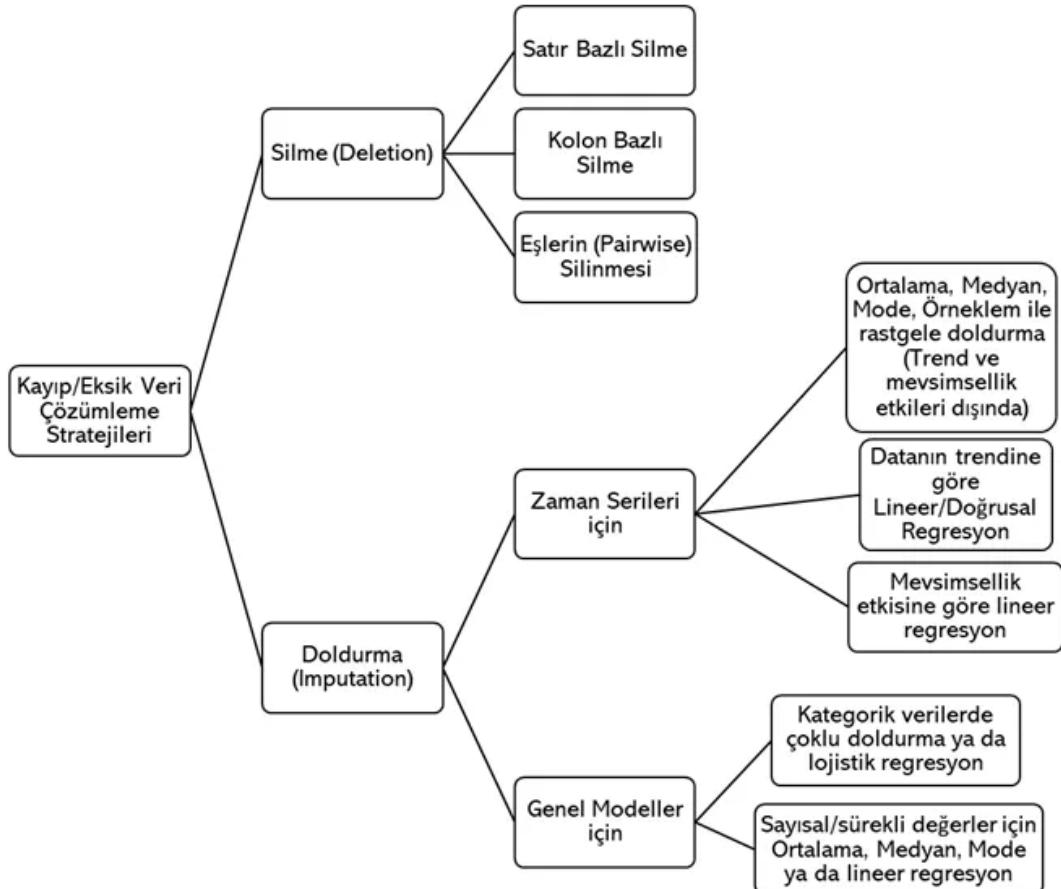
**Steepest Descent Algoritması**, bir optimizasyon yöntemidir ve bir fonksiyonun en düşük (minimum) değerini bulmak için kullanılan bir **gradyan tabanlı** algoritmadır. Genellikle **Gradyan İnişi (Gradient Descent)** olarak da bilinir. Bu yöntem, bir fonksiyonun eğimini (gradyanını) kullanarak, o fonksiyonun en hızlı azalacağı yönü belirler ve adım adım bu yönde ilerleyerek minimum değeri bulmaya çalışır.

### Steepest Descent Algoritmasının Adımları:

1. Bir başlangıç noktası  $x_0$  seç.
2. Gradyanı  $\nabla f(x_k)$  hesapla.
3. Adım büyüklüğünü  $\alpha$  belirle.
4. Yeni noktayı  $x_{k+1} = x_k - \alpha \nabla f(x_k)$  ile hesapla.
5. Gradyanın yeterince küçük olup olmadığını kontrol et. Eğer küçükse dur, değilse adım 2'ye geri dön.

Özellik	Gradient Descent	Steepest Descent
<b>Adım Büyüklüğü (Learning Rate)</b>	Sabit veya dinamik bir adım büyüğü kullanır. Genellikle sabit adım büyüğü seçilir.	Her iterasyonda <b>optimal adım büyüğü</b> hesaplanır ve kullanılır.
<b>Adım Hesaplama</b>	Her iterasyonda aynı yönde ve genellikle sabit adımlarla ilerler.	Her iterasyonda gradyan boyunca en iyi adım uzunluğunu bulur.
<b>Hız</b>	Uygun olmayan adım büyüğü seçilirse, daha yavaş yakınsama olabilir.	Optimal adım büyüğü her seferinde hesaplandığından, daha hızlı bir yakınsama olabilir.
<b>Hesaplama Yükü</b>	Hesaplama yükü daha düşüktür çünkü adım büyüğü genellikle sabittir.	Her iterasyonda adım büyüğünü hesaplamak için ek optimizasyon gerektirir, bu da daha yüksek hesaplama yükü oluşturur.
<b>Kullanım Alanı</b>	Daha genel ve basit optimizasyon problemlerinde kullanılır.	Hesaplaması daha yoğun olduğundan, daha karmaşık ve hassas problemlerde tercih edilir.

Datayı önünüze koyduğunuzda ilk kontrol noktalarınızdan birisi eksik/kayıp veri (missing value) bulunup bulunmadığı. Bu problem ile gerçek hayatı verilerde karşılaşmamız çok olasıdır. Bu duruma sebep olarak bazı operasyonel (eksik veri girişi) ya da sistemsel hatalar örnek gösterilebilir. İster makine öğrenme algoritmaları için olsun ister zaman serilerinde bu problem ile yaygın bir şekilde karşılaşabilirsiniz.



## Eksik Verilerin Doldurulması

**Alt, Üst veya Sabit Değer ile Doldurma:** Kolonda eksik verinin bulunduğu alanın altında ya da üstünde yer alan değeri ile doldurma işlemi yapılabilir. Sabit değer ise tamamen datanın nasıl yorumlandığına bağlı olarak eksik değerlere sizin belirleyeceğiniz bir sayı atanır. Bu yöntem özellikle ardışık olduğundan emin olduğunuz değişkenler için geçerli denilebilir. Yine de datayı işlediğiniz alandaki tecrübenizden yola çıkarak alt, üst ya da sabit değer atamanız daha mantıklı olacaktır.

**Ortalama, Medyan, Tepe Değer(Mode):** Değişkenin genel gidişatına bakarak bu yöntemlerden birisi kullanarak eksik veri doldurulabilir. Çoğunlukla veri bilimi alanında bu yöntemler tercih edilmektedir. Tek dezavantajı ise değişkenin varyans değerini düşürdüğü söylenebilir. Veri setindeki örneklerle inceleyelim;

**Lineer (Doğrusal) Regresyon:** Aslında bu yöntem tamamen regresyon modelinin eksik verilere uygulanmış halidir. Eksik değerlerin yer aldığı kolon bağımlı değişken olarak ve diğer kolonlar bağımsız değişken olarak tanımlanarak kayıp veriler tahmin edilmeye çalışılır. Bu yöntemin dezavantajlarından birisi, eksik verilerin regresyon ile diğer değişkenlere göre doldurulacak olmasından dolayı asıl tahmin edilecek değişken üzerinde veri setinde aşırı öğrenme (overfitting) söz konusu olabilir.

**Kategorik Verilerde Doldurma:** Genel olarak üç yöntemden söz edilebilir.

1. Tepe değer(mode) kullanılarak en fazla frekansa sahip olan değer ile eksik veriler doldurulabilir.
2. Eksik/kayıp veriler etiketlenerek model ya da analiz için değerlendirilebilir.
3. Lojistik regresyon ya da ANOVA gibi yöntemler ile eksik değerler tahmin edilebilir.

## Aktivasyon fonksiyonları neden kullanılır?

Sinir ağlarında aktivasyon fonksiyonları, bir nöronun aldığı girdilere nasıl yanıt vereceğini belirlemek için kullanılır. Aktivasyon fonksiyonu, nöronlara gelen girdileri işler ve bu girdilere göre bir çıkış üretir. Aktivasyon fonksiyonunun kullanılmasının ana nedenleri şunlardır:

1. **Doğrusal olmayanlık ekleme:** Sinir ağları, katmanlar arasında doğrusal ilişkiler olmadan öğrenmeyi sağlar. Eğer aktivasyon fonksiyonları olmasaydı, her katmandan sonra sadece doğrusal bir işlem yapılpırdı ve ağ, ne kadar derin olursa olsun, doğrusal bir model gibi davranışındı. Aktivasyon fonksiyonları, ağı doğrusal olmayan problemleri çözebilir hale getirir.
2. **Nöronları ateşlemeye karar verme:** Aktivasyon fonksiyonları, bir nöronun aktif olup olmamasına karar vermeye yardımcı olur. Örneğin, ReLU (Rectified Linear Unit) fonksiyonu, negatif değerleri sıfıra indirir, pozitifleri olduğu gibi bırakır. Bu sayede bazı nöronlar pasif hale gelir ve modelin daha verimli çalışması sağlanır.
3. **Çıktıları normalleştirme:** Sigmoid veya softmax gibi aktivasyon fonksiyonları, çıktıların belirli bir aralıkta (örneğin, sigmoid 0 ile 1 arasında) olmasını sağlar. Bu özellikle sınıflandırma problemlerinde faydalıdır, çünkü çıktıların olasılık gibi yorumlanmasını sağlar.
4. **Geri yayılımda türev hesaplaması:** Aktivasyon fonksiyonlarının türevleri, sinir ağının geri yayılım algoritmasında kullanılır. Öğrenme sırasında ağıın hatalarını minimize etmek için türevler hesaplanarak ağırlıklar güncellenir.

Sinir ağlarında overfitting, modelin eğitim verisine aşırı derecede uyum sağladığında ve yeni, görülmemiş verilerle iyi genelleme yapamadığında ortaya çıkar. Overfitting'i önlemek için kullanılan yaygın yöntemler şunlardır:

### 1. Daha Fazla Veri Toplama

- **Açıklama:** Eğitim verisini artırmak, modelin genel kalıpları öğrenmesine ve spesifik örnekler üzerinde ezber yapmamasına yardımcı olur.
- **Yöntemler:** Yeni veri toplamak veya mevcut verileri artırma (data augmentation) yöntemleri kullanarak yapay olarak veri kümесini büyütmek.

### 2. Veri Artırma (Data Augmentation)

- **Açıklama:** Görüntü, metin veya ses gibi veri türlerinde, veri kümесinin yapay olarak genişletilmesiyle modelin daha çeşitli verilerle eğitilmesi sağlanır.

- **Yöntemler:** Görüntü verileri için döndürme, ölçekleme, kırpma gibi işlemler yapılabilir. Metin veya zaman serisi verileri için gürültü eklemek veya çarpıtma gibi teknikler kullanılabilir.

### 3. Düzenlileştirme (Regularization)

- **L1 ve L2 Düzenlileştirme:**
  - **L1 (Lasso):** Ağırlıkların toplamının mutlak değerinin cezalandırılması. Bu, bazı ağırlıkları sıfıra yaklaştırarak modelin daha sade olmasına yol açar.
  - **L2 (Ridge):** Ağırlıkların karelerinin toplamının cezalandırılması. Bu da ağırlıkları küçültür, ama sıfıra kadar indirmez.
- **Açıklama:** Düzenlileştirme, modelin aşırı uyum yapmasını engelleyerek, parametrelerin büyümeyi kontrol altında tutar.

### 4. Dropout

- **Açıklama:** Dropout, eğitim sırasında her iterasyonda rasgele nöronları devre dışı bırakarak (örneğin %20-50), modelin belirli bir nörona aşırı bağımlı olmasını önler ve farklı özellik kombinasyonlarını öğrenmeye teşvik eder.
- **Yöntem:** Eğitim sırasında her katmandaki nöronların belli bir kısmını rastgele kapatmak, test sırasında tüm nöronları kullanarak son tahminleri yapmak.

### 5. Erken Durdurma (Early Stopping)

- **Açıklama:** Modelin doğrulama veri kümesi üzerindeki performansını izleyerek, belirli bir noktadan sonra doğrulama hatası artmaya başlarsa eğitimi durdurmak.
- **Yöntem:** Eğitim sırasında doğrulama hatasını izleyip, model doğrulama setinde en iyi performansı gösterdiğinde eğitimi sonlandırmak.

### 6. Daha Küçük ve Daha Sade Bir Model Kullanma

- **Açıklama:** Modelin karmaşıklığı arttıkça overfitting riski de artar. Bu nedenle, daha az katmanlı veya daha az nöronlu bir model seçmek, gereksiz karmaşıklıkları azaltabilir.
- **Yöntem:** Aşırı büyük sinir ağı yapılarından kaçınmak ve daha sade model mimarilerine yöneltmek.

### 7. Çapraz Doğrulama (Cross-Validation)

- **Açıklama:** Eğitim verilerini K-katlamalı çapraz doğrulama ile ayırarak, modelin farklı veri kümelerinde nasıl performans gösterdiğini anlamak.
- **Yöntem:** Veriyi farklı alt kümelere ayırip her alt kümede modeli eğitip test etmek. Bu sayede modelin genelleme yeteneği değerlendirilir.

### 8. Batch Normalizasyonu

- **Açıklama:** Her katmandaki aktivasyonları normalleştirerek, daha dengeli ve daha stabil bir öğrenme süreci sağlar. Bu da overfitting'i azaltabilir.
- **Yöntem:** Modelin her katmanındaki aktivasyonlarının normalizasyonunu sağlayan batch normalization katmanları eklemek.

- Batch normalization sayesinde ağıdaki katmanlar, önceki katmanın öğrenmesini beklemek zorunda kalmaz. Eş zamanlı olarak öğrenime olanak sağlar. Eğitimimizin hızlanması sağlanır.

### Batch Normalizasyonunun Detayları

- Amaç:** Sinir ağlarında, katmanlardaki aktivasyonların dağılımı eğitim boyunca değişimdir (bu duruma "internal covariate shift" denir). Bu, her katmanda giriş verilerinin sürekli değişmesine neden olur ve öğrenme sürecini zorlaştırır. Batch normalization, her mini batch'teki girişleri normalleştirerek (ortalama ve standart sapma ile ölçekleme), bu sorunu azaltır.
- Nasıl Çalışır?:**

- Eğitim sırasında her mini batch'teki aktivasyonlar (katman çıktıları) için ortalama  $\mu$  ve standart sapma  $\sigma$  hesaplanır.

- Aktivasyonlar, bu ortalama ve standart sapmaya göre normalleştirilir:

$$\hat{x} = \frac{x - \mu}{\sigma}$$

- Bu normalleştirilmiş aktivasyonlar, sonra ölçeklenir ve kaydırılır. Öğrenme sırasında bu iki parametre ( $\gamma$  ve  $\beta$ ) öğrenilir:  $y = \gamma \hat{x} + \beta$
- Bu işlem, her katmandaki aktivasyonların dengeli olmasını ve modelin daha hızlı öğrenmesini sağlar.

$$\hat{x} = \frac{x - \mu}{\sigma}$$

## 9. Veri Kümesini Düzenlemek

- Açıklama:** Veri kümesindeki hatalar, tutarsızlıklar veya aşırı veri gürültüsü, modelin yanlış kalıplar öğrenmesine neden olabilir. Veriyi temizlemek, overfitting riskini azaltabilir.
- Yöntem:** Verileri inceleyip hatalı etiketler, gürültülü veriler veya uyumsuzlukları düzeltmek.

### 1. Intersection over Union (IoU)

#### IoU Nedir?

IoU, modelin tahmin ettiği sınır kutusu (bounding box) ile gerçek (ground truth) sınır kutusu arasındaki örtüşme oranını ölçer. Bu oran, modelin nesneyi ne kadar doğru tespit ettiğini anlamak için kullanılır. IoU, tahmin edilen kutunun doğruluğunu değerlendirmek için kullanılan en temel ölçümlerden biridir.

#### IoU Nasıl Hesaplanır?

IoU, tahmin edilen sınır kutusunun gerçek sınır kutusuyla ne kadar örtüştüğünü şu şekilde hesaplar:

$$\text{IoU} = \frac{\text{Tahmin edilen kutu} \cap \text{Gerçek kutu}}{\text{Tahmin edilen kutu} \cup \text{Gerçek kutu}}$$

Burada  $\cap$ , tahmin edilen kutu ile gerçek kutu arasındaki kesişim bölgesini,  $\cup$  ise iki kutunun birleşimini ifade eder. Sonuç, 0 ile 1 arasında bir değer alır:

- **IoU = 1:** Tahmin edilen kutu ve gerçek kutu tamamen örtüşüyor.
- **IoU = 0:** Tahmin edilen kutu ve gerçek kutu hiç örtüşmüyork.

IoU değerine bağlı olarak tahmin edilen kutu doğru kabul edilir veya edilmez. Genelde  $\text{IoU} \geq 0.5$  olan tahminler doğru kabul edilir, ancak bu eşik uygulamaya bağlı olarak değişebilir (örneğin,  $\text{IoU} \geq 0.75$  daha yüksek hassasiyet gerektiren görevlerde tercih edilebilir).

## 2. Average Precision (AP)

### AP Nedir?

AP, modelin bir nesne sınıfı için tespit ettiği nesnelerin doğruluğunu ve hassasiyetini ölçmek için kullanılır. AP, bir nesne sınıfı için "Precision" ve "Recall" (Doğruluk ve Geri Çağırma) değerlerinin integralini alarak hesaplanan bir metriktir. Precision, modelin ne kadar doğru tahminler yaptığını; Recall ise modelin ne kadar çok doğru tahmin yapabildiğini gösterir.

### Precision ve Recall Nasıl Hesaplanır?

- **Precision (Doğruluk):** Doğru pozitif tahminlerin, tüm pozitif tahminlere oranıdır.

$$\text{Precision} = \frac{\text{Doğru Pozitif (True Positives)}}{\text{Doğru Pozitif (True Positives)} + \text{Yanlış Pozitif (False Positives)}}$$

- **Recall (Geri Çağırma):** Doğru pozitif tahminlerin, tüm doğru pozitif olabilecek nesnelere oranıdır.

$$\text{Recall} = \frac{\text{Doğru Pozitif (True Positives)}}{\text{Doğru Pozitif (True Positives)} + \text{Yanlış Negatif (False Negatives)}}$$

### AP Nasıl Hesaplanır?

AP, Precision ve Recall değerlerinin değişen eşiklerde hesaplanmasıyla elde edilir. Yani, modelin farklı eşiklerde yaptığı tahminlere göre precision ve recall değerleri hesaplanır ve bu noktalar kullanılarak bir Precision-Recall eğrisi çizilir. AP, bu eğrinin altındaki alanın integralidir.

- **AP Calculation Adımları:**

1. Modelin tahminlerini confidence score (güven skoru) ile sıralayın.
2. Her bir tahmin için precision ve recall değerlerini hesaplayın.
3. Precision-Recall eğrisini oluşturun.
4. Precision-Recall eğrisinin altındaki alanı hesaplayın (genellikle trapezoidal rule ile).

### 3. Mean Average Precision (mAP)

#### mAP Nedir?

mAP, modelin farklı sınıflardaki nesne tespiti performansının genel bir ölçüsüdür. Model birden fazla nesne sınıfı üzerinde çalışıyorsa (örneğin, insanlar, arabalar, hayvanlar gibi), her sınıf için AP değerleri hesaplanır ve bu AP değerlerinin ortalaması alınarak **mean Average Precision (mAP)** bulunur.

#### mAP Nasıl Hesaplanır?

1. Her sınıf için AP hesaplanır (yani, her sınıf için precision-recall eğrisi ve alanı).
2. Tüm sınıflar için hesaplanan AP değerlerinin ortalaması alınır:

$$mAP = \frac{1}{N} \sum_{i=1}^N AP_i$$

#### 3. mAP@IoU

YOLO gibi nesne tespit modellerinde, mAP genellikle belirli IoU eşiklerinde hesaplanır. Örneğin:

- **mAP@0.5**: IoU  $\geq 0.5$  olan tahminler doğru kabul edilerek mAP hesaplanır.
- **mAP@0.75**: IoU  $\geq 0.75$  olan tahminler kullanılarak mAP hesaplanır.
- **mAP@[0.5:0.95]**: Farklı IoU eşiklerinde (örneğin, 0.5'ten 0.95'e kadar) mAP hesaplanır ve ortalaması alınır. Bu, modelin performansını daha geniş bir aralıkta değerlendiren daha zorlayıcı bir metrik olarak kabul edilir.

### YOLO Modeli İçin Değerlendirme Süreci

1. **Tahmin edilen bounding box'lar ve confidence score'lar**: Model, her tespit için bir sınır kutusu (bounding box) ve bu kutunun doğru bir tespit olup olmadığını belirten bir güven skoru döndürür.
2. **IoU hesaplaması**: Modelin tahmin ettiği her bir sınır kutusu için IoU hesaplanır.
3. **Doğru ve yanlış pozitiflerin belirlenmesi**: IoU eşik değerine (genellikle 0.5) bağlı olarak, tahminler doğru pozitif (True Positive) ya da yanlış pozitif (False Positive) olarak sınıflandırılır.
4. **Precision, recall ve AP hesaplaması**: Her sınıf için precision-recall eğrisi oluşturulur ve AP değeri hesaplanır.
5. **mAP hesaplaması**: Tüm sınıflar için AP değerleri ortalaması alınarak modelin genel performansı mAP ile özetlenir.

Markov Karar Süreçleri (MDP), Bellman Denklemi ve Policy Search, genellikle yapay zeka ve pekiştirmeli öğrenme (reinforcement learning) gibi alanlarda karar verme süreçlerini modellemek ve çözmek için kullanılan önemli kavamlardır. Bu üç kavram birbirleriyle yakından ilişkilidir ve bir ajanın (agent) belirsizlik altında optimal kararlar almasını sağlayan mekanizmalar oluşturur. Şimdi her bir kavramı açıklayalım.

## 1. Markov Karar Süreci (Markov Decision Process - MDP)

MDP, bir ortamda bir ajanın karar verme problemini modellemek için kullanılan matematiksel bir çerçevedir. MDP, ajanların farklı durumlar (states) arasında hareket ederken, eylemler (actions) seçip, ödüller (rewards) kazanarak en iyi stratejiyi bulmasına olanak tanır.

### MDP'nin Bileşenleri:

- **Durumlar ( $S$ ):** Ajanın içinde bulunduğu ortamın durumlarını temsil eder.
- **Eylemler ( $A$ ):** Ajanın her durumda seçebilecegi bir dizi eylem.
- **Geçiş Olasılıkları ( $P(s'|s, a)$ ):** Ajanın bir eylem gerçekleştirdikten sonra hangi duruma geçeceğini belirleyen olasılık dağılımı.
- **Ödül Fonksiyonu ( $R(s, a)$ ):** Ajanın bir durumda bir eylem gerçekleştirdiğinde aldığı ödül.
- **Politika ( $\pi(s)$ ):** Ajanın hangi durumda hangi eylemi seçeceğini belirleyen strateji.

MDP'nin amacı, ajan için en iyi politikayı bulmaktır; yani ajan her durumda hangi eylemi seçmeli ki uzun vadede en yüksek ödülü alınsın.

### Markov Özelliği:

Markov özelliği, MDP'deki geçişlerin sadece mevcut duruma ve eyleme bağlı olduğunu, önceki durumların ve eylemlerin doğrudan etkisinin olmadığını ifade eder. Yani, geçmiş bilmek gereksizdir; mevcut durum gelecekle ilgili tüm bilgiyi taşır.

---

## 2. Bellman Denklemi (Bellman Equation)

Bellman Denklemi, dinamik programlama kullanarak MDP'yi çözmek için kullanılan temel bir denklem olup, bir durumun değerini (value) tanımlayan bir reküratif ilişki kurar. Bellman Denklemi, bir ajanın belirli bir politikaya göre her durumda gelecekteki ödülünü en üst düzeye çıkarmak için hangi eylemleri seçmesi gerektiğini belirlemeye yardımcı olur.

### Durum Değeri (State Value Function - $V(s)V(s)V(s)$ ):

Durum değeri fonksiyonu  $V(s)$ , bir ajanın belirli bir durumda başlayıp gelecekte almayı beklediği toplam ödülü ifade eder. Bellman Denklemi bu durumu reküratif olarak şu şekilde tanımlar:

$$V(s) = \max_a \left( R(s, a) + \gamma \sum_{s'} P(s'|s, a)V(s') \right)$$

- $V(s)$ : Durum  $s$ 'de başlanıldığından beklenen toplam ödül.
- $R(s, a)$ : Durum  $s$ 'de eylem  $a$ 'yı gerçekleştirdiğinde alınan ödül.
- $P(s'|s, a)$ : Ajanın durum  $s$ 'den  $a$  eylemini gerçekleştirdiğinde durum  $s'$ ye geçme olasılığı.
- $\gamma$ : İndirim faktörü (discount factor), gelecekteki ödüllerin ne kadar önemli olduğunu belirler.  $0 \leq \gamma \leq 1$ .

Bu denklemin anlamı şudur: Ajan, bir durumda alabileceği doğrudan ödülü ve gelecekteki ödüllerin indirgenmiş değerini maksimize etmeye çalışır.

#### **Q-Değeri (Q-Value Function):**

Bir başka yaygın fonksiyon da Q-değer fonksiyonudur. Q-değer fonksiyonu, belirli bir durumdayken bir eylem gerçekleştirildiğinde beklenen toplam ödülü hesaplar.

$$Q(s, a) = R(s, a) + \gamma \sum_{s'} P(s'|s, a)V(s')$$

Q-değer fonksiyonu, duruma ve eyleme bağlı olarak gelecekteki ödüllerin ne kadar olacağını hesaplar. Bellman denkleminin bu hali, genellikle pekiştirmeli öğrenme algoritmalarında kullanılır.

#### **Bellman Denkleminin Amacı:**

Bellman Denklemi, optimal politikayı (optimal policy -  $\pi^*$ ) bulmak için kullanılır. Optimal politika, ajanın uzun vadede maksimum ödülü elde etmesini sağlayacak olan stratejidir. Bu politika, Bellman Denklemi ile tanımlanan optimal değer fonksiyonuna dayalıdır.

---

### **3. Politika Arama (Policy Search)**

Politika arama (policy search), MDP'lerde en iyi politikayı doğrudan bulmaya çalışan yöntemlerdir. Bu, optimal eylem seçimlerini belirlemek için politika fonksiyonunu optimize etmeyi amaçlar. Politika arama, genellikle iki ana yaklaşımla yapılır: **Politika Tabanlı Yöntemler** ve **Politika Gradyanı Yöntemleri**.

#### **Politika Tabanlı Yöntemler:**

- **Açıklama:** Politika tabanlı yöntemler, bir politika fonksiyonunu doğrudan optimize eder. Bu fonksiyon, her durumda hangi eylemi seçileceğini belirler.
- **Detaylar:** Politika fonksiyonu  $\pi(a|s)$ , bir durum  $s$ 'de eylem  $a$ 'nın seçilme olasılığını belirleyen bir olasılık fonksiyonudur. Amaç, bu politika fonksiyonunu parametrelerini optimize ederek en yüksek toplam ödül sağlayan politikayı bulmaktır.

- **Avantajı:** Politika tabanlı yöntemler, sürekli eylem alanlarıyla başa çıkmada ve deterministik politikaları öğrenmede daha iyidir.

#### Politika Gradyanı Yöntemleri (Policy Gradient Methods):

- **Açıklama:** Politika gradyanı yöntemleri, ödül fonksiyonunu maksimize etmek için politika parametrelerinin gradyanını hesaplar ve bu gradyanı kullanarak politika parametrelerini optimize eder.
- **Detaylar:** Politika  $\pi_\theta(s, a)$ , parametreler  $\theta$  ile tanımlanmış bir olasılık fonksiyonudur. Bu yöntemler,  $J(\theta)$  olarak ifade edilen toplam ödül fonksiyonunu maksimize etmek için şu gradyanı hesaplar:

$$\nabla_\theta J(\theta) = \mathbb{E} [\nabla_\theta \log \pi_\theta(a|s) Q(s, a)]$$

Daha sonra, bu gradyan kullanılarak politika parametreleri güncellenir.

- **Avantajı:** Politika gradyanı yöntemleri, deterministik politikalar oluşturabilir ve sürekli eylem alanlarında daha etkili olabilir.

#### Politika Aramanın Avantajları:

- **Doğrudan politika optimizasyonu:** Bazı yöntemler durumu ve eylemleri öğrenmeden, doğrudan optimal politikayı öğrenmeye çalışır. Bu, özellikle büyük ve karmaşık durum uzaylarında avantajlı olabilir.
- **Deterministik ve Stokastik Politikalar:** Politika arama yöntemleri, hem deterministik (belirli bir durumda her zaman aynı eylemi seçten) hem de stokastik (eylem olasılığına dayalı) politikalar bulabilir.

---

## Sonuç

- **Markov Karar Süreci (MDP):** Bir ajanın belirsizlik altında karar alma problemini tanımlar. Amaç, en iyi politikayı bulmaktır.
- **Bellman Denklemi:** Dinamik programlama ile optimal politikanın bulunmasında kullanılan rekürsif bir denklem. Ajanın her durumda gelecekte alacağı ödüllü en üst düzeye çıkmasını sağlar.
- **Politika Arama (Policy Search):** MDP'yi çözmek için doğrudan optimal politikayı arayan yöntemlerdir. Politika tabanlı ve politika gradyanı yöntemleri, politikayı doğrudan optimize etmeye odaklanır.

## 1. YOLO (You Only Look Once) ile Nesne Tespiti Soruları

### a. Teknik Sorular

Bu alanda deneyimlerini anlattığında, mülakat yapan kişi sana YOLO ile ilgili derinlemesine teknik sorular sorabilir. Örnek sorular:

#### 1. YOLO'nun nasıl çalıştığını anlatır mısınız?

- Bu soruda, YOLO'nun temel çalışma prensibini, grid tabanlı nesne tespiti yaklaşımını, bounding box ve class olasılıklarının nasıl hesaplandığını detaylı bir şekilde açıklaman beklenir.
- **Beklenen Yanıt:** YOLO, görüntüyü gridlere böler ve her bir gridde nesne olup olmadığını kontrol eder. Her grid, nesne merkezine göre bounding box'lar ve sınıf olasılıkları tahmin eder.

#### 2. YOLO v4 ve YOLO v5 arasındaki farkları açıklayın.

- Bu soruda, YOLO v4 ve v5'in karşılaştırmasını yapman ve teknik detaylarını belirtmen beklenir. Örneğin, YOLO v4'ün CSPDarknet53 gibi gelişmiş backbone yapısı kullanması ve YOLO v5'in daha hafif ve hızlı bir yapıda PyTorch ile geliştirildiğini anlatabilirsın.
- **Beklenen Yanıt:** YOLO v4, daha ağır ve büyük uygulamalar için optimize edilmiştir, YOLO v5 ise daha hafif ve daha hızlı çalışır. YOLO v5 daha esnek olup, düşük donanımla çalışmaya daha uygundur.

#### 3. YOLO'da anchor boxes'in ne işe yaradığını anlatır mısınız?

- **Beklenen Yanıt:** Anchor boxes, farklı boyutlardaki nesneleri tespit etmek için önceden belirlenmiş kutulardır. YOLO, her bir grid hücrende bu anchor boxes'ları kullanarak nesneleri tahmin eder. Bu sayede daha verimli ve doğru tespitler yapılır.

#### 4. YOLO'da IoU nedir ve nasıl kullanılır?

- **Beklenen Yanıt:** IoU (Intersection over Union), tahmin edilen bounding box ile gerçek bounding box arasındaki örtüşmenin oranını gösterir. IoU değeri 1'e ne kadar yakınsa tahmin o kadar doğrudur. IoU, nesne tespiti algoritmalarında modelin doğruluğunu değerlendirmek için kullanılır.

#### 5. Non-Maximum Suppression (NMS) nedir ve neden gereklidir?

- **Beklenen Yanıt:** NMS, nesne tespiti sırasında birden fazla overlapping (örtüşen) bounding box üretildiğinde en yüksek güven skoruna sahip olanı seçer ve diğer kutuları eler. Bu, aynı nesne için birden fazla tespit yapılmasını önler.

### b. Problem Soruları

Şirket, bu teknolojiyi nasıl uygulayacağınızı veya çözüm üretebileceğinizi anlamak için pratik sorular da sorabilir:

#### 1. Bir görüntüde nesneleri tespit etmek için YOLO v5 kullanarak bir model geliştirin ve eğitin. Eğitim sırasında kontrol etmeniz gereken parametreler nelerdir?

- **Beklenen Yanıt:** Eğitim sırasında kontrol edilmesi gereken parametreler arasında anchor boxes, batch size, image size, learning rate, epoch sayısı gibi faktörler vardır. Ayrıca loss fonksiyonunun bileşenlerini takip ederek, modelin doğru eğitilip eğitilmediğini kontrol etmelişin.

#### 2. YOLO v5'i kullanarak bir nesne tespit modeli eğittiniz, modelin test doğruluğu %90 ancak validation doğruluğu %70 çıktı. Ne yaparsınız?

- **Beklenen Yanıt:** Bu durum overfitting işaretidir. Bu problemi çözmek için data augmentation kullanabilir, model complexity'yi düşürebilir, dropout ekleyebilir veya regularization gibi yöntemler uygulayarak validation doğruluğunu artırmayı deneyebilirsin.

## 2. Reinforcement Learning ile Engelden Kaçınan Robot Simülasyonu Soruları

### a. Teknik Sorular

Reinforcement learning ile ilgili deneyiminden bahsettiğinde, aşağıdaki gibi sorularla karşılaşabilirsin:

#### 1. Reinforcement learning nedir ve nasıl çalışır?

- **Beklenen Yanıt:** Reinforcement learning, bir ajanın çevresiyle etkileşimde bulunarak ödül elde etmeye çalıştığı bir öğrenme metodudur. Ajan, bir durumdan bir aksiyon alır, bu aksiyon sonucunda bir ödül alır ve öğrenme süreci bu döngü içinde gerçekleşir.

#### 2. State, Action, Reward kavramlarını açıklayın.

- **Beklenen Yanıt:**
  - **State (Durum):** Ajanın içinde bulunduğu ortamın mevcut durumu.
  - **Action (Aksiyon):** Ajanın duruma göre yapabileceği hareketler.
  - **Reward (Ödül):** Ajanın aldığı aksiyon sonucunda aldığı geri bildirim.

#### 3. Markov Karar Süreçleri (MDP) nedir?

- **Beklenen Yanıt:** Markov Karar Süreçleri, reinforcement learning problemlerinde kullanılan bir çerçevedir. Bir ajanın aksiyon alarak ödül maksimize etmesini sağlar ve geçiş olasılıklarıyla tanımlanır.

#### 4. Q-learning ve SARSA arasındaki farklar nelerdir?

- **Beklenen Yanıt:** Q-learning, off-policy bir algoritmadır ve en iyi aksiyonları öğrenmeye çalışırken, SARSA on-policy'dir ve mevcut politikaya bağlı olarak aksiyon alır.

#### 5. DQN ve Q-learning arasındaki farklar nelerdir?

- **Beklenen Yanıt:** DQN, Q-learning'i derin öğrenme ile birleştirir ve Q-değerlerini tahmin etmek için bir derin sinir ağı kullanır. Q-learning'de Q-tablosu varken, DQN'de bu tablo yerine bir sinir ağı kullanılır.

### b. Problem Soruları

Reinforcement learning uygulamalarıyla ilgili gerçek problemleri çözme yeteneğini anlamak için şu sorular sorulabilir:

#### 1. Engelden kaçınan bir robot simülasyonu geliştirdiniz, ancak robot bazı durumlarda engellere çarpiyor. Bu durumu nasıl iyileştirebilirsiniz?

- **Beklenen Yanıt:** Robotun davranışlarını iyileştirmek için ödül fonksiyonunu gözden geçirebilir, ödül ve ceza oranlarını değiştirebilirim. Ayrıca exploration-exploitation dengesini optimize etmek için epsilon-greedy stratejisini iyileştirebilir, daha fazla eğitim verisiyle modelin performansını artırabilirim.

#### 2. Engelden kaçınan robot için reward fonksiyonunu nasıl tasarlarsınız?

- **Beklenen Yanıt:** Robot, bir engelden başarıyla kaçındığında pozitif bir ödül almalı, ancak engellere çarptığında negatif bir ödül almalıdır. Ayrıca, engelden ne kadar uzak olduğu ve hedefe ne kadar yaklaştığı da ödül fonksiyonuna eklenebilir.

### 3. Multi-Agent Reinforcement Learning (MARL) nedir ve nasıl çalışır?

- **Beklenen Yanıt:** MARL, birden fazla ajanın aynı ortamda öğrenme sürecine katıldığı bir yöntemdir. Ajanlar birlikte işbirliği yaparak ya da rekabet ederek ödüllerini maksimize etmeye çalışırlar.

## 3. Problemler ve Çözümler

Baykar, mülakat sırasında sana gerçek dünya senaryolarına dayalı bir problem verip, bunu nasıl çözebileceğini sorabilir. Olası problemler şunlar olabilir:

1. **Verilen bir görüntü setinde dronlar için nesne tespiti yapmanız isteniyor. Hangi algoritmayı kullanırdınız ve neden?**
  - **Beklenen Yanıt:** YOLO v5 kullanırdım çünkü hızlı çıkışım yapabilme kabiliyeti ve düşük donanım gereksinimi ile gerçek zamanlı nesne tespiti uygulamalarında idealdir. Ayrıca, kolaylıkla eğitilebilir ve nesne çeşitliliği yüksek uygulamalarda başarılı sonuçlar verir.
2. **Birden fazla robotun aynı anda bir engel parkurunda birbirine çarpmadan dolaşması gerekiyor. Hangi RL algoritmasını kullanırdınız ve nasıl bir strateji izlersiniz?**
  - **Beklenen Yanıt:** Multi-Agent Reinforcement Learning (MARL) kullanarak her robotun birbirile iletişim kurmasını sağlayan bir strateji geliştirilir. Robotların her biri ödüllerini maksimize etmeye çalışırken, çarpışmaları önlemek için koordineli bir şekilde hareket ederler.

### 1. Nesne Tespiti ile İlgili Problem Soruları

**Problem 1:** Bir drone uygulaması geliştiryorsunuz ve drone'un hem yerden hem havadan farklı nesneleri tespit etmesi gerekiyor. YOLO'y'u nasıl yapılandırırdınız ve nasıl bir strateji izlerdiniz?

- **Yanıt Önerisi:** İlk olarak, YOLO'nun havadan çekilen görüntülerde de yüksek doğrulukla çalışabilmesi için veritabanının uygun bir şekilde genişletilmesi gereklidir. Örneğin, drone kamerası ile çekilen yer ve hava görüntülerini üzerinde bir veri seti oluşturulmalıdır. YOLO'nun grid boyutlarını ve anchor box'larını bu nesnelerin boyutlarına göre optimize etmek önemlidir. Ek olarak, transfer learning kullanarak, önceden eğitilmiş bir modelin ağırlıklarıyla eğitim süresi kısaltılabilir. Eğitim sırasında aşırı uçları ve küçük nesneleri daha iyi tespit etmek için veri artırma (data augmentation) yöntemleri kullanılabilir.

**Problem 2:** Nesne tespit modelinizde bazı nesneler yüksek doğrulukla tespit ediliyor, ancak bazıları sürekli yanlış sınıflandırılıyor. Bu sorunu nasıl çözersiniz?

- **Yanıt Önerisi:** Bu durum, modelin belirli sınıflar arasında ayırt etme yeteneğinin düşük olmasından kaynaklanabilir. Bu sorunu çözmek için sınıflar arası denge sağlanarak veri artırma yöntemleri uygulanabilir. Ayrıca, tespit edilen bounding box'ların overlap sorununu düzeltmek için **Non-Maximum Suppression (NMS)** parametrelerini ayarlamak gerekebilir. Modelin eğitim sırasında belirli sınıflara yeterince odaklanabilmesi için daha dengeli bir veri seti sağlanmalı ve gerekiyorsa farklı nesne tespit algoritmalarıyla kıyaslamalar yapılabilir.

**Problem 3:** Bir görüntüde birden fazla nesne varken, tespit edilen bounding box'ların aşırı örtüşüğünü fark ediyorsunuz. Nasıl bir çözüm önerirsiniz?

- **Yanıt Önerisi:** Bu sorunu çözmek için **IoU (Intersection over Union)** ve **Non-Maximum Suppression (NMS)** parametrelerini optimize edebilirim. Özellikle NMS değerleri daha sıkı tutulursa, overlap olan bounding box'lar elenebilir. Ayrıca, anchor box'ların boyutları nesnelerin gerçek boyutlarına göre ayarlanarak daha doğru tespitler sağlanabilir.

**Problem 4:** YOLO modeliyle çalışırken küçük nesnelerin tespitinde zorlaniyorsunuz. Küçük nesneleri daha doğru tespit etmek için hangi yöntemleri kullanabilirsiniz?

- **Yanıt Önerisi:** Küçük nesnelerin tespitini iyileştirmek için **higher-resolution** (yüksek çözünürlüklü) görüntüler kullanabilir ve **image pyramid** gibi tekniklerle görüntü boyutlarını farklı seviyelerde analiz edebilirim. YOLO'nun grid boyutlarını küçülterek daha hassas tespitler yapmasını sağlayabilirim. Ayrıca, **data augmentation** ile küçük nesnelerin daha belirgin hale getirilmesi ve bu tür nesnelerin daha fazla temsil edildiği veri seti oluşturmak etkili olabilir.

**Problem 5:** Bir endüstriyel sahada, farklı makineleri tespit etmek için YOLO v5 kullanıyorsunuz. Makineler çok yakın olduğunda bazlarını ayırt edemiyor. Bu sorunu nasıl çözersiniz?

- **Yanıt Önerisi:** Öncelikle **anchor boxes**'ların yeniden hesaplanarak makinelerin boyutlarına göre optimize edilmesi gereklidir. Aynı zamanda, makinelerin çok yakın olduğu durumlarda veri setine çeşitli örnekler ekleyerek bu duruma modelin daha iyi uyum sağlaması sağlanabilir. Veritabanına daha fazla sınıf eklenmesi ve **fine-tuning** yapılarak sınıflar arası ayırmayı artırılabilir. Ayrıca, **NMS** ayarlarını optimize ederek çakışan bounding box'lar daha iyi ayırt edilebilir.

---

## 2. Reinforcement Learning ile İlgili Problem Soruları

**Problem 6:** Reinforcement learning ile bir robotun engellerden kaçmasını sağladınız, ancak robot bazen aynı engelden tekrar tekrar kaçınmak yerine duraksıyor. Bu sorunu nasıl çözersiniz?

- **Yanıt Önerisi:** Bu sorun, **exploration-exploitation** dengesinin iyi kurulmadığını veya robotun yerel minimuma takıldığı göstermektedir. Öncelikle, robotun daha fazla keşif yapabilmesi için **epsilon-greedy** stratejisi güncellenebilir ve epsilon değerini başlangıçta yüksek tutarak daha fazla keşif yapılması sağlanabilir. Ek olarak, **reward shaping** yöntemi ile ödül fonksiyonunu daha hassas hale getirip robotu daha akıllı davranışa teşvik edebilirim.

**Problem 7:** Engelden kaçınan robotun her seferinde aynı rotayı takip ettiğini fark ediyorsunuz. Bunu nasıl düzeltebilirsiniz?

- **Yanıt Önerisi:** Bu durum, robotun sadece belirli bir rotayı öğrenip diğer rotaları keşfetmediğini gösterir. Bu problemi çözmek için **exploration-exploitation** dengesini yeniden değerlendirmek gereklidir. **Randomized actions** ekleyerek robotu farklı rotaları keşfetmeye teşvik edebilirim. Aynı zamanda ödül fonksiyonunu çeşitlendirerek her rotada farklı ödüller sağlayabilir ve robotun daha geniş bir strateji yelpazesini geliştirmesini sağlayabilirim.

**Problem 8:** Reinforcement learning ile bir robotun hedefe ulaşmasını sağlamaya çalışıyorsunuz, ancak her adımda verilen küçük ödüller nedeniyle robot hedefe daha yavaş ulaşıyor. Bunu nasıl çözersiniz?

- **Yanıt Önerisi:** Bu durumda ödül fonksiyonunu yeniden tanımlayarak robotun daha kısa sürede hedefe ulaşmasını sağlayabilirim. Örneğin, hedefe ulaştığında büyük bir ödül vererek robotu teşvik edebilirim. Aynı zamanda, ödül fonksiyonunda adım sayısına dayalı cezalar ekleyerek robotun daha kısa sürede hareket etmesini sağlayabilirim.

**Problem 9:** Bir reinforcement learning modelinde ajan sürekli çevreyi keşfetmek yerine mevcut stratejiye sadık kaliyor. Bunu nasıl çözebilirsiniz?

- **Yanıt Önerisi:** Ajanın sürekli aynı stratejiyi kullanması, **exploration-exploitation** dengesinin bozulduğunu gösterir. Bu durumu düzeltmek için **epsilon-greedy** stratejisini uygulayabilir ve epsilon değerini dinamik olarak değiştirerek ajanı daha fazla keşfe yönlendirebilirim. Ek olarak, ajan için bir **decaying epsilon** stratejisi uygulanabilir, böylece ajan başlangıçta keşif yapar, zamanla daha fazla sömürüğe geçer.

**Problem 10:** Birden fazla robotun aynı ortamda aynı anda öğrenmesi gerekiyor. Bu durumda nasıl bir reinforcement learning stratejisi izlersiniz?

- **Yanıt Önerisi:** Bu durumda **Multi-Agent Reinforcement Learning (MARL)** kullanarak, robotlar arasında koordinasyonu sağlamak için bir strateji oluşturabilirim. Robotların birbirleriyle etkileşimde bulunmalarını sağlamak ve birbirlerinin hareketlerini öğrenmelerini sağlamak için her ajanın ödül fonksiyonunu optimize edebilirim. Ayrıca, ajanlar arasında bilgi paylaşımı yaparak işbirliği sağlayabilirim.

---

### 3. Genel Yapay Zeka Problemleri

**Problem 11:** Yapay zeka modeliniz çok yüksek bir doğrulukla çalışıyor ancak tahmin sonuçları gerçeğe uymuyor. Bunun sebebi ne olabilir ve nasıl çözüm öneririsiniz?

- **Yanıt Önerisi:** Modelin doğruluk oranının yüksek olup tahminlerin gerçeğe uymaması overfitting işaretidir. Bu durumda, modelin aşırı öğrenim yaptığını ve eğitim verisine çok fazla uyum sağladığını gösterebilir. Bu problemi çözmek için:
  - **Regularization (L2/L1)** teknikleri kullanılabilir.
  - **Cross-validation** uygulanabilir.
  - Eğitim verisi artırılarak modelin genelleme yeteneği iyileştirilebilir.
  - **Early stopping** kullanılarak modelin aşırı eğitim yapması önlenebilir.

**Problem 12:** Bir veri setindeki sınıfların dengesiz olduğunu fark ettiniz. Bu durumda hangi yöntemleri kullanabilirsiniz?

- **Yanıt Önerisi:** Sınıf dengesizliği, makine öğrenmesi modellerinde doğruluk ve genelleme sorunlarına yol açabilir. Bu durumu düzeltmek için:
  - **Data Augmentation** uygulanabilir (eksik sınıfın verilerini artırmak).
  - **Class Weighting** tekniği ile modelin azınlık sınıflarına daha fazla önem vermesi sağlanabilir.
  - **Over-sampling (SMOTE)** veya **under-sampling** gibi teknikler kullanılabilir.

## 1. Genel Yapay Zeka Problemleri ve Sorular

**Problem 1:** Veri setinizde feature (özellik) sayısı çok fazla ve modeliniz düşük performans gösteriyor. Bu durumda ne yaparsınız?

- **Yanıt Önerisi:** Bu, **high-dimensional data** (yüksek boyutlu veri) problemi olarak bilinir ve **curse of dimensionality** (boyutsallık laneti) modelin genelleme yeteneğini olumsuz etkileyebilir. Bu problemi çözmek için:
  - **Feature selection:** Model için en anlamlı özellikleri seçmek amacıyla, **Backward Elimination, Forward Selection** gibi yöntemler kullanılabilir.
  - **Dimensionality Reduction:** Özelliklerin boyutunu düşürmek için **PCA (Principal Component Analysis)**, **t-SNE** veya **LDA (Linear Discriminant Analysis)** gibi yöntemler uygulanabilir.
  - **Regularization:** Aşırı öğrenmeyi önlemek ve gereksiz özellikleri cezalandırmak için **L1/L2 regularization** kullanılabilir.

**Problem 2:** Bir sınıflandırma modeli geliştirdiniz, ancak modelinizin Precision'ı yüksek, Recall'ü ise düşük. Bu durumu nasıl iyileştirirsiniz?

- **Yanıt Önerisi:** Bu durum, modelin azınlık sınıflarını tanımda başarısız olduğunu gösterir. Bu sorunu çözmek için:
  - **Class Weighting:** Azınlık sınıflarına daha fazla ağırlık vererek modelin bu sınıfları daha fazla dikkate almasını sağlayabilirim.
  - **Precision-Recall Tradeoff: Threshold** (eşik) ayarlarını değiştirerek, Precision ve Recall arasında daha iyi bir denge sağlayabilirim.
  - **Data Augmentation:** Azınlık sınıfının örneklerini artırmak (örneğin, **SMOTE** yöntemi ile).
  - **Over-sampling veya under-sampling:** Azınlık sınıfını çoğaltmak veya çoğunluk sınıfını azaltmak.

**Problem 3:** Eğitim verinizde çok fazla gürültü var ve modeliniz düşük doğrulukla çalışıyor. Gürültüyü azaltmak için ne yaparsınız?

- **Yanıt Önerisi:** Gürültü, verideki hatalı, eksik veya yanlış etiketlenmiş örnekler olabilir. Gürültüyü azaltmak için şu yöntemler kullanılabilir:
  - **Outlier Detection:** Aykırı değerleri tespit etmek ve gerekirse onları kaldırın.
  - **Veri Temizleme:** Eksik ve hatalı verileri düzeltmek veya uygun değerlerle doldurmak (imputation).
  - **Ensemble Modeller:** Gürültüden etkilenmeyen modeller oluşturmak için birden fazla modelin birleştirildiği **ensemble** yöntemleri kullanılabilir (örneğin, **Bagging, Boosting**).
  - **Regularization:** Gürültüye karşı dayanıklılığı artırmak için **L2 regularization** gibi yöntemler kullanılabilir.

**Problem 4:** Modeliniz validation setinde iyi sonuç veriyor, ancak gerçek dünyadaki verilerle test edildiğinde performansı düşüyor. Bunu nasıl düzeltirsiniz?

- **Yanıt Önerisi:** Bu, modelin genelleme yeteneğinin zayıf olduğunu, yani modelin overfitting yaptığını gösterir. Çözüm için:
  - **Daha fazla gerçek dünya verisi ile model eğitmek.**
  - **Cross-validation** kullanarak modelin daha sağlam bir şekilde eğitilmesini sağlamak.

- **Regularization** yöntemlerini (L1, L2) uygulayarak modelin aşırı öğrenmesini önlemek.
- Eğitim verisini artırmak veya modelin karmaşıklığını azaltmak (daha az katmanlı veya daha az parametreli bir model).

**Problem 5:** Modelin eğitim süresi çok uzun ve yüksek hesaplama gücü gerektiriyor. Eğitimi hızlandırmak için ne yaparsınız?

- **Yanıt Önerisi:**
  - **Batch size'ı artırmak:** Daha büyük batch'lerle eğitim yaparak hesaplama sürelerini azaltabilirim.
  - **Distributed Computing:** Eğitim sürecini birden fazla GPU veya dağıtık sunucular üzerinde gerçekleştirmek.
  - **Transfer Learning:** Önceden eğitilmiş bir modeli kullanarak eğitim sürecini hızlandırmak ve sadece fine-tuning yapmak.
  - **Modelin karmaşıklığını azaltmak:** Daha basit bir model mimarisi kullanarak hesaplama maliyetlerini düşürebilirim.
  - **Early Stopping** kullanarak gereksiz uzun eğitim süreçlerinden kaçınmak.

**Problem 6:** Sınıf sayısı çok fazla ve her sınıf için yeterli veri yok. Model performansını nasıl iyileştirirsiniz?

- **Yanıt Önerisi:**
  - **Data Augmentation:** Yetersiz veri olan sınıflar için veri artırma teknikleri (rotation, flipping, cropping) uygulayarak daha fazla veri üretmek.
  - **Transfer Learning:** Başka bir geniş veri seti üzerinde önceden eğitilmiş bir modelin ağırlıklarını kullanarak sınırlı veri ile daha iyi sonuçlar elde edilebilir.
  - **One-vs-Rest veya One-vs-One Sınıflandırma:** Çok sınıfı problemleri daha iyi çözmek için birden fazla ikili sınıflandırma modeli kullanmak.
  - **Few-shot learning:** Az veri ile öğrenme yöntemlerini uygulamak.

**Problem 7:** Modeliniz veriyi doğru tahmin ediyor ancak modelin nasıl sonuçlar ürettiğini açıklamakta zorlanıyorsunuz. Modelin açıklanabilirliğini nasıl artırırsınız?

- **Yanıt Önerisi:**
  - **LIME (Local Interpretable Model-agnostic Explanations):** Modelin kararlarını açıklamak için kullanabileceğim bir yöntemdir. Özellikle kara kutu modellerin (derin öğrenme gibi) sonuçlarının daha açıklanabilir hale getirilmesi sağlanabilir.
  - **SHAP (SHapley Additive exPlanations):** Modelin her bir kararına hangi özelliklerin ne kadar katkıda bulunduğu analiz edebilir.
  - Daha basit ve açıklanabilir modeller tercih etmek: Karar ağaçları, lojistik regresyon gibi modeller daha açıklanabilir sonuçlar verir.
  - **Feature Importance** ve **Partial Dependence Plots** gibi tekniklerle, modelin hangi özelliklere ağırlık verdiğiğini görselleştirebilirim.

**Problem 8:** Modelinizdeki overfitting'i nasıl tespit eder ve önlersiniz?

- **Yanıt Önerisi:**
  - **Validation ve training loss grafikleri:** Eğitim kaybı azalırken doğrulama kaybı artıyorsa overfitting olduğunu anlayabilirsiniz.

- **Cross-validation** kullanarak modelin genellemeyi yeteneğini değerlendirebilirim.
- **Regularization** (L1, L2) kullanarak modelin aşırı karmaşık hale gelmesini önleyebilirim.
- **Early stopping** ile modelin aşırı eğitim yapmasını engellebilirim.
- **Dropout** gibi teknikler ekleyerek sinir ağlarında aşırı öğrenmeyi önleyebilirim.

**Problem 9:** Eğitim verinizde çok sayıda etiketlenmemiş veri var ve bunları kullanarak bir model geliştirmek istiyorsunuz. Hangi yöntemleri kullanırsınız?

- **Yanıt Önerisi:**
  - **Semi-supervised Learning:** Bir miktar etiketlenmiş veri ve çok sayıda etiketlenmemiş veriyi bir arada kullanarak eğitim yapılabılır.
  - **Self-training:** İlk başta küçük bir etiketlenmiş veri seti ile bir model eğitilir, sonra modelin en yüksek güvenle tahmin ettiği etiketlenmemiş veriler modele geri beslenerek eğitim genişletilir.
  - **Generative Adversarial Networks (GANs):** Veri üretmek için GAN'ler kullanılabilir.
  - **Transfer Learning:** Etiketli veri eksikliğini telafi etmek için önceden eğitilmiş modellerin ağırlıkları kullanılabilir.

**Problem 10:** Makine öğrenmesi modeliniz için en iyi hiperparametreleri nasıl belirlersiniz?

- **Yanıt Önerisi:**
  - **Grid Search:** Hiperparametrelerin farklı kombinasyonlarını denemek ve en iyi performansı sağlayan ayarları bulmak için kullanabilirim.
  - **Random Search:** Daha geniş bir hiperparametre aralığında rastgele arama yaparak iyi bir kombinasyon bulabilirim.
  - **Bayesian Optimization:** Hiperparametre arama sürecini hızlandırmak için Bayes optimizasyonu kullanarak daha etkili bir şekilde hiperparametreleri seçebilirim.
  - **Cross-validation** kullanarak her bir hiperparametre kombinasyonunun performansını ölçebilirim.

## 2. Önerilebilecek Daha Fazla Yapay Zeka Problemi

1. **Bir modelin performansını değerlendirmek için hangi metrikleri kullanırsınız ve hangi durumlarda hangi metriği tercih edersiniz?**
  - Yanıt: Sınıflandırma problemleri için **accuracy**, **precision**, **recall**, **F1-score**, **AUC-ROC** kullanılır. Regresyon problemleri için ise **MSE (Mean Squared Error)**, **MAE (Mean Absolute Error)**, **R-squared** gibi metrikler tercih edilir. Her bir metriğin kullanım alanları açıklanabilir.
2. **Elinizde büyük boyutlu bir veri seti var ve bellekte işlem yapmakta zorluyorsunuz. Ne yaparsınız?**
  - Yanıt: Veriyi parçalar halinde işlemek için **mini-batch processing** kullanabilirim. Ayrıca, **distributed computing** ve **cloud-based computation** çözümleri de tercih edilebilir. Veriyi bellekten bağımsız işlemek için **generators** kullanılabilir.
3. **Unbalanced veri setinizde, çok fazla pozitif sınıf yerine negatif sınıf bulunuyor. Sınıf dengesini sağlamak için ne yaparsınız?**

- Yanıt: **Over-sampling (SMOTE), under-sampling, class weighting** gibi yöntemlerle sınıf dengesini sağlayarak modeli daha iyi hale getirebilirim.,

## 1. Senaryo: Bir Ev Fiyatını Tahmin Etmeniz Gerekiyor

- **Soru:** Evin büyülüüğü, konumu, oda sayısı gibi özelliklere dayalı olarak bir evin satış fiyatını tahmin etmeniz gerekiyor. Bu problemi çözmek için hangi makine öğrenmesi yöntemini kullanırsınız: regresyon mu sınıflandırma mı?
- **Yanıt:** Bu problem **regresyon** problemidir. Çünkü tahmin edilmesi gereken sonuç (ev fiyatı) sürekli bir değerdir. Ev fiyatı, belirli bir sınıfa ya da kategoriye girmediği için sınıflandırma kullanılamaz. **Doğrusal regresyon (linear regression)** veya **Ridge/Lasso regression** gibi algoritmalar kullanılabilir.

## 2. Senaryo: Bir E-postanın Spam Olup olmadığını Belirleme

- **Soru:** Bir e-postanın spam olup olmadığını belirlemeniz gerekiyor. Bu problemi çözmek için hangi yaklaşımı kullanırsınız?
- **Yanıt:** Bu problem **sınıflandırma** problemidir. Çünkü sonuç iki kategoriden birine aittir: **spam** veya **spam değil**. **Lojistik regresyon**, **Naive Bayes** veya **Destek Vektör Makineleri (SVM)** gibi sınıflandırma algoritmaları kullanılabilir.

## 3. Senaryo: Hastaların Kanser Riskini Tahmin Etme

- **Soru:** Bir grup hastanın sağlık verilerine dayanarak, bir hastanın kanser olup olmadığını tahmin etmek istiyorsunuz. Bu problemi çözmek için hangi yöntemi kullanırsınız?
- **Yanıt:** Bu bir **sınıflandırma** problemidir. Çünkü model, hastayı **kanser** veya **kanser değil** olarak sınıflandırır. **Lojistik regresyon**, **Destek Vektör Makineleri (SVM)**, **Random Forest** veya **Gradient Boosting** gibi sınıflandırma algoritmaları uygundur.

## 4. Senaryo: Bir Otomobilin Yakıt Tüketimini Tahmin Etme

- **Soru:** Bir otomobilin motor gücü, ağırlığı ve aerodinamik yapısı gibi özelliklere dayanarak yakıt tüketimini tahmin etmeniz gerekiyor. Bu problemi çözmek için hangi yaklaşımı kullanırsınız?
- **Yanıt:** Bu problem bir **regresyon** problemidir. Çünkü tahmin edilmesi gereken sonuç (yakıt tüketimi), sürekli bir değerdir. **Doğrusal regresyon (linear regression)**, **polynomial regresyon** veya **random forest regression** gibi regresyon algoritmaları kullanılabilir.

## 5. Senaryo: Banka Müşterilerinin Kredi Kartı Alıp Almayacağını Tahmin Etme

- **Soru:** Banka müşterilerinin kredi kartı başvurusu yapıp yapmayacağı tahmin etmeniz isteniyor. Hangi yöntemi kullanırsınız?
- **Yanıt:** Bu bir **sınıflandırma** problemidir. Çünkü müşterinin kredi kartı alıp almayacağı **evet** veya **hayır** şeklinde sınıflandırırsınız. **Lojistik regresyon**, **Random Forest** veya **XGBoost** gibi sınıflandırma algoritmaları uygundur.

## 6. Senaryo: Bir Ürünün Fiyatına Göre Satış Miktarını Tahmin Etme

- **Soru:** Bir ürünün fiyatına bağlı olarak o üründen kaç adet satılacağını tahmin etmek istiyorsunuz. Bu durumda hangi yaklaşımı kullanırsınız?
- **Yanıt:** Bu problem bir **regresyon** problemidir. Çünkü tahmin edilmesi gereken sonuç (satış miktarı) sürekli bir değerdir. **Doğrusal regresyon (linear regression)**, **Ridge/Lasso regression** gibi algoritmalar uygundur.

## 7. Senaryo: Bir Hastanın Hayatta Kalma Süresini Tahmin Etme

- **Soru:** Kanser hastalarının tedavi sonrasında hayatta kalma sürelerini tahmin etmeniz gerekiyor. Hangi yaklaşımı kullanırsınız?
- **Yanıt:** Bu bir **regresyon** problemidir. Çünkü tahmin edilmesi gereken sonuç (hayatta kalma süresi) sürekli bir değerdir. **Doğrusal regresyon** veya **survival analysis** gibi teknikler kullanılabilir.

## 8. Senaryo: Bir Görüntüdeki Nesneleri Tanımlama

- **Soru:** Bir görüntüdeki farklı nesneleri (örneğin, kedi, köpek, araba gibi) tanımlamanız isteniyor. Bu problemi çözmek için hangi yöntemi kullanırsınız?
- **Yanıt:** Bu problem bir **sınıflandırma** problemidir. Çünkü model, her nesneyi belirli bir sınıfı (kedi, köpek, araba) sınıflandırmalıdır. Bu tür problemler için **Convolutional Neural Networks (CNN)** gibi derin öğrenme sınıflandırma modelleri kullanılabilir.

## 9. Senaryo: Bir Kullanıcının Bir Sonraki Adımda Tıklayacağı Ürünü Tahmin Etme

- **Soru:** Bir kullanıcının e-ticaret sitesinde bir sonraki adımda hangi ürüne tıklayacağını tahmin etmeniz gerekiyor. Hangi yöntemi kullanırsınız?
- **Yanıt:** Bu bir **sınıflandırma** problemidir. Çünkü model, kullanıcının belirli bir ürünü seçip seçmeyeceğini veya birden fazla ürün arasında seçim yapacağını tahmin eder. **Lojistik regresyon**, **Random Forest**, **Recurrent Neural Networks (RNN)** gibi algoritmalar kullanılabilir.

## 10. Senaryo: Bir Elektrik Motorunun Hangi Durumda Arızalanacağını Tahmin Etme

- **Soru:** Bir elektrik motorunun çeşitli sensör verilerine dayanarak arızalanma olasılığını tahmin etmeniz isteniyor. Hangi yöntemi kullanırsınız?
- **Yanıt:** Bu problem bir **sınıflandırma** problemidir. Çünkü motorun ya **arızalanacak** ya da **arızalanmayacak** şeklinde sınıflandırılması gereklidir. **Lojistik regresyon**, **Random Forest**, **XGBoost** gibi algoritmalar uygundur.

## 11. Senaryo: Bir Geliştiricinin Yazdığı Kodun Hatalı Olup Olmadığını Tahmin Etme

- **Soru:** Yazılım geliştirme süreçlerinde, bir geliştiricinin yazdığı kodun hatalı olup olmadığını tahmin etmeniz gerekiyor. Hangi yöntemi kullanırsınız?
- **Yanıt:** Bu bir **sınıflandırma** problemidir. Çünkü kodun **hatalı** ya da **hataya sahip değil** olarak sınıflandırılması gereklidir. **Lojistik regresyon**, **Random Forest** veya **Destek Vektör Makineleri (SVM)** gibi sınıflandırma algoritmaları uygundur.

## 12. Senaryo: Bir Aracın Hangi Renk Olacağını Tahmin Etme

- **Soru:** Verilen özelliklere (model, üretim yılı, satış bölgesi) dayanarak bir aracın hangi renkte olacağını tahmin etmeniz gerekiyor. Bu problemi çözmek için hangi yöntemi kullanırsınız?
- **Yanıt:** Bu problem bir **sınıflandırma** problemidir. Çünkü sonuç belirli kategorilerden (renkler) birine aittir. **Random Forest**, **Decision Trees** veya **k-en yakın komşu (k-NN)** gibi sınıflandırma algoritmaları kullanılabilir.

## 2. Veri Ön İşleme ile İlgili Sorular

**Soru 3:** Veri setinizde eksik veriler var. Bu durumu nasıl işlersiniz?

- **Yanıt Önerisi:**
  - Eksik verileri **ortalama (mean)**, **medyan** veya **mod** ile doldurabilirsiniz.
  - Eksik verileri **silmek** bir seçenek olabilir ancak veri kaybına neden olabilir.
  - Daha karmaşık durumlar için **K-NN imputation** veya **model tabanlı tahmin** gibi yöntemler kullanılabilir.

**Soru 4:** Veri setinizde aykırı değerler (outliers) bulunuyor. Bu değerlerle nasıl başa çıksınız?

- **Yanıt Önerisi:**
  - Aykırı değerler tespit edildikten sonra bu veriler ya **silinir** ya da doğru bir şekilde dönüştürülür.
  - **Z-score** veya **IQR (Interquartile Range)** gibi yöntemlerle aykırı değerleri tespit edebilirsiniz.
  - Aykırı değerleri veri setine zarar vermemek için **dönüştürme (transformation)** veya **winsorization** teknikleri kullanılabilir.

**Soru 5:** Veri setinizde kategorik veriler bulunuyor. Bu verileri makine öğrenmesi modellerine nasıl hazır hale getirirsınız?

- **Yanıt Önerisi:**
  - **Label encoding** veya **one-hot encoding** kullanarak kategorik verileri sayısal formata dönüştürebilirim.
  - Eğer kategorik değişken çok sayıda sınıfı sahipse, **frequency encoding** veya **target encoding** gibi yöntemler kullanılabilir.

## 5. Deep Learning ve Neural Networks ile İlgili Sorular

**Soru 11:** Derin öğrenme modelleri neden büyük veri gerektirir?

- **Yanıt Önerisi:** Derin öğrenme modelleri, genellikle çok sayıda parametre içerdiği için büyük miktarda veri ile daha iyi performans gösterir. Yeterli veri olmadan model overfitting yapabilir. Ayrıca, daha fazla veri ile sinir ağları daha doğru tahminler yapabilir ve daha iyi genelleme yeteneği kazanır.

**Soru 12:** CNN (Convolutional Neural Networks) nedir ve ne zaman kullanılır?

- **Yanıt Önerisi:**

- **CNN'ler** özellikle görüntü işleme ve video analizi gibi alanlarda kullanılır. CNN'ler, görüntülerdeki yerel özellikleri tanıabilen ve bu özellikleri öğrenebilen filtreler (convolutional kernels) kullanır.
- **Convolutional katmanlar**, evrişim işlemleri yaparak görüntüdeki kenarlar, dokular gibi özellikleri çıkarır. **Pooling katmanları**, görüntü boyutlarını küçülterek hesaplama maliyetini azaltır ve **fully connected** katmanlar, bu özelliklerle sınıflandırma işlemi yapar.

**Soru 13:** Transfer learning nedir ve ne zaman kullanılır?

- **Yanıt Önerisi:** **Transfer learning**, önceden eğitilmiş bir modelin başka bir problem üzerinde yeniden kullanılmasıdır. Özellikle küçük veri setleri veya eğitim süresinin kısaltılması gereken durumlarda kullanılır. Örneğin, ImageNet gibi büyük bir veri setinde eğitilmiş bir modelin ağırlıkları başka bir görüntü sınıflandırma problemine uygulanabilir.

## 1. Hyperparameter Tuning (Hiperparametre Optimizasyonu) Nedir?

**Hiperparametre tuning**, bir makine öğrenmesi modelinin performansını artırmak için **hiperparametrelerin** optimize edilmesi sürecidir. Hiperparametrelər, modelin eğitimi sırasında belirlenen ve modelin yapısını veya nasıl öğrenileceğini tanımlayan parametrelərdir. Bu parametrelər modelin öğrenme süreci boyunca değişmez ve kullanıcı tarafından belirlenir. Örneğin, **learning rate (öğrenme oranı)**, **batch size**, **epoch sayısı**, **dallanma derinliği** gibi değerler hiperparametrelərdir.

**Hiperparametre Optimizasyonu Yöntemleri:**

1. **Grid Search:** Hiperparametrelərin belirli bir aralıktı, önceden belirlənmiş bir kılavuz boyunca denenmesidir. Bu yöntem genellikle kapsamlı ancak zaman açısından maliyetlidir, çünkü her kombinasiyon tek tek test edilir.
2. **Random Search:** Grid Search'e kıyasla daha etkili bir yöntemdir. Hiperparametrelərin belirlənən aralıklar içinde rastgele değerlərə ayarlanmasıyla test yapılır. Genellikle daha geniş bir aralıktı hızlı sonuçlar almak için kullanılır.
3. **Bayesian Optimization:** Bu yöntem, her denemenin sonuçlarına dayanarak sonraki testlərin en iyi hiperparametre ayarlarını tahmin eder ve daha az deneme ile daha doğru sonuçlara ulaşmaya çalışır.

## 2. Transfer Learning Nedir?

**Transfer learning**, önceden eğitilmiş bir modelin ağırlıklarının kullanıldığı ve bu ağırlıkların başka bir probleme veya veri setine uygulanarak yeni bir model oluşturulmasını sağlayan bir öğrenme yöntemidir. Özellikle küçük veri setleri veya hesaplama kaynakları kısıtlı olduğunda kullanılır. Bu yöntem, derin öğrenme modellerinin eğitimini hızlandırır ve daha az veri ile daha iyi sonuçlar elde etmeyi sağlar.

**Transfer Learning Nasıl Kullanılır?**

- **Önceden Eğitilmiş Modeller:** Transfer learning'de yaygın olarak kullanılan modeller, büyük veri setlerinde (örneğin, ImageNet) eğitilmiş **VGG**, **ResNet**, **Inception** gibi derin sinir ağlarıdır.

- **Yeni Bir Görev için Uygulama:** Örneğin, önceden eğitilmiş bir modelin ilk katmanları görüntüdeki temel özelliklerini (kenar, doku) öğrenmişse, bu ağırlıkları başka bir görüntü sınıflandırma problemini çözmek için kullanabilirsiniz.
- Transfer learning, özellikle **görüntü işleme** ve **doğal dil işleme** gibi alanlarda sıkça kullanılır.

### 3. YOLO Transfer Learning'e Örnek midir?

Evet, **YOLO (You Only Look Once)**, transfer learning'e uygun bir yapıdır ve bu teknik, YOLO gibi derin öğrenme modellerinde sıkılıkla kullanılır. YOLO gibi büyük ve karmaşık modellerin eğitimi zaman alıcı ve veri seti açısından maliyetli olabilir. Bu yüzden, önceden eğitilmiş YOLO modelleri başka veri setleri üzerinde **fine-tuning** (ince ayar) yapılarak farklı nesne tespit görevlerine uyarlanabilir.

**YOLO için Transfer Learning:**

- Örneğin, **YOLOv4** veya **YOLOv5** modelleri, önceden büyük veri setleri üzerinde eğitilmiş ağırlıklarla başlatılabilir. Bu ağırlıklar temel özellikleri öğrenmiştir, dolayısıyla modelin yalnızca son birkaç katmanını (output layer) yeniden eğitmek yeterli olabilir.
- Bu, **nesne tespiti** gibi problemler için transfer learning kullanmanın yaygın bir örneğidir. Modelin tüm katmanlarını sıfırdan eğitmek yerine, sadece yeni veri setine özgü son katmanlar (output layer) yeniden eğitilir.

### 4. Fine-Tuning (İnce Ayar) Nedir?

**Fine-tuning**, transfer learning sürecinde, önceden eğitilmiş bir modelin bazı katmanlarını yeniden eğiterek belirli bir görevde uyarlamaktır. Transfer learning ile alınan önceden eğitilmiş modelin çoğu katmanı, eğitildiği veri seti üzerindeki temel özellikleri öğrenmiştir. Ancak, yeni veri seti üzerinde daha iyi sonuç almak için modelin son birkaç katmanı veya tamamı yeniden eğitilebilir.

**Fine-Tuning Süreci:**

1. **Önceden Eğitilmiş Modelin Kullanılması:** Önceki adımda eğitilmiş bir modelin ağırlıkları kullanılır.
2. **Son Katmanların Yeniden Eğitilmesi:** Coğu durumda, modelin son katmanları yeni bir görev için adapte edilir ve eğitimde bu katmanlar yeniden eğitilir. Örneğin, bir nesne tespiti modelinde sadece son sınıflandırma katmanı yeniden eğitilebilir.
3. **Learning Rate'i Düşürme:** Fine-tuning sırasında genellikle daha küçük bir öğrenme oranı (**learning rate**) kullanılır. Bu, daha önceden öğrenilmiş özelliklerini korurken, yeni özelliklerin eklenmesine olanak tanır.

Mini-batch boyutlarının genellikle **2'nin kuvvetleri** (örneğin, 12 değil ama 32, 64, 128 gibi) olarak seçilmesinin birkaç önemli teknik sebebi vardır. Bu seçim, özellikle donanım ve hesaplama performansı açısından avantajlar sağlar. İşte bunun nedenleri:

#### 1. Donanım Optimizasyonu (GPU ve CPU Performansı)

- **GPU Bellek Optimizasyonu:** GPU'lar, verileri paralel işleme üzerine optimize edilmiştir ve bellek yapıları genellikle 2'nin kuvvetleri şeklinde çalışır. Mini-batch

boyutunu 2'nin kuvvetleri olarak seçmek, GPU bellek erişiminde ve işlemlerde verimlilik sağlar. Bu tür boyutlar, GPU'nun işlemci ve bellek yapılarına daha uygun şekilde eşlenir, bu da hız ve verimlilik artışı sağlar.

- **Bellek Yığınları (Memory Blocks):** İşlemci ve GPU'lar, bellek işlemlerinde 2'nin kuvvetleri şeklinde yığınlarla çalışır. Eğer mini-batch boyutu 2'nin kuvvetleri olarak seçilmezse, işlemci veya GPU daha küçük verilerle çalışırken bellek erişimi sırasında ek işlemler yaparak performansı düşürebilir.

## 2. Veri Paralelliği ve Hesaplama Verimliliği

- **Veri paralelliği:** Derin öğrenme algoritmalarında mini-batch boyutu genellikle paralel hesaplama için optimize edilir. 2'nin kuvvetleri şeklinde seçilen boyutlar, işlemcinin (CPU) ve GPU'nun matris çarpması gibi işlemleri hızlı bir şekilde yapabilmesi için daha uygun boyutlar sağlar.
- **Daha hızlı hesaplamalar:** Mini-batch boyutları 2'nin kuvvetleri şeklinde olduğunda, matris işlemleri gibi hesaplamalar daha hızlı gerçekleşir. Özellikle sinir ağı eğitimi sırasında, büyük miktarda veri üzerinde yapılacak olan bu paralel işlemler GPU performansını doğrudan etkiler.

## 3. Hafıza ve Hesaplama Dengesi

- **Hafıza kullanımının dengelenmesi:** Mini-batch boyutları çok küçükse (örneğin, 2 veya 4 gibi), modelin hesaplama verimliliği düşer. Çok büyük mini-batch boyutları ise (örneğin, 1024 gibi), GPU belleğini zorlayabilir ve daha fazla bellek kullanımı gerektirir. Bu nedenle, genellikle 32, 64, 128 gibi **2'nin kuvveti** boyutlar, hem bellek kullanımı hem de hesaplama verimliliği açısından dengeli bir çözüm sunar.

## 4. Standartlaşmış Seçim

- **Pratikte yaygın kullanımı:** Mini-batch boyutları, 2'nin kuvvetleri olarak standartlaştırıldığı için, derin öğrenme kütüphanelerinin çoğu (örneğin, TensorFlow, PyTorch gibi) bu boyutlarla daha iyi çalışacak şekilde optimize edilmiştir. Dolayısıyla, bu kütüphanelerin algoritmalarında varsayılan olarak 2'nin kuvveti boyutlarının kullanılması daha yaygın ve performanslıdır.

## 5. Matematiksel ve Hesaplama Kolaylığı

- **Matematiksel olarak kolay işlenebilirlik:** 2'nin kuvvetleri genellikle matematiksel işlemlerde daha hızlı ve kolay işlenir. Hem hesaplamlar hem de bellek yönetimi için 2'nin kuvvetleri, daha düşük maliyetli ve hızlı bir işlem sağlar.
- **Matris ve tensör işlemleri:** Derin öğrenme modellerinde kullanılan matris ve tensör işlemleri, boyutlar 2'nin kuvveti olduğunda daha verimli gerçekleştirilir.

## Neden 12 Gibi Değerler Değil de 64, 128 Gibi Değerler?

- **12 gibi bir batch boyutu** seçmek, matematiksel ve donanımsal anlamda işlemler için verimli değildir. 12, 2'nin kuvveti olmadığı için GPU ve CPU'nun paralel işlemlerine uyum sağlayamaz, bu da gereksiz bellek ve zaman kaybına yol açar.

# KODUM

## 1. Q-learning Nedir?

**Q-learning**, bir **Reinforcement Learning** (pekiştirmeli öğrenme) algoritmasıdır ve modelin çevreye etkileşime geçip aldığı geri bildirimler (ödüller) sayesinde bir **policy (politika)** öğrenmesini sağlar. Q-learning, **state-action (durum-eylem) çiftlerine** dayalı bir tablo (Q-table) oluşturarak hangi aksiyonların hangi durumlardan daha iyi olduğunu öğrenmeye çalışır.

**Anlatabileceğin Anahtar Noktalar:**

- **State (Durum):** Robotun bulunduğu x ve y koordinatları, bu koordinatlar Q-table'da bir duruma (state) karşılık gelir. Kodda, bu koordinatlar diskretize edilerek belirli aralıklara bölünmüş (x\_bins ve y\_bins) bir şekilde **state** olarak kullanılıyor.

**Action (Eylem):** Q-learning'de robotun yapabileceği eylemler (ileri gitme, geri gitme, sola veya sağa dönme gibi) belirlenir. Burada robotun eylemleri **3 olasılıkla** (ileri, geri, sola) sınırlıdır. Kodda bu eylemler **choose\_action** fonksiyonunda belirlenmiş.

**Q-table:** Bu tablo, her durum-eylem çifti için bir değer saklar ve bu değerler **q-learning** güncelleme kuralıyla güncellenir:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[ r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right]$$

Burada:

- $\alpha$  öğrenme oranıdır (kodda `learning_rate=0.01` ).
- $\gamma$  gelecekteki ödüllerin ne kadar önemli olduğunu belirleyen indirim faktörüdür (`reward_decay=0.9` ).
- $r$  ise alınan ödülüdür.

## 2. Policy (Politika Seçimi) ve Epsilon-Greedy Stratejisi

**Politika (Policy)**, bir ajanın (robotun) hangi durumlardan hangi eylemleri seçtiğini belirleyen stratejidir. Bu kodda kullanılan politika, **Epsilon-Greedy** stratejisine dayanmaktadır. Bu strateji, robotun belirli bir oranda (epsilon değeriyle) rastgele aksiyonlar almasını ve geri kalan zamanda en iyi aksiyonu seçmesini sağlar. Böylece **exploration** (keşif) ve **exploitation** (kazanım) arasında bir denge kurulur.

- **Exploration (Keşif):** Robot, yeni durumları keşfetmek için rastgele aksiyonlar seçer (epsilon %10 rastgele aksiyon seçiliyor).
- **Exploitation (Kazanım):** Robot, Q-tablosuna dayanarak en yüksek Q-değerine sahip aksiyonu seçer.

Mülakatta bunu şöyle anlatabilirsin:

- "Epsilon-Greedy stratejisi ile robotum, %90 oranında en iyi bilinen aksiyonu, %10 oranında ise keşif amaçlı rastgele aksiyonları seçiyor. Bu sayede robot, çevresini tanıma fırsatı bulurken en iyi stratejiyi öğrenmeye çalışıyor."

### 3. Reward (Ödül) Belirleme

Ödül, robotun çevre ile etkileşimi sonucunda aldığı geri bildirimdir. **Q-learning**'de ödül, bir aksiyonun ne kadar başarılı olduğunu ölçmek için kullanılır. Bu kodda ödül belirleme şu şekildedir:

- **Hedefe Yaklaşma Ödülü:** Robot hedefe ne kadar yakınsa, o kadar yüksek ödül alır. Hedefe olan mesafe azaldıkça **goal\_reward** artar.
- **Engelden Kaçınma Cezası:** Robot bir engelle yakın olduğunda **obstacle\_penalty** alır. Bu engelden kaçınma başarısızlığını ifade eder.

Bu ödül sistemini şöyle açıklayabilirsin:

- "Ödül fonksiyonum, robotun hedefe olan mesafesiyle ters orantılı bir ödül veriyor. Robot engellere yaklaşlığında ise bir ceza uygulanıyor. Böylece robot, hem hedefe yaklaşmayı hem de engellerden kaçınmayı öğreniyor."

### 4. Markov Karar Süreci (Markov Decision Process - MDP)

Bu senaryoda robotun engellerden kaçınma ve hedefe ulaşma problemi bir **Markov Karar Süreci** (MDP) olarak modellenebilir. **MDP**, bir ajanın durumlardan aksiyonlar seçerek ödüller aldığı ve gelecekteki aksiyonları planladığı bir çerçevedir.

Bir **MDP** şu dört bileşeni içerir:

- **Durumlar (States):** Robotun bulunduğu konumlar (diskretize edilmiş x-y koordinatları).
- **Aksiyonlar (Actions):** Robotun yapabileceği hareketler (ileri gitme, geri gitme, sağa veya sola dönme).
- **Ödüller (Rewards):** Hedefe yaklaşlığında olumlu ödül, engellerden kaçınırken ceza.
- **Geçiş Olasılıkları (Transition Probabilities):** Robotun bir durumdan diğerine geçme olasılığı, burada deterministik (yani, belirli bir aksiyon her zaman aynı sonucu verir).

MDP'yi mülakatta şöyle açıklayabilirsin:

- "Problemi bir Markov Karar Süreci (MDP) olarak modelledim. Burada robotun her durumu bir konum, aksiyonları ise hareketleri temsil ediyor. Her aksiyondan sonra robot bir ödül alıyor ve bu ödüller kullanılarak robot, en iyi stratejiyi öğreniyor."

### 5. Genel Anlatım Stratejisi

Mülakat sırasında şunlara dikkat edebilirsin:

1. **Q-learning Algoritması:** Temel Q-learning mantığını açıklayıp, robotun durumdan aksiyon seçme sürecini ve ödüllerle nasıl öğrenme yaptığı detaylandırıbilirsin.
2. **Politika (Epsilon-Greedy):** Keşif ve kazanım dengesini nasıl kurduğunu ve neden epsilon-greedy kullandığını vurgulayabilirsin.
3. **Ödül Fonksiyonu:** Hedefe ulaşma ve engelden kaçınma ödüllerini nasıl belirlediğini ve bu ödül fonksiyonunun nasıl etkili olduğunu anlatabilirsin.
4. **Markov Karar Süreci:** Problemin MDP çerçevesinde ele alındığını ve bunun nasıl sistematik bir öğrenme sağladığını belirtebilirsin.

# ARAMA ALGORİTMALARI

## Doğrusal Arama (Linear Search)

```
# Linear Search in Python
```

```
def linearSearch(array, n, x):  
  
    for i in range(0, n):  
        if (array[i] == x):  
            return i  
    return -1
```

## Binary Search (İkili Arama)

```
def binarysearch(array, x, low, high):  
  
    # Repeat until the pointers low and high meet each other  
    while low <= high:  
  
        mid = low + (high - low)//2  
  
        if array[mid] == x:  
            return mid  
  
        elif array[mid] < x:  
            low = mid + 1  
  
        else:  
            high = mid - 1  
  
    return -1
```

Doğrusal Arama	İkili Arama
Doğrusal aramada girdi verilerinin sıralı olmasına gerek yoktur.	İkili aramada girdi verilerinin sıralı olması gereklidir.
Sıralı arama olarak da adlandırılır.	Yarım aralıklı arama olarak da adlandırılır.
Doğrusal aramanın zaman karmaşıklığı $O(n)$ .	İkili aramanın zaman karmaşıklığı $O(\log n)$ .
Çok boyutlu dizi kullanılabilir.	Sadece tek boyutlu dizi kullanılmıştır.
Doğrusal arama eşitlik karşılaştırmaları gerçekleştirir	İkili arama sıralama karşılaştırmaları gerçekleştirir
Daha az karmaşıktır.	Daha karmaşıktır.
Çok yavaş bir süreç.	Çok hızlı bir işlem.

## Ara Değer Arama ( Interpolation Search )

- Sadece önceden sıralanmış veri setlerinde kullanılabilir
  - Verilerin listenin içinde eşit aralıklarla dağıldığını varsayar.
  - Listenin başlangıç değerinden sonra her eleman arasındaki fark aynıdır.
  - Bu varsayıma dayanarak, aranan değerin konumu tahmin edilir.
  - Tahmin edilen konum kontrol edilir ve eğer yanlışsa, arama daha dar bir aralıkta (sağda veya solda) devam ettirilir

En küçük ve en büyük eleman kullanılarak tahmini konum hesaplanır. Tahmini konumdaki değer kontrol edilir. Eğer tahmini konumdaki değer aranan değer ise, arama başarıyla tamamlanır ve konum döndürülür. aranan değerden küçükse, arama listenin sağ yarısında devam ettirilir. aranan değerden büyükse, arama listenin sol yarısında devam ettirilir. Bu adımlar, aranan değer bulunana kadar veya listenin tüm elemanları kontrol edilene kadar tekrarlanır.

```

def interpolation_arama(dizi, n, aranan):
    bas = 0
    son = n - 1
    while bas <= son:
        pozisyon = bas + ((aranan - dizi[bas]) * (son - bas)) // (dizi[son] - dizi[bas]))
        if dizi[pozisyon] == aranan:
            return pozisyon
        elif dizi[pozisyon] < aranan:
            bas = pozisyon + 1
        else:
            son = pozisyon - 1
    return -1

```

## Atla Ara Bul (Jump) Algoritması

- Engelli koşuda engelleri aşarak ilerleyen bir atlet gibi çalışır
- Listeyi önceden belirlenen büyülüklükte parçalara ayırır.
- İlk olarak listenin başından büyük bir adım atlar ve o konumdaği öğeyi kontrol eder. Eğer aranan değer bu konumdaysa, arama başarıyla sona erer.
- Eğer atlanan konumdaysa, atlanan aralığa geri dönülerek o aralıkta doğrusal arama yapılır.
- Bu sayede, tek tek tüm öğeleri kontrol etmek yerine daha hızlı bir şekilde arama gerçekleştirilebilir.

```

def jump_search(dizi, eleman, n):
    atlama = int(math.sqrt(n))
    konum = 0
    while dizi[min(atlama, n)-1] < eleman:
        konum = atlama
        atlama += int(math.sqrt(n))
        if konum >= n:           return -1
    while dizi[konum] < eleman:
        konum += 1
        if konum == min(atlama, n):      return -1
    if dizi[konum] == eleman:          return konum
    return -1

```

## Üstel (Exponential) Arama

- Define avcisının harita işaretlerini takip ederek defineye yaklaşması gibi.
- Listeyi, üstel olarak büyüyen parçalara ayırır.
- Örneğin, listenin uzunluğu 1024 ise,
- ilk arama 1024. konuma, yani listenin sonuna yakın bir konuma yapılır.
- aranan değer burada değilse, atlama mesafesi geriye çekilir ve arama 512 ile 1023 arasındaki kısımda devam eder.
- her adımda atlama mesafesi küçülerek arama alanı daraltılır.

```

def usluArama(dizi, x, n):
    konum = 1
    while konum < n and dizi[konum - 1] < x:
        konum *= 2
    for i in range(konum // 2, n):
        if dizi[i] == x:
            return i
    return -1

```

## Hash Tablo Arama Algoritması

- Bir anahtarın kilide tam olarak oturması gibi çalışır.
- Veriler, anahtar kelimeler ve bunlara karşılık gelen değerlerden oluşur.
- Anahtar kelimeler, hash fonksiyonu ile benzersiz değerlere dönüştürülür.
- Hash değeri, hash tablosundaki verilerin yerini işaret eder.
- Aranan anahtar kelimenin hash değeri ile konumda (kova) arama yapılır. Eğer kova boş değilse, anahtar kelime kovadaki değerlerle karşılaştırılır, aranan değer bulunursa işlem tamamlanır.

```

def hash_arama(hash_table, anahtar):
    index = hash_fonksiyonu(anahtar)
    while hash_table[index] is not None:
        if hash_table[index][0] == anahtar:
            return hash_table[index][1]
        index = (index + 1) % len(hash_table)
    return None

```

```

def hash_fonksiyonu(anahtar):
    toplam = 0
    for karakter in anahtar:
        toplam += ord(karakter)
    return toplam % len(hash_table)

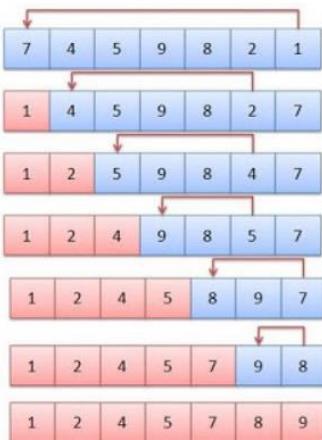
```

Algoritma	Kullanılabilirlik	Zaman Karmaşıklığı	En İyi Durum	Ortalama Durum	En Kötü Durum
Doğrusal Arama	Sırasız, sıralı	$O(n)$	$O(1)$	$O(n)$	$O(n)$
İkili Arama	Sıralı	$O(\log n)$	$O(1)$	$O(\log n)$	$O(\log n)$
Atlama Araması	Sıralı	$O(\sqrt{n})$	$O(1)$	$O(\sqrt{n})$	$O(\sqrt{n})$
Üçlü Arama	Sıralı	$O(\log_3 n)$	$O(1)$	$O(\log_3 n)$	$O(\log_3 n)$
Interpolasyon Araması	Sıralı	$O(\log \log n)$	$O(1)$	$O(\log n)$	$O(n)$
Eksponansiyel Arama	Sıralı	$O(\log n)$	$O(1)$	$O(\log n)$	$O(\log n)$

# SIRALAMA ALGORİTMALARI

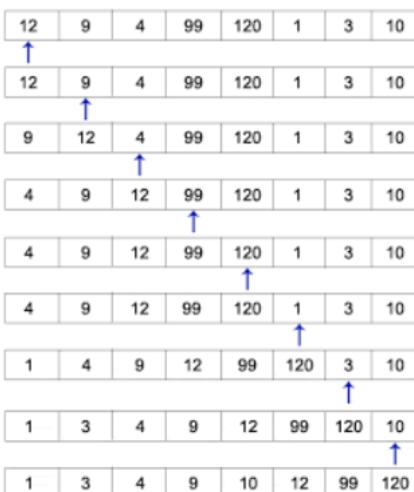
## 1. Seçmeli Sıralama (Selection Sort)

İlk elemandan son elemana kadar her elemanı kendinden sonra gelen kendinden küçük elemanların en küçüğü ile yer değiştiriyoruz.



## 2. Eklemeli Sıralama (Insertion Sort)

Bu algoritma dizinin 2. elemanından başlar ve son elemana varana kadar üzerine geldiği her elemanı, o elemandan önce gelip de o elemandan büyük olan elemanlarla yer değiştirir. Bu durumda işlem sırasında üzerine geldiğimiz bir elemanın solunda kalan elemanlar sıralı, sağındakiler ise karışık düzende olacaktır.



## 2. Kabuk Sıralaması (Shell Sort)

Yukarıda anlattığım eklemeli sıralama algoritmasını bir düşünelim. Çok büyük bir dizi üzerinde bu algoritmayı kullandığımızda, eğer dizinin küçük elemanları dizinin sonlarında duruyorsa, onları dizinin başına kaydırınmak bizim için bir hayli zor bir işlem olacaktır. Şunu da biliyoruz ki eklemeli sıralama algoritması sıralıya yakın şekilde düzenlenmiş dizilerde çok daha hızlı çalışmaktadır. Bu durumda eklemeli sıralama algoritmasının biraz daha geliştirilmiş versiyonu olan kabuk sıralaması imdadımıza yetişiyor. Kabuk sıralaması, diziyi büyük parçalara bölüp bu parçaları bir tablonun satırlarımış gibi alt alta düşünerek, bu tablonun sütunlarını eklemeli sıralama

algoritmasıyla sıralı hale getiriyor. Daha sonra diziyi biraz daha küçük parçalara bölüp tekrar tekrar aynı işlemleri yapıyor ve en sonunda dizi elemanlarını birer birer ayırmış oluyor, yani tek sütun haline getirmiş oluyor ve o sütunu da eklemeli sıralama mantığıyla sıraladığında sıralama işlemi tamamlanmış oluyor.

{13, 14, 94, 33, 82, 25, 59, 94, 65, 23, 45, 27, 73, 25, 39}

Bu diziyi ilk olarak 5'e böldüğümüzü varsayıyalım. Aşağıdaki tabloyu elde ediyoruz:

13	14	94	33	82
25	59	94	65	23
45	27	73	25	39

Şimdi bu tablonun sütunlarını sıralıyoruz ve şu tabloyu elde ediyoruz:

13	14	73	25	23
25	27	94	33	39
45	29	94	65	82

Bu durumda dizimizin yeni hali şu oluyor:

{14, 73, 25, 23, 13, 27, 94, 33, 39, 25, 59, 94, 65, 82, 45}

Şimdi bu diziyi 3'e böldüğümüzü düşünelim. Aşağıdaki tabloyu elde ediyoruz.

14	73	25
23	13	27
94	33	39
25	59	94
65	82	45

Şimdi bu tablonun sütunlarını sıralıyoruz ve aşağıdaki tablo ile diziyi elde ediyoruz

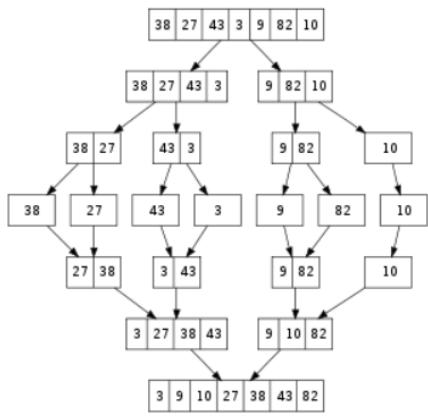
14	13	25
23	33	27
25	59	39
65	73	45
94	82	94

{14, 13, 25, 23, 33, 27, 25, 59, 39, 65, 73, 45, 94, 82, 94}

Son olarak da dizimizi 1'e böldüğümüzü düşünürsek, basit bir eklemeli sıralama işlemi yaptığımızı ve elemanların neredeyse sıralanmış olduğunu görüyoruz. Basit birkaç işlem ile karmaşık bir diziyi eklemeli sıralama ile kolayca sıralayabileceğimiz bir dizi haline getirdik.

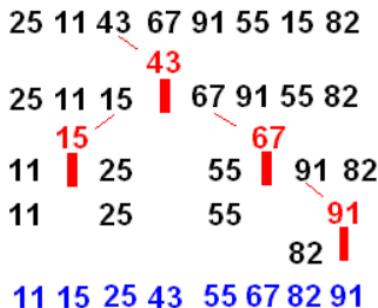
### 3. Birleşirmeli Sıralama (Merge Sort)

Sıralı iki diziyi yine sıralı olacak şekilde birleştirmek kolay bir işlemidir. Dizilerin ilk elemanlarını karşılaştırırsınız ve küçük olanını alırsınız. Daha sonra diğer dizide kalan almadığınız elemanı diğer dizinin ikinci elemanıyla karşılaştırırsınız ve yine küçük olanı alırsınız. İki dizide de eleman kalmayınca kadar bu böyle sürüp gider. Birleşirmeli sıralama algoritmamız da buna benzer bir mantık kullanıyor. Diziyi ikiye bölüp oluşan alt kümeleri tekrar birleşirmeli sıralama algoritmasına alıyor ve böylece dizi sürekli ikiye bölünmüş oluyor. Elemanları ikişerlik olarak karşılaştırıp sıralı biçimde birleştiriyor, daha sonra bu iki elemanlı alt kümeleri sıralı biçimde birleştirerek dört elemanlı alt kümeler elde ediyor. Böyle devam ettiğimizde, örneğin dizimiz 16 elemanlı ise elimizde 8 elemanlı, elemanları sıralı iki alt küme kalıyor. Bu alt kümeleri de aynı şekilde sıralı biçimde birleştirdiğimiz zaman dizimiz sıralanmış oluyor.



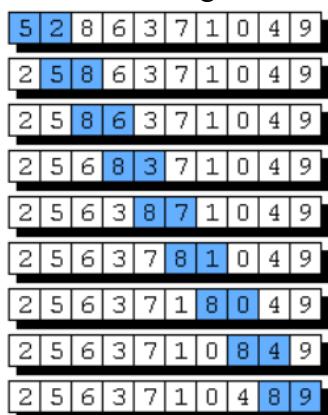
## 5. Hızlı Sıralama (Quick Sort)

Hızlı sıralama algoritması, verilen dizinin ortasına yakın bir yerinden bir pivot eleman seçer ve pivottan büyük elemanları sağa, pivottan küçük elemanları sola kaydırır. Bu işlemi yaptıktan sonra pivot elemanın solundaki ve sağındaki dizilerden ayrı ayrı pivot elemanlar seçer ve o dizileri kendi içerisinde tekrar bu işleme tabi tutar. Reküratif bir şekilde bu işlem devam ettikten sonra tüm dizi sıralanmış olur.



## 6. Kabarcık Sıralaması (Bubble Sort)

Mantık ve kodlama olarak çok basit fakat nasıl çalışacağını düşündüğümüzde işlemcinin canını sıkabilecek kadar çok işlem yapan bir algoritma. Dizinin başından başlıyor ve sonuna kadar tüm elemanları kendisinden önceki eleman ile karşılaştırıp, gerekirse yerlerini değiştiriyor. Bu şekilde diziyi dizi tamamen sıralanana kadar tekrar tekrar baştan sona tariyor. Dizinin tamamen sıralandığını, yani yer değiştirecek hiçbir eleman kaldığını anladığında çalışmayı durduruyor ve sıralama bitmiş oluyor.



Algoritma	Sıralı Dizi Üzerinde (ms)	Tersten Sıralı Dizi Üzerinde (ms)	Rastgele Sıralı Dizi Üzerinde (ms)
Seçmeli Sıralama (Selection Sort)	583334	511961	516530
Ekleme Sıralama (Insertion Sort)	10	407946	234287
Kabuk Sıralaması (Shell Sort)	80	100	230
Birleştirilmeli Sıralama (Merge Sort)	260	230	330
Hızlı Sıralama (Quick Sort)	60	90	160
Kabarcık Sıralaması (Bubble Sort)	10	1536317	2062972

# OOP'nin 4 Ana Prensibi: Encapsulation, Inheritance, Abstraction, Polymorphism

## Encapsulation (Kapsülleme)

Türkçe'de **kapsülleme** veya **sarmalama** olarak bilinen encapsulation prensibi class'ın property'lerini korumaya almasıdır. Bu korumayı sağlamak için private ve protected access modifier'larını (erişim belirteci) kullanız. Eğer bir property'i tanımlarken private kullanırsak o class dışında hiçbir yerden erişim sağlanamaz. Protected kullanırsak da yalnızca subclass'lar ve aynı package'da bulunan classlar tarafından erişilir. Encapsulation prensibinin maksadı class'a ait property'lere her classın istediği gibi erişmesini ve değiştirmesini engellemektir. Erişim kısıtlanmayacaksa bile erişimi getter/setter methodları gibi yöntemlerle kontrol altına almaktır. Zaten encapsulation prensibine gözümüzün en aşina olduğu yer getter/setter methodlarıdır. Bu prensiple alakalı biri getter/setter methodları olmak üzere iki örnek vereceğim.

Yukarıdaki örnekte gördüğünüz gibi hiçbir classın Personel property'lerine doğrudan erişim hakkı yoktur. Erişimi yalnızca getter/setter methodlarıyla yapabilirler. Tahmin edileceği gibi getter/setter methodları private olamaz.

```
public class Personel {  
  
    private Long personelNu;  
    private String isim;  
    private String soyisim;  
    private String departman;  
  
    public Long getPersonelNu() {  
        return personelNu;  
    }  
  
    public void setPersonelNu(Long personelNu) {  
        this.personelNu = personelNu;  
    }  
  
    public String getIsim() {  
        return isim;  
    }  
  
    public void setIsim(String isim) {  
        this.isim = isim;  
    }  
  
    public String getSoyisim() {  
        return soyisim;  
    }  
  
    public void setSoyisim(String soyisim) {  
        this.soyisim = soyisim;  
    }  
  
    public String getDepartman() {  
        return departman;  
    }  
  
    public void setDepartman(String departman) {  
        this.departman = departman;  
    }  
}  
  
1  public class Sayac {  
2  
3      private Integer sayac;  
4  
5      public Sayac() {  
6          sayac = 1;  
7      }  
8  
9      public Integer sayacArtir(){  
10         return sayac++;  
11     }  
12  
13     public Integer getSayac() {  
14         return sayac;  
15     }  
16 }
```

Yukarıda ise örnek bir Sayac classı bulunmaktadır. Diğer class'lar erişim sağlanmak istediğiinde getter methodunu, sayacı artırmak istediği ise sayacArtir() methodunu kullanmak zorunda. Yani herhangi bir class'in ornekSayac.sayac += 1 gibi bir şey yazmaya yetkisi yok. Çünkü eğer herhangi bir class bunu yazabilirse sayacı istediği gibi

manipüle edebilir. Bu sebeple encapsulation'ı kullanarak Sayac'ın property'sini korumaya almış oluyoruz.

Kapsülleme, bir nesnenin verilerini ve o veriler üzerinde işlem yapan yöntemleri (metotları) bir araya toplar ve dışarıdan doğrudan erişimi sınırlar. Bu prensip, bir nesnenin sadece gerekli bilgilerini dış dünyaya açmasını, diğer detayları gizli tutmasını sağlar.

### Inheritance (Kalıtım)

Adından tahmin edilebileceği üzere herhangi bir class'ın üst class'lara ait olan method ve property'leri kalıtım yoluyla almasıdır. Günlük hayattan bir örnekle anlaması çok daha kolay. Telefon adında bir class düşünelim. Bu class tüm telefonlarda ortak olan bazı property'leri ve method'ları barındıracak. `aramaYap()`; `mesajGonder()`; methodları ve `imeiNumber` property'sini örnek verebiliriz. Bu class'ın tüm alt class'lari bu methodları ve property'leri kullanabilecektir. Böylece AkilliTelefon ve TusluTelefon classlarını tanımlarken kalıtım yoluyla alınan bu property'leri ve method'ları tekrar tanımlamak zorunda kalmayacağız.

Kalıtım, bir sınıfın özelliklerini başka bir sınıfa miras olarak aktarabilmesidir. Alt sınıflar, üst sınıfların özelliklerini ve metotlarını devralabilir, böylece kod tekrarını azaltarak daha esnek yapılar oluşturur.

Yukarıdaki örnekte görüldü üzere AkilliTelefon ve TusluTelefon class'larının instance'ları hem Telefon class'ına ait olan `aramaYap()` ve `mesajGonder()` methodlarını ve `imeiNumber` property'sini **inheritance (kalıtım)** yoluyla alıp kullanıyor hem de kendi class'lara ait olan `interneteBaglan()` ve `yilanOyunuAc()` methodlarını kullanıyor.

```
public class Telefon {  
  
    protected String imeiNumber;  
  
    protected void aramaYap(){  
        System.out.println("Arama yapıldı.");  
    }  
  
    protected void mesajGonder(){  
        System.out.println("Mesaj gönderildi.");  
    }  
  
    public String getImeiNumber() {  
        return imeiNumber;  
    }  
  
    public void setImeiNumber(String imeiNumber) {  
        this.imeiNumber = imeiNumber;  
    }  
}
```

Şimdi de bu class'ın iki subclass'ını tanımlayalım.

```
1  public class AkilliTelefon extends Telefon{  
2  
3      protected void interneteBaglan() {  
4          System.out.println("İnternete bağlandı.");  
5      }  
6  }  
AkilliTelefon.java hosted with ❤ by GitHub
```

```
1  public class TusluTelefon extends Telefon {  
2  
3      protected void yilanOyununuAc(){  
4          System.out.println("Yılan oyunu açıldı.");  
5      }  
6  }  
TusluTelefon.java hosted with ❤ by GitHub
```

Şimdi de Test class'ını yazalım.

```
1  public class Test {  
2  
3      public static void main(String[] args) {  
4  
5          AkilliTelefon akilliTelefon = new AkilliTelefon();  
6          akilliTelefon.setImeiNumber("354816220461203");  
7          akilliTelefon.aramaYap();  
8          akilliTelefon.interneteBaglan();  
9  
10         TusluTelefon tusluTelefon = new TusluTelefon();  
11         tusluTelefon.setImeiNumber("354816221162234");  
12         tusluTelefon.mesajGonder();  
13         tusluTelefon.yilanOyununuAc();  
14     }  
15  
16 }
```

## Abstraction (Soyutlama)

Bu prensibe doğrudan bir örnekle başlıyoruz. Aşağıdaki kod parçacığının ne yaptığını, ne işe yaradığını biliyor musunuz? Biraz incelerseniz ne iş yaptığına tahmin edeceksiniz ama buna hiç gerek yok. Ben değil, abstraction prensibi söylüyor bunu. Aşağıdaki kod parçacığı IDE'nizi her açtığınızda en az bir kere kullandığınız System.out.println(); methodunun arkaplanda yaptığı işlerin bir kısmı. Fakat siz System.out.println(); yazarken arka planda neler döndüğünü bilmeden yazıyorsunuz. Bunun sebebi abstraction prensibi. Daha derli toplu bir ifadeyle objelerin ayrıntılarıyla uğraşmak yerine yalnızca girdi ve çıktılarına odaklanarak tasarımları daha iyi oluşturmayı ve anlamayı sağlamaktır.

```
try {
    synchronized (this) {
        ensureOpen();
        textOut.write(s);
        textOut.flushBuffer();
        charOut.flushBuffer();
        if (autoFlush && (s.indexOf('\n') >= 0))
            out.flush();
    }
}
catch (InterruptedException x) {
    Thread.currentThread().interrupt();
}
catch (IOException x) {
    trouble = true;
}
```

Buna prensibe araba kullanmak örneği de verilebilir. Araba kullanan birinin gaz pedalına ve frene bastığında veya vites değiştirdiğinde neler olduğunu bilmesine gerek yok. Yalnızca girdi ve çıktıları öğrenerek rahatça araba kullanabilir.

Soyutlama, karmaşık sistemlerin sadece gerekli detaylarını sunarak gereksiz detayları gizler. Bu sayede kullanıcılar veya diğer kod bölgüleri, nesnenin yalnızca önemli işlevleriyle ilgilenir.

## Polymorphism (Çok Biçimlilik)

Polymorphism, (çok biçimlilik) methodların objeye göre farklı çıktılar üretmesi veya farklı işler yapmasıdır. Yani alışageldiğimiz gibi methodlara sabit görevler vermek yerine onlara çok biçimli (polimorf) davranışacak şekilde bir esneklik vermektedir.

Polymorphism, farklı nesnelerin aynı arayüzü kullanarak farklı şekillerde davranışılmasını sağlar. Aynı metot, farklı sınıflarda farklı işlevsellikler gösterebilir.

## Polymorphism Örnek 1 (Method Overriding)

```
public class Hayvan {  
  
    public void sesCikar(){  
  
        System.out.println("Hayvan sesi.");  
    }  
  
}  
  
public class Kedi extends Hayvan {  
  
    @Override  
    public void sesCikar() {  
        System.out.println("Miyav");  
    }  
}  
  
public class Kopek extends Hayvan {  
  
    @Override  
    public void sesCikar() {  
        System.out.println("Hav Hav");  
    }  
}
```

Yukarıda Hayvan class'ı ve Kedi — Kopek subclass'ları var. Hayvan class'ının sesCikar metodu olmasına rağmen Kedi ve Kopek subclass'ları için özel sesCikar metodu tanımladık. Aşağıda ise Hayvan tipindeki değişkene Kedi instance'ı verince kedi sesi, Kopek instance'ı verince köpek sesi çıkaracak.

```
public class Test {  
  
    public static void main(String[] args) {  
  
        Hayvan hayvan = new Hayvan();  
        hayvan.sesCikar();  
  
        System.out.println("-----");  
  
        hayvan = new Kedi();  
        hayvan.sesCikar();  
  
        System.out.println("-----");  
  
        hayvan = new Kopek();  
        hayvan.sesCikar();  
  
    }  
}
```

Çıktı:  
Hayvan sesi.  
— — —  
Miyav  
— — —  
Hav Hav

## Polymorphism Örnek 2 (Method Overloading)

Bu örnekte ise aynı isimde iki method var. Bu iki method'un biri iki Integer değeri alıyor, diğeri ise bir Integer değeri alıyor. Biz kaç parametre göndererek çağrırsak o method çalışacaktır.

```
public void yemekYe(Integer hamburgerSayisi, Integer kolaSayisi){  
    System.out.println(((hamburgerSayisi*261)+(kolaSayisi*168)) + " kalori alındı.");  
}  
  
public void yemekYe(Integer hamburgerSayisi){  
    System.out.println((hamburgerSayisi*261)+ " kalori alındı.");  
}  
  
public void test() {  
  
    yemekYe(2, 1);  
    yemekYe(2);  
}
```

## C Dili ile İlgili Sorular

### 1. C'de pointer nedir ve ne amaçla kullanılır?

- **Cevap:** Pointer, bellekteki bir adresi saklayan bir değişkendir. C dilinde bellek yönetimini sağlamak ve veri yapılarına doğrudan erişim yapmak için kullanılır. Örneğin, dizilerde, fonksiyonlarda ve dinamik bellek tahsisinde yaygın olarak kullanılır.

### 2. Malloc ve calloc arasındaki fark nedir?

- **Cevap:** malloc, bellekte belirtlen boyutta bir alan ayırır ancak bu alan sıfırlanmaz, yani rasgele veri içerir. calloc ise bellekte belirtlen miktarda sıfırlanmış bir alan ayırır. malloc için örnek: malloc(size \* sizeof(int)), calloc için örnek: calloc(num\_elements, sizeof(int)).

### 3. C'de yapı (struct) ve birleşim (union) arasındaki fark nedir?

- **Cevap:** struct yapısında tüm üyeleri bellekte ayrı ayrı yer kaplar ve aynı anda erişilebilir. union ise üyelerin belleği paylaştığı bir yapı türüdür ve sadece bir üye aktif olabilir. struct, farklı veri türlerini aynı yapıda saklamak için, union ise bellek tasarrufu sağlamak için kullanılır.

### 4. Statik (static) değişkenler nedir ve ne amaçla kullanılır?

- **Cevap:** Statik değişkenler, tanımlandıkları blok veya fonksiyondan dışarıya erişilemezler ve programın ömrü boyunca bellekte saklanır. Fonksiyonlarda static değişkenler, çağrılar arasında değerlerini korur. Dosya düzeyinde tanımlanmış static değişkenler, sadece o dosya içinde geçerlidir.

### 5. Dangling pointer nedir? Nasıl oluşur?

- **Cevap:** Dangling pointer, bellekte geçersiz veya serbest bırakılmış bir adresi gösteren pointer'dır. Örneğin, bir pointer dinamik olarak tahsis edilen belleği serbest bıraktıktan sonra kullanılmaya çalışıldığında veya yerel bir değişkenin adresi döndürüldüğünde oluşur.

### 1. C'de dizi ile pointer arasındaki fark nedir?

- **Cevap:** Dizi, sabit bir boyut ve sabit bir bellek adresine sahiptir. Diziler bellekte ardışık olarak yer alır. Pointer ise herhangi bir bellek adresini gösterebilir ve daha esnekdir. Pointer bir diziyi işaret edebilir veya dinamik bellek yönetiminde kullanılabilir.

### 2. Fonksiyon işaretçileri (function pointers) nedir ve nasıl kullanılır?

- **Cevap:** Fonksiyon işaretçileri, bir fonksiyonun adresini saklayan pointer'lardır. Dinamik olarak hangi fonksiyonun çağrılacağını belirlemek veya callback fonksiyonları tanımlamak için kullanılır. Kullanımı: int (\*func\_ptr)(int, int) = &functionName;;

### 3. Stack ve Heap bellek farkı nedir?

- **Cevap:** Stack, otomatik bellek yönetimi sağlar, değişkenler fonksiyon sona erdiğinde otomatik olarak silinir. Heap ise dinamik bellek yönetimi sağlar, malloc

veya calloc ile ayrılan bellek, free ile manuel olarak serbest bırakılana kadar varlığını sürdürür.

#### 4. Static ve global değişken arasındaki fark nedir?

- Cevap: Global değişken, tüm program boyunca herhangi bir yerden erişilebilen bir değişkendir. Static değişken ise, tanımlandığı dosya veya blok içinde geçerlidir ve bellekte program boyunca kalır ancak dış dosyalardan erişilemez.

#### 5. Volatile anahtar kelimesi ne işe yarar?

- Cevap: volatile, değişkenin değerinin program dışında başka bir unsur (örneğin bir donanım) tarafından değiştirilebileceğini belirtir. Bu nedenle, derleyici optimizasyon yaparken bu değişkenin değerini her seferinde bellekten okur, cache'lemeyi engeller.

---

### Nesne Yönelimli Programlama (OOP) ile İlgili Sorular

#### 1. Kapsülleme (Encapsulation) nedir? Örnekle açıklayın.

- **Cevap:** Kapsülleme, bir nesnenin verilerini ve bu verilere erişim sağlayan yöntemleri bir arada tutarak dışarıya açılan kısmını sınırlar. Örneğin, bir BankAccount sınıfında balance değişkeni gizlenebilir ve sadece deposit ve withdraw gibi metodlarla erişilebilir.

#### 2. Kalıtım (Inheritance) nedir? Neden kullanılır?

- **Cevap:** Kalıtım, bir sınıfın başka bir sınıfın özelliklerini ve metodlarını devralmasını sağlar. Kalıtım, kodun tekrarını azaltarak daha modüler ve bakımı kolay kod yazmaya imkan verir. Örneğin, Animal sınıfından türeyen Dog ve Cat sınıfları, Animal sınıfının özelliklerini miras alır.

#### 3. Polymorphism nedir ve nasıl çalışır?

- **Cevap:** Polymorphism, aynı metod isminin farklı sınıflarda farklı davranışlar gösterebilmesidir. Örneğin, bir Shape sınıfında draw() методu her şeklin kendi çizim işlemini yapabilir. Polymorphism, compile-time (zamanında) ve run-time (çalışma zamanında) polymorphism olarak ikiye ayrılır. Compile-time polymorphism, metod overloading; run-time polymorphism ise metod overriding ile sağlanır.

#### 4. Soyut sınıf (Abstract Class) ile arayüz (Interface) arasındaki fark nedir?

- **Cevap:** Soyut sınıf, bazı metodları tanımlanmış, bazıları soyut olan bir sınıf türüdür ve diğer sınıflar için temel olarak kullanılır. Arayüz ise tüm metodların soyut olduğu bir yapıdır ve çoklu kalıtım için kullanılır. Soyut sınıf, ortak davranışları tanımlamak için, arayüz ise benzer işlevleri olan sınıflar için genel bir çerçeve sağlamak için kullanılır.

#### 5. "Overloading" ve "Overriding" arasındaki fark nedir?

- **Cevap:** Overloading, aynı isimde farklı parametre tipleri veya sayısına sahip metodların aynı sınıfta tanımlanmasıdır. Overriding ise üst sınıfta tanımlanmış bir metodun alt sınıfta yeniden tanımlanmasıdır. Overloading compile-time, overriding ise run-time polymorphism örneğidir.

#### 6. Virtual function (sanal fonksiyon) nedir?

- **Cevap:** Sanal fonksiyon, taban sınıfta tanımlanmış ancak alt sınıflarda geçersiz kılınabilecek bir fonksiyondur. Sanal fonksiyonlar, run-time polymorphism sağlar. Örneğin, bir Animal sınıfındaki speak() metodu sanal olarak tanımlandığında, Dog ve Cat sınıflarında farklı şekillerde uygulanabilir.

#### 7. C++ ile C arasındaki farklardan bazıları nelerdir?

- **Cevap:** C, prosedürel bir dildir ve yapıların, pointerların ve fonksiyonların öne çıktığı bir yapıdadır. C++ ise nesne yönelimli bir dildir ve kalıtım, kapsülleme gibi OOP prensiplerini destekler. Ayrıca, C++ class ve template gibi daha ileri özelliklere sahiptir ve C'de olmayan referanslar ve sanal fonksiyonlar gibi özellikleri vardır.

#### 1. OOP'de "Constructor" ve "Destructor" nedir?

- **Cevap:** Constructor, bir nesne oluşturulurken çağrılan özel bir metottur ve nesneyi ilk değerlerine ayarlar. Destructor ise nesne yok edilirken çağrılır ve belleği temizlemek veya kaynakları serbest bırakmak için kullanılır.

#### 2. Nesne Yönelimli Programlamada SOLID prensipleri nelerdir?

- **Cevap:** SOLID, beş temel tasarım prensibini temsil eder:
  - **Single Responsibility Principle:** Her sınıf yalnızca bir sorumluluğa sahip olmalıdır.
  - **Open-Closed Principle:** Sınıflar genişletmeye açık, değişikliğe kapalı olmalıdır.
  - **Liskov Substitution Principle:** Taban sınıf nesnesi yerine alt sınıf nesnesi kullanılabilirmeli.
  - **Interface Segregation Principle:** Kullanıcıların ihtiyacı olmayan işlevleri zorunlu olarak almaması için arayüzler ayrılmalı.
  - **Dependency Inversion Principle:** Sınıflar düşük seviyeli detaylara bağımlı olmamalı, arayzlere veya soyut sınıflara bağımlı olmalıdır.

#### 3. Sanal masa (Virtual Table) nedir ve nasıl çalışır?

- **Cevap:** Sanal masa (vtable), sanal metodların adreslerini içeren bir yapıdır. Bir sınıf sanal metodlara sahipse, nesnenin sanal masası (vtable) kullanılarak hangi sınıfın metodu çağrılabileceği belirlenir. Polymorphism sağlamak için önemlidir.

#### 4. OOP'de "has-a" ve "is-a" ilişkisi nedir?

- **Cevap:** "Is-a" ilişkisi, kalıtım yoluyla kurulur, örneğin bir Dog bir Animaldır. "Has-a" ilişkisi ise kompozisyon ile kurulur, örneğin bir Car bir Enginee sahiptir. Kompozisyon, yeniden kullanılabilirliği artırır.

#### 5. C++ dilinde "this" işaretçisi nedir ve ne amaçla kullanılır?

- **Cevap:** this işaretçisi, üye fonksiyonlar içinde çağrılan nesneyi işaret eder. Aynı isimdeki üye değişken ve parametreleri ayırmak veya zincirleme fonksiyon çağrıları yapmak için kullanılır.

#### 6. Statik ve dinamik polimorfizm arasındaki fark nedir?

- **Cevap:** Statik polimorfizm, compile-time'da belirlenir ve metod overloading ile sağlanır. Dinamik polimorfizm ise run-time'da belirlenir ve sanal fonksiyonlar (virtual functions) ile sağlanır.

#### 7. Örnekle OOP'de Soyutlama ile Kapsülleme arasındaki fark nedir?

- **Cevap:** Soyutlama, gereksiz detayları gizleyerek nesnelerin yalnızca önemli yönlerini gösterir (örneğin bir arabayı sürmek için motorun iç yapısını bilmeye gerek yoktur). Kapsülleme ise verileri gizler ve sadece gerekli metodları açığa çıkarır. Örneğin, bir BankAccount sınıfında balance değişkenini özel (private) yapmak kapsüle etmedir; deposit ve withdraw gibi işlemler ise soyutlamadır.

#### 8. C++ dilinde "RAII" (Resource Acquisition Is Initialization) nedir?

- **Cevap:** RAII, bir nesnenin yaşam döngüsü boyunca bir kaynağı yönetme konseptidir. Nesne oluşturulduğunda kaynak ayrılır, nesne yok edildiğinde kaynak serbest bırakılır. RAII, C++'ta bellek ve dosya yönetimi gibi işlemlerde kullanılır.