

## Standardizasyon

Standardizasyonun amacı, her bir verinin ortalamaya göre ne kadar uzaklaştığını belirlemek ve bu uzaklığı standart sapma birimleri cinsinden ifade etmektir. Böylece tüm veri seti, ortalaması 0 ve standart sapması 1 olan bir ölçeğe getirilir. Bu süreç **standardizasyon** olarak adlandırılır.

$$z = \frac{x_i - \mu}{\sigma}$$

Bu, verilerin ortalamadan ne kadar saptığını görmek için kullanılır ve belirli algoritmaların performansını artırır.

## Normalizasyon (Min-Max Scaling)

Verileri belirli bir aralığa (genellikle 0 ile 1) sıkıştırır. Minimum ve maksimum değerlere göre verileri yeniden ölçeklendirir.

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}}$$

Bu, mesafe tabanlı algoritmaların ve sinir ağlarının performansını artırır.

## Encoding

"Encoding" dediğimiz işlem, kategorik (yani metin ya da sınıf etiketleri şeklindeki) verilerin, makine öğrenimi modellerinin daha kolay anlayıp işleyebileceği sayısal değerlere dönüştürülmesidir. Bu, modelin kategorik verilerle çalışabilmesi için gereklidir, çünkü çoğu makine öğrenimi algoritması sayısal girdi gerektirir.

Örneğin, bir veri kümesinde "Evet" ve "Hayır" gibi kategorik yanıtlar varsa, bu yanıtlar genellikle 1 ve 0 gibi sayısal değerlere dönüştürülür.

**`encoder = LabelEncoder ()`**

**`df['Outcome'] = encoder.fit_transform(df['Outcome'])`**

`LabelEncoder` sınıfı, `scikit-learn` kütüphanesinden gelen bir sınıftır. Kategorik verileri sayısal değerlere dönüştürmek için kullanılır.

`encoder = LabelEncoder ()`, `LabelEncoder` sınıfından bir örnek oluşturuluyor ve bu örnek `encoder` adlı bir değişkene atanıyor. Artık bu değişkeni kullanarak kategorik verileri sayısal değerlere dönüştürebiliriz.

`fit_transform` yöntemi, hem `fit` (veriyi inceleyip hangi etiketin hangi sayısal değere dönüşeceğini belirler) hem de `transform` (bu dönüşümü gerçekleştirir) işlemlerini aynı anda yapar.

fit metodu, kategorik verilerdeki benzersiz değerleri tespit eder ve bu değerlere sayısal etiketler atar. Örneğin, bir veri kümesinde "Evet" ve "Hayır" gibi kategorik değerler varsa, fit metodu bu iki benzersiz değeri tespit eder ve bu değerlere karşılık gelen sayısal etiketler oluşturur.

```
x_train, x_test, y_train, y_test=train_test_split(x, y, shuffle=True, test_size=0.2, random_state=42)
```

**shuffle=True:** Bu parametre, verinin karıştırılıp karıştırılmayacağını belirler. Varsayılan olarak True yani veri karıştırılır. Veriyi karıştırmak, modelin eğitimi sırasında herhangi bir sıralama yanlılığını azaltmaya yardımcı olur.

**random\_state=42:** rastgele işlemler yapıldığında aynı sonuçların tekrar elde edilmesini sağlar. 42 olarak ayarlandığında ise bu, belirli bir rastgelelik durumunu sabitler, böylece her seferinde aynı rastgele işlemi tekrar edebiliriz. Random state için herhangi bir tam sayı kullanılabilir ve bu sadece kodun yeniden üretilebilir olmasını sağlar. 42'nin seçilmesi tamamen bir tercihtir ve rastgele bir anlam taşımaz. Bu nedenle, eğer aynı veriyi ve modeli farklı zamanlarda veya farklı sistemlerde kullanırken tutarlı sonuçlar elde etmek istiyorsanız, random\_state parametresini sabit bir değere ayarlamak iyi bir uygulamadır.

- **x\_train:** Eğitim setinin özelliklerini (bağımsız değişkenler) içeren bir dataframe. Model, bu veriyi kullanarak öğrenir.
- **x\_test:** Test setinin özelliklerini içeren bir dataframe. Model, eğitildikten sonra bu veriyi kullanarak test edilir ve performansı değerlendirilir.
- **y\_train:** Eğitim setinin hedef değişkenlerini içeren bir dataframe. Model, bu çıktıların ulaşmaya çalışarak öğrenir.
- **y\_test:** Test setinin hedef değişkenlerini içeren bir dataframe. Modelin tahminleri, bu gerçek değerlerle karşılaştırılır.

Eğitim seti doğruluğu yüksek, ancak test seti doğruluğu düşükse, model muhtemelen overfitting (aşırı öğrenme) yapmıştır.

Train setinin içerisinde bağımlı ve bağımsız değişkenler: **x\_train, y\_train**

Test setinin içerisinde bağımlı ve bağımsız değişkenler: **x\_test, y\_test**

### **Hyperparameter Tuning (hiperparametre ayarlaması):**

Makine öğrenmesi modelleri, öğrenilen parametrelere ya da veriyi bölümlendirme şekillerine ek olarak öğrenilemeyen, kullanıcı tarafından baştan belirlenmesi gereken parametreler bulundurmakta ve bunlara hiperparametre denilmektedir. Genellikle hangi hiperparametrelerin model için daha uygun olduğunu bilemeyiz. Bunu model içinde deneyerek veya modele otomatik ayarlama yaparak, optimal değerlerin bulunmasını

sağlamaya çalışırız. İşte bu optimal parametreleri bulma işlemine hiperparametre ayarlaması (hyperparameter tuning) diyoruz.

Peki ben bu optimizasyonu nasıl en iyi şekilde sağlayabilirim?

Bunun için hata fonksiyonunu minimize eden parametre değerlerine erişmemiz gerekmektedir.

Makine öğrenmesi modellerinde hiperparametreleri optimize etmek için kullanılan yöntemlerden iki tanesi:

- GridSearchCV
- Random Search CV

### **GridSearchCV (grid search - cross-validation)**

#### **Grid Search (Kafes Araması):**

Belirli bir model için denemek istediğiniz hiperparametre değerlerinin bir ızgarasını oluşturur. GridSearchCV, bu ızgaradaki her bir olası hiperparametre kombinasyonunu sistematik olarak dener.

#### **CV: Cross-Validation (Çapraz Doğrulama):**

Modeli değerlendirmek için veriyi k sayıda alt kümeye böler (k-fold cross-validation).

Model, k-1 alt kümede eğitilir ve kalan 1 alt kümede test edilir. Bu işlem her alt küme için tekrar edilir ve sonuçlar ortalanarak modelin genel performansı hesaplanır.

### **Random Search CV (Randomized Search Cross-Validation)**

RandomizedSearchCV, hiperparametrelerin olası kombinasyonlarını rastgele seçerek dener ve her bir kombinasyon için modelin performansını değerlendirir. Sonuçta en iyi performansı veren kombinasyonu bulur.

- **Rastgele Seçim:** n\_iter parametresi, kaç farklı kombinasyonun deneneceğini belirtir. Bu, geniş bir hiperparametre alanında hızlıca en iyi ayarları bulmak için faydalıdır.
- **Çapraz Doğrulama:** Her seçilen kombinasyon, çapraz doğrulama (cross-validation) yöntemi kullanılarak değerlendirilir. Çapraz doğrulama, veri setini birkaç alt gruba böler ve modelin her bir alt grupta ne kadar iyi performans gösterdiğini ölçer. Bu, modelin genelleme yeteneğini artırır ve overfitting riskini azaltır.
- **Performans Metrikleri:** Modelin performansını değerlendirmek için bir metrik belirlenir. Örneğin, sınıflandırma problemleri için doğruluk (accuracy), F1 skoru gibi metrikler kullanılabilir. RandomizedSearchCV en iyi performans metrik skorunu elde eden hiperparametre kombinasyonunu seçer.

## RandomizedSearchCV Kullanmanın Avantajları

- **Verimlilik:** GridSearchCV ile karşılaştırıldığında daha az sayıda kombinasyonu dener. Özellikle çok fazla hiperparametre ve değer aralığı olduğunda, GridSearchCV tüm kombinasyonları denerken RandomizedSearchCV yalnızca belirli sayıda kombinasyon dener. Bu, daha az zaman ve hesaplama kaynağı gerektirir.
- **Çeşitlilik:** Rastgele seçim, geniş bir hiperparametre alanında çeşitliliği artırır ve beklenmedik iyi sonuçlar verebilecek kombinasyonların keşfedilmesini sağlar.
- **Basitlik ve Hız:** RandomizedSearchCV'nin uygulanması basit ve hızlıdır. Az sayıda iterasyonla bile tatmin edici sonuçlar elde edilebilir.

## Model Tuning (parametre ayarı- model ayarı)

Amaç model tahmin performansını arttırmak. Model optimizasyonunu sağlamak için parametre ve hiperparametre ayarlamalarının hepsine model tuning denir. Bu sürecin amacı, hem model parametrelerini hem de hiperparametrelerini optimize ederek modelin **genelleme yeteneğini** artırmak ve test seti üzerinde en iyi performansı elde etmektir.

### Model Parametreleri:



Model eğitimi sırasında veriden öğrenilen değerler, katsayılardır. Modelin verilerle etkileşime geçtikçe güncellediği içsel ağırlıklar veya katsayılardır.

### Hiperparametreler:

Modelin eğitim sürecini kontrol eden, önceden belirlenmiş ve eğitim sırasında öğrenilmeyen değerlerdir. Örneğin, karar ağacı modelinde maksimum derinlik veya destek vektör makinelerinde (SVM) C değeri hiperparametrelerdir.

**Modeli eğitirken ayarladığınız değerler** (örneğin, öğrenme oranı, epoch sayısı, katman sayısı) **hiperparametreler** oluyor. Bu değerler, modelin eğitim sürecini ve yapısını kontrol eder.

**Modelin eğitiminden sonra elde edilen ağırlıklar ve biaslar** gibi **öğrenilen değerler** ise **parametreler** oluyor. Bu parametreler, modelin veriyi nasıl yorumladığını belirler ve eğitim süreci boyunca güncellenir.

-  Hiperparametreler: Modeli eğitmeden önce belirlenen, eğitim sürecini etkileyen sabit değerler.
-  Parametreler: Eğitim sırasında model tarafından öğrenilen ve veriyi işlemek için kullanılan değerler.

## Cross-validation (Çapraz doğrulama):

Makine öğrenmesi modellerinin performansını daha doğru ve güvenilir şekilde değerlendirmek için kullanılan bir yöntemdir. **Validation**, modelin performansını değerlendirmek için kullanılan bir yöntemdir. Modelin eğitim verisi üzerinde ne kadar iyi çalıştığını görmekten daha fazlasını içerir.

- **Veriyi Parçalara Bölme:** Tüm veri setini birkaç parçaya (**folds**) ayırırsın. Örneğin, 5 katlı (5-fold) çapraz doğrulama yapıyorsan, verini 5 eşit parçaya bölersin.

- **Eğitim ve Test:**

- 5 parçadan birini **test verisi** olarak ayırırsın, geri kalan 4 parçayı **eğitim verisi** olarak kullanırsın.
- Modeli 4 parça üzerinde eğitirsin ve ayırdığın 1 parçayı test etmek için kullanırsın.

- **Tüm Parçalar için Tekrarla:** Bu işlemi her parça için tekrar edersin, yani her seferinde farklı bir parçayı test verisi olarak kullanırsın, kalan parçalarla modeli eğitirsin. Sonuçta her parça bir kez test verisi olarak kullanılır.

- **Sonuçları Ortalama Al:** Tüm test sonuçlarını alıp ortalama bir performans ölçümü çıkarırsın. Bu sayede modelin performansı hakkında daha güvenilir bir fikir edinarsın çünkü model her veri parçası için test edilmiştir.

Yani burada yaptığımız şey şu: train datasını kendi içerisinde bölmek ve bu böldüğümüz datanın bir kısmını validation (kendi içerisinde test kısmı diye düşünebiliriz) bir kısmını train olarak almak, daha sonrasında hata matrislerine göre modeli seçmek.

1. **Train datasını bölmek:**

- Eğitim (train) datasını belirlediğiniz **K** katmana bölüyorsunuz. Örneğin, 5-fold çapraz doğrulama yapıyorsanız, train verinizi 5 eşit parçaya ayırıyorsunuz.

2. **Validation ve train seti olarak kullanmak:**

- Her iterasyonda bu parçalardan birini **validation (doğrulama)** seti olarak kullanıyorsunuz. Geri kalan **k-1** parçayı ise modelinizi eğitmek (train) için kullanıyorsunuz.
- Bu işlem, tüm katlar sırayla validation seti olarak kullanılarak tekrarlanıyor.

3. **Hata matrislerine göre performansı değerlendirmek:**

- Her iterasyon sonunda model, validation seti üzerinde test ediliyor ve hata matrisleri veya performans ölçütleri (örneğin, doğruluk, F1 skoru) hesaplanıyor.
- Tüm katmanlar için bu değerlendirmeler yapıldıktan sonra, ortalama bir performans ölçütü elde ediyorsunuz.

4. **En iyi modeli seçmek:**

- Son olarak, tüm bu iterasyonlar sonucunda en iyi performans gösteren modelin **hata matrisi** veya performans ölçütleri dikkate alınarak, en iyi model seçiliyor.

## Özet:

- Eğitim verinizi bölüyorsunuz.
- Her bölümde bir parça validation seti, geri kalanlar ise train seti oluyor.
- Modelin performansı, her iterasyondaki validation setine göre ölçülüyor.
- En iyi performans gösteren modeli seçiyorsunuz

## Bias(sapma) – Varyans(değişkenlik)

**Bias:** Modelin sistematik olarak hatalı tahminler yapmasına neden olan bir hata türüdür. Modelin, gerçek veriyi yeterince iyi temsil edememesinden kaynaklanır ve genellikle **aşırı basitleştirme** durumunda ortaya çıkar.

**Varyans:** bir modelin eğitim verisindeki küçük değişikliklere ne kadar duyarlı olduğunu ifade eder. Yüksek varyansa sahip bir model, eğitim verisinde çok iyi sonuçlar verirken, farklı bir veri kümesiyle karşılaştığında performansı önemli ölçüde düşer. Modelin eğitim verisindeki **değişikliklere duyarlılığını** ve yeni verilerde ne kadar iyi performans göstereceğini ifade eden bir terimdir.

### Bias:

- **Yüksek bias**, modelin verideki karmaşık yapıları göz ardı ederek basit bir tahmin yöntemi kullandığını ve bu nedenle düşük performans gösterdiğini ifade eder. Bu, **eksik uyuma (underfitting)** neden olur, çünkü model veriye yeterince iyi uyum sağlamaz.
- **Düşük bias** durumunda ise model, veriye daha iyi uyum sağlar ve daha doğru tahminler yapar.

### Varyans:

- **Yüksek Varyans:** Model, eğitim verisine aşırı uyum sağlamıştır ve eğitim verisindeki küçük hataları ve ayrıntıları bile öğrenmiştir. Bu durum **aşırı öğrenme (overfitting)** olarak bilinir. Yani, model eğitim verisine fazlasıyla bağlıdır ve test verisi üzerinde düşük performans gösterir.
- **Düşük Varyans:** Model eğitim verisini iyi genelleyebilir ve yeni verilere daha sağlam bir şekilde uyum sağlar. Aşırı karmaşıklıktan kaçınır, ancak bu kez karmaşık desenleri yakalayamayabilir.

## Örnek:

Bir doğrusal regresyon modeli, çok karmaşık bir ilişkisi olan veriyi basit bir düz çizgiyle tahmin etmeye çalışırsa, **yüksek bias** sergiler çünkü gerçekte var olan ilişkiyi yakalayamaz. Bu, modelin eksik uyum göstermesine neden olur.

## Bias-Variance Dengesizliği:

Bias, **varyans** ile birlikte modelin hatasına katkıda bulunur. İyi bir model geliştirmek için **bias-varyans dengesi** sağlanmalıdır.

- **Yüksek bias:** Model fazla basit ve genelleme yeteneği zayıf.
- **Yüksek varyans:** Model çok karmaşık, veriye aşırı uyum sağlamış (aşırı uyum – overfitting).

📊 **Underfitting (Yüksek Bias):** Model veriyi yeterince öğrenememiştir ve genelleme yeteneği zayıftır.

📊 **Overfitting (Yüksek Varyans):** Model veriye aşırı uyum sağlamıştır, eğitim verisindeki hataları bile öğrenmiştir, bu yüzden yeni verilere genelleme yapamaz.

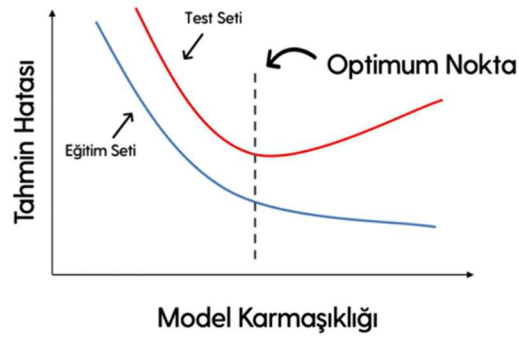
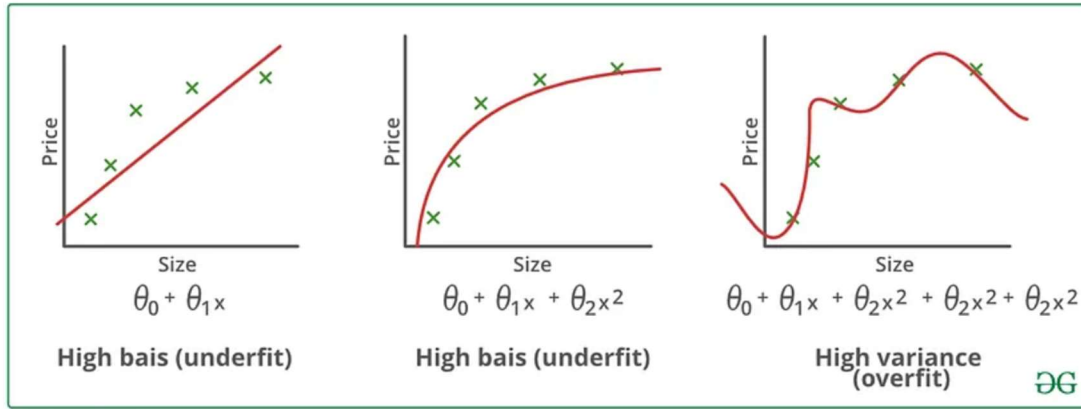
İyi bir model geliştirmede amaç, **bias ve varyansı dengeleyerek** hem eğitim hem de test verilerinde iyi performans göstermesini sağlamaktır.

- Bias: Modelin veriyi **basitleştirerek** kötü tahmin yapması.
- Varyans: Modelin veriyi **aşırı karmaşıktırarak** gürültüyü bile öğrenmesi.

İdeal bir modelde hem **düşük bias** hem de **düşük varyans** istenir. **Düşük bias ve düşük varyans** dengesini bulmak. Bu denge sayesinde model, hem eğitim verisini iyi öğrenir hem de yeni verilere genelleme yapabilir.

**Bias** ve **varyans** arasındaki dengeyi sağlamak ve doğru model seçimi yapmak için **ortalama test hatası** (MSE) temel alınır. Genellikle modelin performansını değerlendirirken, hem **eğitim hatası** hem de **test hatası** dikkate alınır, ancak **test hatası** daha önemli bir ölçüttür, çünkü modelin genelleme yeteneğini gösterir.

## Yanlılık Varyans Değiş Tokuşu (Bias Variance Tradeoff )



Bu grafiğe learning curve (öğrenme eğrisi) grafiği denir. Burada eğitim işleminde nerede durmamız gerektiğine bulduğumuz optimum nokta ile karar veririz.

Model karmaşıklığı: hiperparametreleri optimize etmek, yeni değişkenler eklemek gibi etkiler model karmaşıklığını artırır.

Eğitim setinin model karmaşıklığı arttıkça test setinde tahmin hatası bir noktaya kadar azalır bu da düşük hata gösterdiğini ve model başarısının yüksek olduğunu gösterir. Bir noktadan sonra bu model karmaşıklığına negatif olarak yansır ve tahmin hatası artmaya başlar bu da model başarısını azaltır.

Bir algoritmasını eğittiğimizde modelin her yenilenmesinde ne kadar iyi performans gösterdiğini ölçebiliriz. Belirli sayıda yenilemeye kadar yeni yenilemeler modeli iyileştirir. Ancak daha sonrasında model eğitim verilerini aşırı doldurmaya başladığında eskisi gibi olmayabilir. Bunun için yapılması gereken bu noktayı geçmeden eğitimi durdurmaktır. Bu noktadaki çözüm ise karmaşıklığı azaltmaktır.



## F-statistic - Prob (F-statistic)

### 1. F-statistic

- **F-statistic**, modeldeki **bağımsız değişkenlerin topluca bağımlı değişken üzerinde anlamlı bir etkisi olup olmadığını** test eder.
- Basitçe, modeldeki bağımsız değişkenlerin (X'ler) bir anlam ifade edip etmediğini gösterir. Yani, bu değişkenlerin bağımlı değişkeni (y) açıklamada başarılı olup olmadığını sorgular.
- **F-statistic** değeri ne kadar yüksekse, modeldeki bağımsız değişkenlerin topluca bağımlı değişken üzerinde anlamlı bir etkisi olduğu ihtimali o kadar yüksektir.

### 2. Prob (F-statistic)

- **Prob (F-statistic)**, bu F-testinin **p-değeri**dir. F-testinin sonucunun **rastgele elde edilme olasılığını** gösterir.
- Genelde "p-değeri" ile ifade edilir ve bu değerin küçük olması, modelin anlamlı olduğunu gösterir.
  - Eğer **Prob (F-statistic) değeri** 0.05'ten küçükse, genellikle modeldeki bağımsız değişkenlerin **istatistiksel olarak anlamlı** olduğu kabul edilir. Bu, bağımsız değişkenlerin topluca bağımlı değişkeni açıklamada etkili olduğuna işaret eder.
  - Eğer **Prob (F-statistic) değeri** büyükse (0.05'ten büyük), modelin anlamlı olmayabileceği anlamına gelir, yani bağımsız değişkenler bağımlı değişkeni yeterince açıklayamıyor olabilir.

### Özet:

- **F-statistic**: Modeldeki bağımsız değişkenlerin topluca anlamlı olup olmadığını test eden istatistik.
- **Prob (F-statistic)**: Bu testin p-değeri; genelde 0.05'ten küçükse, modelin anlamlı olduğunu gösterir.

## t-testi ve Anlamı:

- **t-testi**, her bir bağımsız değişkenin katsayısının sıfır olup olmadığını test eder. Yani, o değişkenin bağımlı değişken üzerinde anlamlı bir etkisi var mı yok mu bunu sorgular.
- **t-değeri** ne kadar büyükse, o bağımsız değişkenin katsayısının sıfır olma olasılığı o kadar düşük olur. Bu da o değişkenin bağımlı değişken üzerinde **anlamlı** bir etkisi olduğunu gösterir.

## Prob (t-statistic) veya p-değeri:

- **p-değeri** ( $\text{Prob} > |t|$ ) genelde t-testiyle birlikte gösterilir ve bu değerin anlamlı olup olmadığını belirler.
  - Eğer p-değeri **0.05'ten küçükse**, o bağımsız değişkenin katsayısı istatistiksel olarak anlamlı kabul edilir.
  - Eğer p-değeri **0.05'ten büyükse**, o bağımsız değişkenin bağımlı değişken üzerinde anlamlı bir etkisi olmadığı düşünülür. Bu durumda, katsayı sıfıra yakın olabilir.

## Özetle:

- **t-testi** değeri, her bir bağımsız değişkenin katsayısının anlamlı olup olmadığını gösterir.
- Eğer t-testi yüksekse ve **p-değeri 0.05'in altındaysa**, o değişken bağımlı değişken üzerinde anlamlı bir etkiye sahiptir.
- Eğer **p-değeri yüksekse** (0.05'ten büyükse), o değişken modelde anlamlı bir katkı sağlamıyor olabilir ve çıkarılabilir.

## Coefficients

Bağımsız değişkenlerin bağımlı değişken üzerindeki etkisini gösterir. Bu, bir bağımsız değişkende (X) meydana gelen 1 birimlik değişimin, bağımlı değişkeni (y) nasıl etkilediğini ölçen sayısal bir değerdir.

Regresyon modelinde her bağımsız değişkenin (X) bir **katsayısı (coef)** vardır. Bu **katsayı**, bağımsız değişkenin 1 birim artışında, bağımlı değişkende (y) ne kadar bir artış veya azalış olacağını belirtir.

Örneğin: Eğer bir bağımsız değişkenin katsayısı **2** ise, o bağımsız değişken **1 birim arttığında** bağımlı değişken **2 birim artar**. Eğer bir bağımsız değişkenin katsayısı **-3** ise, o bağımsız değişken **1 birim arttığında** bağımlı değişken **3 birim azalır**. Bu sayede, bağımsız değişkenlerin modelde bağımlı değişkeni nasıl etkilediğini anlayabiliriz.

## Hold-out yöntemi (veri ayırma yöntemi)


Makine öğrenimi modelinin eğitim ve test süreçlerini ayırarak, modelin doğruluğunu ve genel performansını değerlendirmek için kullanılan basit bir yöntemdir.

Hold-out yöntemi, modelin genel başarısını ve genelleme yeteneğini değerlendirmek, aşırı öğrenme sorunlarını tespit etmek, hiperparametre ayarlaması yapmak ve farklı modelleri karşılaştırmak için kullanılır. Bu yöntem, modelin sağlam bir şekilde çalıştığını ve yeni verilere uygun olduğunu garanti etmeye yardımcı olur.

`model = lm.fit(X_train, y_train)` => train set kullanılarak bağımlı değişkenler, bağımsız değişkenler üzerinde fit edilir. Model, bağımsız değişkenlerin bağımlı değişken üzerindeki etkisini öğrenerek tahmin yapma yeteneğini geliştirir.


`model.predict(X_test)` => Modeli eğittikten sonra, test setini kullanarak tahminlerde bulunuruz. Bu, modelin daha önce görmediği veriler üzerinde nasıl performans gösterdiğini değerlendiririz. `X_test`'i kullanarak `y_predict` tahmin edilir. Yani model, `X_test` verilerine dayalı olarak bağımlı değişkenin tahmin edilen değerlerini (`y_pred`) üretir.

Elde edilen `y_pred` değerleri, gerçek bağımlı değişken değerleri olan `y_test` ile karşılaştırılarak modelin performansı değerlendirilir. Bu değerlendirme, çeşitli metrikler (örneğin,  $R^2$ , MSE, RMSE gibi) kullanılarak yapılır.

 **mse = mean\_squared\_error(y\_train, model.predict(X\_train))**

Bu adımda, modeli eğittiğimiz **train seti** üzerinde tahminler yapıyoruz ve ardından bu tahminlerle gerçek train seti değerleri (`y_train`) arasındaki farkları (MSE) hesaplıyoruz.

- **Amaç:** Modelin, eğitim verisine ne kadar iyi uyum sağladığını görmek.
- **Risk:** Eğitim setinde hata düşük olabilir çünkü model bu verilerle eğitildi. Ancak bu, modelin genel anlamda iyi performans gösterdiği anlamına gelmez. Model fazla uyum sağlamış (**overfitting**) olabilir.

 **mse = mean\_squared\_error(y\_test, model.predict(X\_test))**

Bu adımda, modeli daha önce hiç görmediği **test seti** üzerinde test ediyoruz. Burada yapılan işlem, test setiyle modelin gerçek dünyada nasıl performans göstereceğini anlamak.

- **Amaç:** Modelin genelleme yeteneğini ölçmek. Yani model, yeni ve daha önce görmediği verilerle ne kadar iyi performans gösteriyor?
- **Gerçek Değer:** Test seti MSE, modelin gerçek dünyadaki performansına daha yakın bir ölçümdür. Eğer test seti üzerindeki hata çok büyükse, modelin genelleme yeteneği zayıf olabilir.

Eğer iki hata arasında büyük bir fark varsa (örneğin train hatası çok düşük, test hatası çok yüksek), bu durumda model muhtemelen overfitting yapmıştır.

## K-fold cross validation

```
from sklearn.model_selection import cross_val_score
```

```
cross_val_score(model, X, y, cv=10, scoring="neg_mean_squared_error")
```

- **model:** Burada eğitim için kullandığımız model (LinearRegression gibi) verilmekte.
- **X ve y:** Bağımsız değişkenler (özellikler, X) ve bağımlı değişkenler (hedef, y). Bu veriler modelin tahmin etmeyi öğrendiği verilerdir.
- **cv=10:** Bu parametre, **10 katlı çapraz doğrulama** yapılacağını belirtiyor. Yani veriler, her bir seferde %90'ı eğitim, %10'u test olmak üzere 10 farklı parçaya bölünür. Her bir parça test seti olarak bir kez kullanılır ve diğer parçalar eğitim için kullanılır. Bu, toplamda 10 farklı model eğitileceği anlamına gelir.
- **scoring="neg\_mean\_squared\_error":** Hata ölçümünü belirtiyor. Burada, modelin performansını **Mean Squared Error (MSE)** kullanarak ölçüyoruz.
- **cross\_val\_score** fonksiyonu hata metriklerini minimize etmek yerine maksimize etmeye çalışır, bu nedenle negatif bir MSE döndürür. Eğer pozitif bir MSE değeri istiyorsanız, sonuçların negatifini almanız gerekir.
- **Cross-validation**, modelin daha genel bir değerlendirmesini sağlar ve overfitting/underfitting risklerini azaltır.
- **Maksimize Etme Amacı:** cross\_val\_score, verilen bir skorumu kriterini en yüksek değere ulaşmayı hedefler. Yani modelin ne kadar iyi olduğunu değerlendirmek için kullanılan herhangi bir metriği (örneğin accuracy, F1 skoru, vb.) artırmaya çalışır. Hata metrikleri genellikle daha düşük değerler (örneğin, MSE, RMSE) ile daha iyi sonuçlanır, bu nedenle bu tür metrikler negatif işaretli döner.

## Ridge regression - L2 Regularization

Ridge regresyon, doğrusal regresyon modelinin bir türüdür ve aşırı öğrenmeyi (overfitting) önlemek için kullanılır. Doğrusal regresyon, bağımlı değişken ile bağımsız değişkenler arasındaki ilişkiyi anlamak için bir çizgi çizer, ancak verilerdeki gürültüden (noise) etkilenebilir. Ridge regresyon, modeli daha basit hale getirerek bu durumu düzeltir.

Ridge regresyon, regresyon katsayılarını cezalandırarak aşırı öğrenmeyi önler. Normal doğrusal regresyonda hata fonksiyonu sadece tahmin hatalarını minimize etmeye çalışırken, ridge regresyon bu hatalara ek olarak katsayıların büyüklüğünü de minimize eder. Matematiksel olarak, hata fonksiyonuna bir **L2 ceza terimi** ekler.

- ✚ Amaç hata kareler toplamını minimize eden katsayıları, bu katsayıları bir ceza uygulayarak bulmaktır. Over-fittinge karşı dirençlidir. Çok boyutluluğa çözüm sunar. Tüm değişkenler ile model kurar, ilgisiz değişkenleri çıkarmaz sadece katsayılarını sıfıra yaklaştırır. Modeli kurarken alpha (ceza) için iyi bir değer bulmak gerekir, bunun için cross validation yöntemi kullanılır.

$$RSS_{L_2} = \sum_{i=1}^n (Y_i - \hat{Y}_i)^2 + \lambda \sum_{j=1}^P B_j^2$$

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

$$SSE_{L_2} = \sum_{i=1}^n (y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^P \beta_j^2$$

Ayar Parametresi Lambda      Ceza Terimi

$\lambda$ : Ayar parametresi, kullanıcı tarafından ayarlanması ve optimize edilmesi gerekir. (hiperparametre)

$\lambda = 0$  iken, Ordinary Least Squares (OLS)'i elde ederiz. **Sum of Squared Errors (SSE) (Hata kareler toplamı)**'ni minimum yapan  $\lambda$ 'yı arıyoruz.

$\lambda$ 'nın belirli değerleri içeren bir küme seçilir ve her biri için cross validation test hatası hesaplanır. En küçük cross validationı veren  $\lambda$ , ayar parametresi olarak seçilir. yani farklı  $\lambda$  değerleri ile farklı katsayılar oluşacak ve bu katsayılar ile kurulan modeller (seçilen  $\lambda$  ile model yeniden tüm gözlemlerle fit edilir.) ve bu modellerin hatalarını incelenecek buna göre optimum  $\lambda$  ya karar verilecek. Bu katsayılar öyle bir yerde durmalı ki RMSE değeri min olsun.

#### Rezidü (Hata, $e_i$ ):

Rezidü, **gözlem değerleri ile tahmin edilen değerler arasındaki farktır**. Bu fark, modelin ne kadar iyi tahmin yapabildiğini gösterir. Yani modelin hatası olarak da düşünülebilir. Rezidü şu şekilde hesaplanır:

$$e_i = y - \hat{y}$$

### pd.get\_dummies - L2 Regularization:

**Dummy Variables** (sahte değişkenler): Kategorik değişkenlerin sayısal verilere dönüştürülmesi için kullanılan bir teknik olan "dummy encoding" ya da "one-hot encoding" yöntemidir. Bu teknik, her bir kategoriye ayrı bir sütun olarak temsil eder ve o sütunda 1 veya 0 değerleri kullanarak kategorinin varlığını belirtir.

- **pd.get\_dummies ()**: Pandas kütüphanesinde bir fonksiyon.
- **One-Hot Encoding**: pd.get\_dummies () ile gerçekleştirilen işlemin adıdır.

```
dms = pd.get_dummies(df[["League","Division","NewLeague"]])
```

	League_A	League_N	Division_E	Division_W	NewLeague_A	NewLeague_N	
0	True	False	True	False	True	False	League : A, N
1	False	True	False	True	False	True	Division: E, W
2	True	False	False	True	True	False	NewLeague: A, N
3	False	True	True	False	False	True	
4	False	True	True	False	False	True	

Her bir sütundaki kategorik verileri ayrı sütunlar halinde yapıyor. Böylece istediğimiz sütunları ayrı ayrı alabiliyoruz.

### Model tuning:

**RidgeCV**, fonksiyonu, Ridge regresyonu için cross-validation (CV) ile birlikte en uygun alpha değerini otomatik olarak seçen bir modeldir. (Ridge regresyonunun cross-validation versiyonudur)

**RidgeCV (lambdas = lambdas2, scoring = "neg\_mean\_squared\_error", cv=10)**

### Losso Regression – L1 Regression

Amaç hata kareler toplamını minimize eden katsayıları, bu katsayıları bir ceza uygulayarak bulmaktır.

$$SSE_{L_1} = \sum_{i=1}^n (y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^P |\beta_j|$$

Ayar Parametresi Lambda Ceza Terimi

Ridge'den farkı: yeteri kadar büyük lambda değerleri gelirse bu katsayıları sıfıra yaklaştırır ve sıfır yapar.

- ✚ Ridge regresyonun ilgili ilgisiz tüm değişkenleri modelde bırakma dezavantajını gidermek için vardır.
- ✚ Lasso'da katsayılar sıfıra yaklaştırılır ve sıfır olur böylece değişken seçimi işlemi gerçekleştirilir.
- ✚ L1 normu  $\lambda$  yeteri kadar büyük olduğunda bazı katsayıları sıfır yapar, L2 normunda sıfır yapmıyordu sıfıra yaklaşıyordu.
- ✚ Burada da cross validation kullanılır.
- ✚ Bu iki yöntem birbirinden üstün değildir
  - $\lambda$  nın sıfır olduğu yer OLS dir. MSE(OLS/n) yi min yapan değeri arıyoruz.
  - $\lambda$  için belirli değerler içeren küme seçilir ve her birisi için cross validation test hatası hesaplanır.
  - En düşük cross validationı veren  $\lambda$  değeri ayar parametresi olarak seçilir
  - Son olarak seçilen  $\lambda$  değeri ile model yeniden fit edilir

**Not:** R<sup>2</sup> score - modelin açıklanabilirliğini ifade eder. Bağımsız değişkenin, bağımlı değişkeni ne kadar iyi açıkladığını gösterir (açıklama yüzdesidir).

- 0.0: Model, verideki değişkenliği hiç açıklayamıyor.
- 1.0: Model, verideki değişkenliğin tamamını açıklıyor.

- $R^2 \approx 0.7$  ve üzeri: Model verideki değişkenliğin büyük kısmını açıklıyor, genellikle iyi bir performans göstergesidir.
- $R^2 \approx 0.5 - 0.7$ : Orta düzeyde bir performans. Model verinin yaklaşık yarısını açıklıyor.
- $R^2 < 0.5$ : Model verideki değişkenliğin yarısından azını açıklıyor. Bu, genellikle modelin geliştirilmeye ihtiyaç duyduğunu gösterir.

## ElasticNet Regression = L1 + L2

Amaç hata kareler toplamını minimize eden katsayıları, bu katsayıları bir ceza uygulayarak bulmaktır. Ridge ve lasso regresyonunun toplamıdır. Daha etkin bir düzgünleştirme işlemi yapar.

Ridge tarzı cezalandırma, Lasso tarzı değişken seçimi yapar.

L2 ve L1 Ayar Parametreleri

$$SSE_{Enet} = \sum_{i=1}^n (y_i - \hat{y}_i)^2 + \boxed{\lambda_1} \sum_{j=1}^P \beta_j^2 + \boxed{\lambda_2} \sum_{j=1}^P |\beta_j|$$

Ceza Terimleri

Lasso ve ridge tekniklerinin birleştiriyor, bu iki tekniğin fonksiyonel yapı üzerindeki etkilerini kontrol eden parametre: `l1_ratio` parametresidir. Default olarak 0.5 e ayarlıdır.

ElasticNet, **Lasso** (L1 regülarizasyonu) ve **Ridge** (L2 regülarizasyonu) regresyon tekniklerini birleştiren bir yöntemdir. Bu iki yöntemin bir arada kullanılmasını sağlayan **l1\_ratio** parametresi, hangi regülarizasyon türünün modele daha fazla katkıda bulunacağını kontrol eder.

Örn: **l1\_ratio=0.7**, Lasso'ya %70, Ridge'ye %30 ağırlık verir. **alpha** ise her iki regülarizasyonun gücünü kontrol eden parametredir.

- **l1\_ratio = 0**: Sadece **Ridge** (L2 regülarizasyonu) uygulanır. Bu durumda model tamamen Ridge regresyonu gibi davranır.
- **l1\_ratio = 1**: Sadece **Lasso** (L1 regülarizasyonu) uygulanır. Bu durumda model tamamen Lasso regresyonu gibi davranır.



## Çoklu doğrusal bağlantı problemi nedir

**Çoklu doğrusal bağlantı problemi**, bir regresyon modelinde bağımsız değişkenler arasında yüksek derecede korelasyon olduğunda ortaya çıkar. Bu durum, bağımsız değişkenlerden en az birinin diğer bağımsız değişkenlerle lineer bir ilişkiye sahip olduğu anlamına gelir.

- **Korelasyon:** Çoklu doğrusal bağlantı, bağımsız değişkenlerin birbirleriyle güçlü bir şekilde ilişkilendirilmesi anlamına gelir. Örneğin, iki bağımsız değişken arasında yüksek bir Pearson korelasyonu varsa, bu çoklu doğrusal bağlantı problemi olabilir.

- **Tahmin Hataları:** Çoklu doğrusal bağlantı, regresyon modelinin katsayı tahminlerinin güvenilirliğini azaltabilir. Bu durumda, bazı katsayıların standart hataları büyük olur, bu da istatistiksel olarak anlamlı sonuçlar elde etmeyi zorlaştırır.

## NonLinear regression

Diğer regresyon problemlerinde olduğu gibi doğrusal olmayan regresyon çeşitlerinde de bağımlı değişkenimiz sayısal bir değişken.

- K-En Yakın Komşu
- Destek Vektör Regresyonu
- Yapay Sinir Ağları
- CART
- Random Forests
- Gradient Boosting Machines
- XGBoost
- LightGBM
- CatBoost

## K-nearest neighbors

Gözlemlerin birbirine olan benzerlikleri üzerinden tahmin yapılır.

Parametrik olmayan bir öğrenme türüdür.

Classification veya regression problemlerinde kullanılabilir.

**GridSearchCV:** bir modelin hiperparametrelerini optimize etmek için kullanılan bir yöntemdir. Birbirinden farklı ve fazla sayıda parametre olduğunda onları bir grid(ızgara) mantığı ile değerlendirip farklı hiperparametre kombinasyonlarını denemek ve en iyi sonucu veren kombinasyonu bulmak için kullanılır. Bu işlem, çapraz doğrulama (cross-validation) ile birlikte yapılır, bu sayede modelin doğrulama seti üzerindeki performansı değerlendirilir.

```
knn_cv_model = GridSearchCV(model, knn_params, cv=10).fit(X_train, y_train)
```

(Aranmasını istediğimiz parametreleri bir sözlük içerisinde ifade ediyoruz)

●  $\curvearrowright$  Y tahmini nedir?

$X1 = 50, X2 = 230$

Bağımsız değişken değerleri verilen gözlem biriminin bağımlı değişken değeri olan "Y"sini tahmin etmek için ilgili gözlem birimlerinin tablodaki diğer gözlem birimleriyle olan benzerlikleri hesaplanacaktır.

Y	X1	X2
100	56	241
120	85	250
150	25	233
.	.	.
.	.	.
140	56	231

Ve bu benzerlikler üzerinden ilgili gözlem birimimize tablodaki gözlem birimlerinden hangisi en yakın ise ilgili gözlem biriminin bağımlı değişkeni olan "Y" tahmin edilmiş olacaktır.

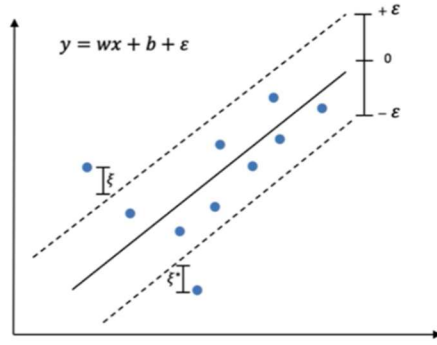
- Komşu sayısını belirle (K)
- Bilinmeyen nokta ile diğer tüm noktalar ile arasındaki uzaklıkları hesapla
- Uzaklıkları sırala ve belirlenen k sayısına göre en yakın olan k gözlemi seç
- Sınıflandırma ise en sık sınıf, regresyon ise ortalama değeri tahmin değeri olarak ver.

## Support Vector Regression (SVR)

Güçlü ve esnek modelleme tekniklerinden biridir.

Classification ve regression alanında kullanılabilir.

Robust(dayanıklı) bir regression tekniğidir. (Robust, aykırı gözlem değerlerine karşı dayanıklı)



Minimizasyon Problemi:

$$\frac{1}{2} \|w\|^2 + C \sum_{i=1}^m (\xi_i + \xi_i^*)$$

Kısıtlar:

$$y_i - (w * x_i) - b \leq \varepsilon + \xi_i$$

$$(w * x_i) + b - y_i \leq \varepsilon + \xi_i^*$$

$$\xi_i, \xi_i^* \geq 0, \quad i = 1, \dots, m$$

SVR, özellikle hataların çok büyük öneme sahip olmadığı, ancak tahmin edilen değerlerin genel olarak doğru bir trende uyması gerektiği durumlarda kullanılır.

Kısıtlar der ki: öyle bir regresyon doğrusu bulmana yardım edeceğim ki gerçek değerler ile tahmin edilen değerler arasındaki farklar regresyon eğrisinin iki yönünden belirli bir epsülon ve kısı değerinden daha uzakta olmayacak.

`svr_cv_model = GridSearchCV(svr_model, svr_params, cv=5, verbose=2, n_jobs=-1).fit(X_train, y_train)`

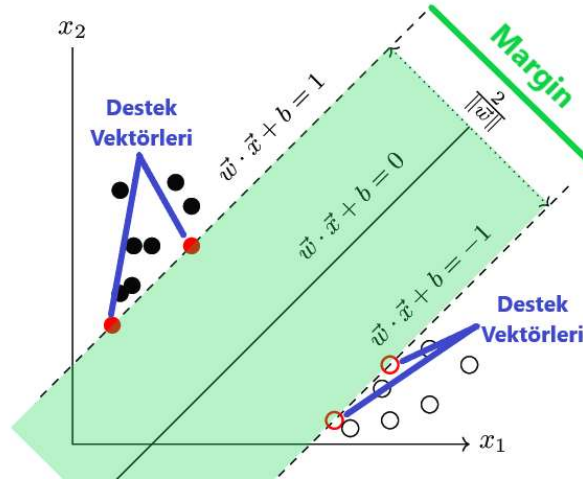
**verbose=2:**

- **verbose**, fonksiyonun çalışma sürecinde ne kadar bilgi yazdıracağını belirler.
- **verbose=2**, her bir katlamada (**fold**) yapılan işlemler hakkında detaylı bilgi verir. Bu, özellikle modelin eğitim sürecini takip etmek için kullanışlıdır. Daha büyük verbose değerleri daha fazla ayrıntı sağlar, örneğin:
  - **verbose=0**: Sessiz mod. Hiçbir çıktı yazdırılmaz.
  - **verbose=1**: Temel bilgiler yazdırılır (örneğin, hangi hiperparametrelerin denendiği gibi).
  - **verbose=2**: Daha ayrıntılı bilgiler verilir, her bir katlama için bilgi sağlar.

**n\_jobs=-1:**

- **n\_jobs**, grid search'ün kaç işlemci çekirdeği (CPU core) kullanarak paralel olarak çalışacağını belirler. Bu parametre, modeli hızlandırmak için önemlidir.
- **n\_jobs=-1** ayarı, tüm kullanılabilir işlemci çekirdeklerinin kullanılmasını sağlar. Yani bilgisayarınızda kaç çekirdek varsa (örneğin 8 çekirdek), hepsi paralel olarak grid search işlemlerini gerçekleştirmek için kullanılır.
  - **n\_jobs=1**: Tek işlemci çekirdeği kullanılır (varsayılan).
  - **n\_jobs=2**: 2 çekirdek kullanılır.
  - **n\_jobs=-1**: Tüm çekirdekler kullanılır (maksimum hız).

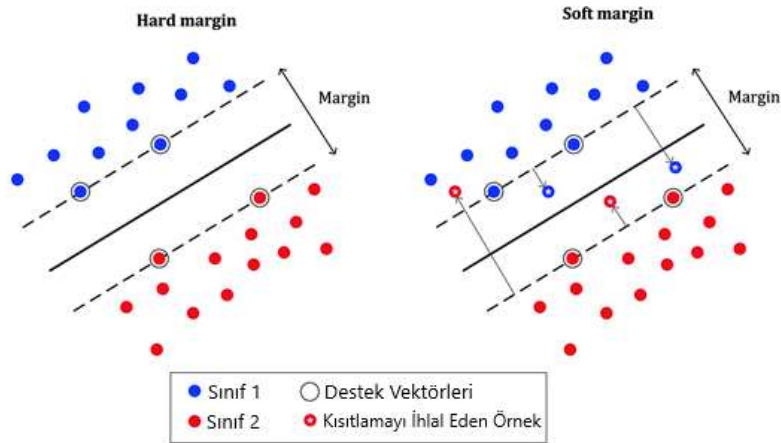
# SVM



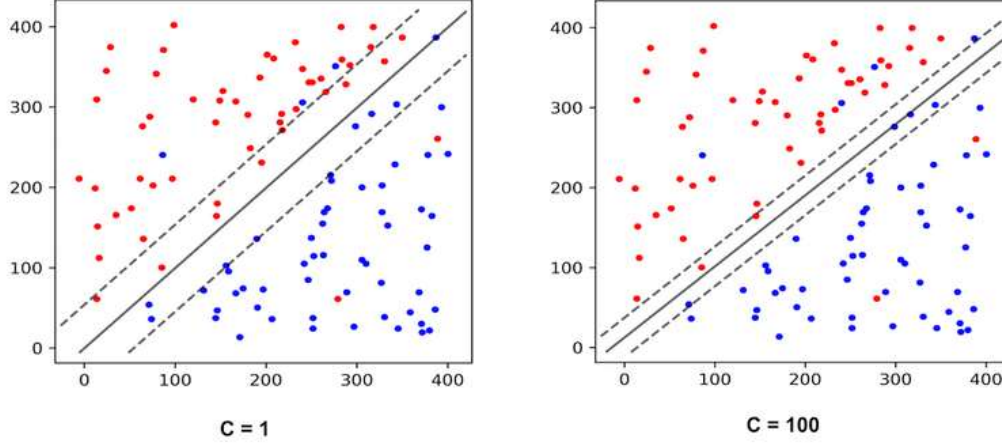
Tabloda siyahlar ve beyazlar olmak üzere iki farklı sınıf var. Sınıflandırma problemlerindeki asıl amacımız gelecek verinin hangi sınıfta yer alacağını karar vermektir. Bu sınıflandırmayı yapabilmek için iki sınıfı ayıran bir doğru çizilir ve bu doğrunun  $\pm 1$ 'i arasında kalan yeşil bölgeye Margin adı verilir. Margin ne kadar geniş ise iki veya daha fazla sınıf o kadar iyi ayrıştırılır.

## Hard Margin vs Soft Margin

Marginimiz her zaman bu şekilde olmayabilir. Bazen örneklerimiz Margin bölgesine girebilir. Buna Soft Margin denir. Hard Margin, verimiz doğrusal olarak ayrılabilir çalışır ve aykırı değerlere karşı çok duyarlıdır. Bu yüzden bazı durumlarda Soft Margin'i tercih etmemiz gerekebilir.



İkisi arasındaki dengeyi SVM içerisindeki C hiperparametresi ile kontrol edebiliriz. C ne kadar büyükse Margin o kadar dardır. Ayrıca model overfit olursa C'yi azaltmamız gerekir.



## Yapay sinir ağıları

Amaç en küçük hata ile tahmin yapabilecek katsayılar elde etmektir.

Yapay sinir ağlarında **ağırlık** (weight), bir **nörondan** diğerine aktarılan bilginin önemini ve etkisini ifade eder. Bir nöronun ürettiği sinyaller diğer nöronlara taşınırken, bu sinyaller belirli bir **ağırlık katsayısı** ile çarpılır. Bu katsayı, bilginin ne kadar güçlü veya zayıf bir şekilde iletileceğini belirler.

Daha açık bir şekilde:

- **Pozitif ağırlık:** Bilginin güçlenerek iletilmesine neden olur.
- **Negatif ağırlık:** Bilginin zayıflatılarak veya ters çevrilerek iletilmesine neden olur.
- **Ağırlık değeri ne kadar büyükse,** sinyalin etkisi o kadar fazla olur.

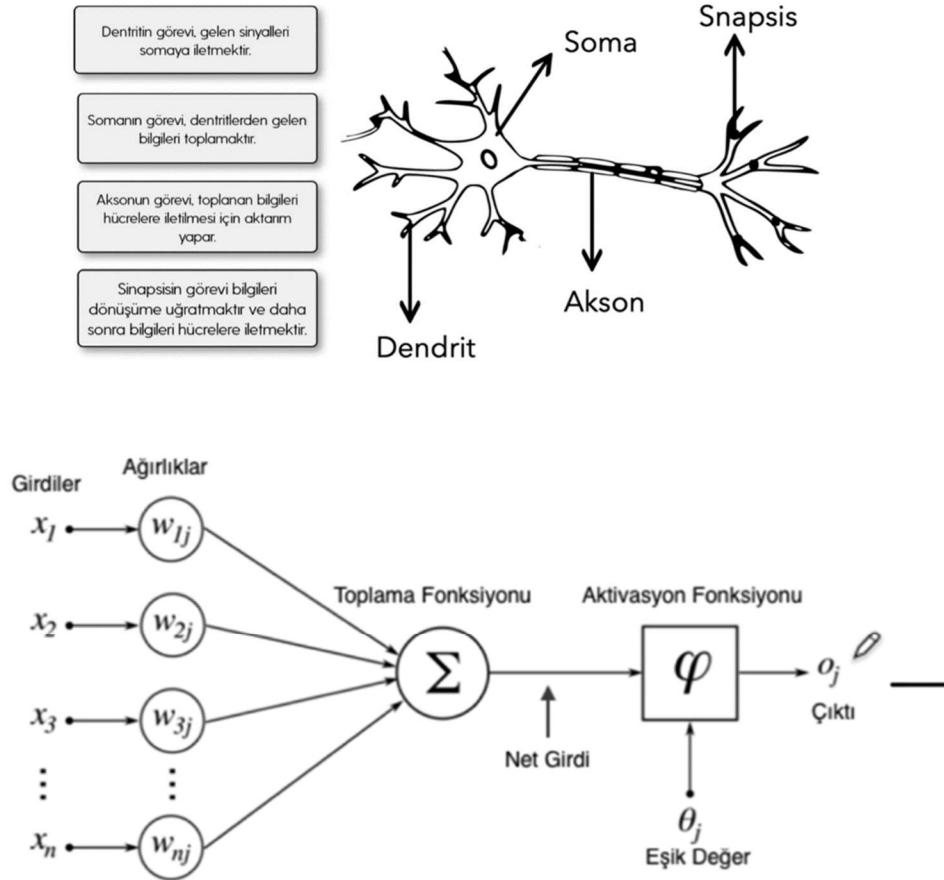
Ağırlıklar, öğrenme sürecinde güncellenir ve ağırlık doğru tahminler yapmasını sağlayan temel parametrelerdir. Eğitim sırasında ağırlıklar optimize edilerek, ağırlık girdi ile çıktı arasındaki ilişkiyi doğru bir şekilde öğrenmesi amaçlanır.

Sinyallerin ağırlık katsayısı ile çarpılması, yapay sinir ağlarının her bağlantıya ne kadar önem verdiğini veya her bilginin **ne kadar değerli** olduğunu belirlemek içindir.

Örneğin, bir yapay sinir ağı, bir görüntüdeki bazı piksellerin diğerlerinden daha önemli olduğunu öğrenebilir. Bu durumda, önemli piksellerin taşıdığı bilgilere daha yüksek ağırlık verecektir.

Ağırlıklar, sinir ağına gelen bilginin önemli veya önemsiz olup olmadığını ayırt etmeye yardımcı olur. Eğer bir girdi çok düşük bir ağırlıkla çarpılırsa, bu girdi çıktıyı çok az etkiler, yani o bilgi önemsizdir. Eğer bir girdi büyük bir ağırlıkla çarpılırsa, bu o bilginin çıktıyı büyük ölçüde etkilediği anlamına gelir.

Ağırlıklar, ağırlık girdilere göre nasıl bir çıktı üreteceğini belirleyen en önemli faktördür. Öğrenme sürecinde sinir ağı, her ağırlığı ayarlayarak, çıktının ne olacağına karar vermeyi öğrenir. Bu şekilde ağı, daha doğru sonuçlar üretmek için girdilerin hangilerinin daha önemli olduğunu öğrenir.



**X:** bağımsız değişkenler

**W:** beta katsayıları

**Aktivasyon fonksiyonu:** yapay sinir ağlarında nöronlar tarafından yapılan toplama işlemi sonucunda elde edilen değeri alıp bu değeri bir sonuca dönüştürmektir. Basitçe, nöronun aktif olup olmayacağına karar verir. Yani nöronun çıktısını sınırlı bir aralığa sıkıştırır ve böylece ağı karmaşık ve doğrusal olmayan problemleri çözmesini sağlar.

- **Toplama işlemi:** Her bir nöron, kendisine gelen sinyallerin ağırlıklarla çarpılıp toplanması sonucu bir değer üretir.
- **Aktivasyon fonksiyonu:** Bu toplama sonucunu alır ve belirli bir **matematiksel dönüşüm** uygular. Bu işlem sonucu, nöronun çıktısının ne olacağını belirler.

## Neden Aktivasyon Fonksiyonu Kullanılır?

- **Doğrusal olmayanlık:** Aktivasyon fonksiyonları, yapay sinir ağlarına **doğrusal olmayan** ilişkileri öğrenme yeteneği kazandırır. Eğer ağ sadece toplama işlemi yapsaydı, sadece doğrusal ilişkileri öğrenebilirdi. Aktivasyon fonksiyonları sayesinde ağ, karmaşık ve doğrusal olmayan problemleri de çözebilir.
- **Çıktıyı sınırlar:** Aktivasyon fonksiyonu, girdiyi belirli bir aralığa sıkıştırır. Örneğin, bazı fonksiyonlar sonucu 0 ile 1 arasında tutar, bazıları ise -1 ile 1 arasında sınırlandırır. Bu, sinir ağının daha kararlı çalışmasını sağlar.

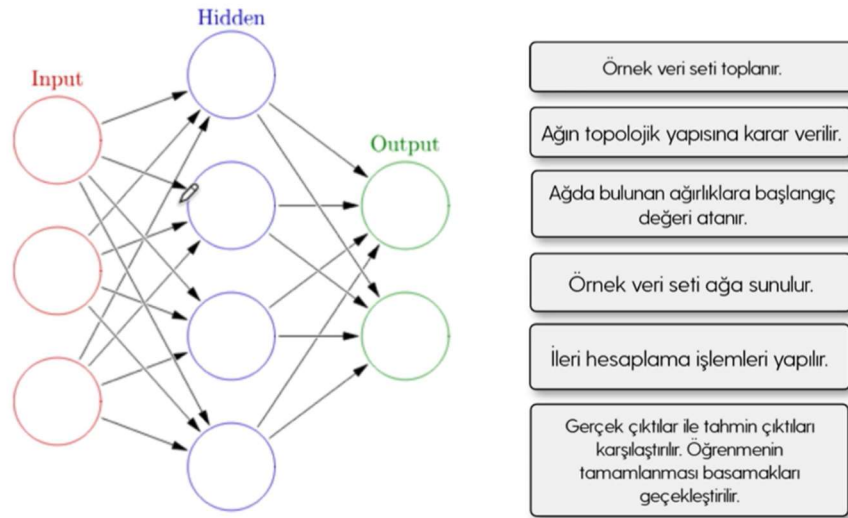
## En Yaygın Aktivasyon Fonksiyonları:

- **Sigmoid:** Sonucu 0 ile 1 arasında sınırlayarak nöronun aktif olup olmayacağını belirler.
- **ReLU (Rectified Linear Unit):** Girdi pozitifse aynen bırakır, negatifse sıfırlar. Özellikle derin öğrenme modellerinde popülerdir.
- **Tanh:** Sonucu -1 ile 1 arasında sınırlar, Sigmoid'e benzer ama çıktı aralığı daha geniştir.

Aktivasyon fonksiyonu olmadan, sinir ağı sadece basit matematiksel ilişkiler öğrenebilir, ancak karmaşık problemleri çözemezdi.

- ReLU, genellikle daha derin ağlarda kullanılır ve hesaplama açısından verimlidir.
- Sigmoid, ikili sınıflandırmada kullanılır.
- Tanh, çıktıyı -1 ile 1 arasında sınırlar ve simetrik çıktı gerektiğinde tercih edilir.

\*\* Tahmin edilen değerler ile gerçek değerler arasındaki farkı minimum yapabilmek adına ağda yer alan katsayılar değiştirilir, bu değişim işlemine teknik olarak ağ öğreniyor adı verilir.



Aktivasyon fonksiyonu işlemi sonrasında buradaki bilgi diğer tüm ağ elemanlarıyla paylaşılır.

Gizli katman sayısı tanımlandığında her bir katman, çıktı ile ilişkili olmalıdır.

$$h_k(x) = g\left(\beta_{0k} + \sum_{j=1}^P x_j \beta_{jk}\right)$$

$$g(u) = \frac{1}{1 + e^{-u}}$$

$$f(x) = \gamma_0 + \sum_{k=1}^H \gamma_k h_k$$

$$\sum_{i=1}^n (y_i - f_i(x))^2 + \lambda \sum_{k=1}^H \sum_{j=0}^P \beta_{jk}^2 + \lambda \sum_{k=0}^H \gamma_k^2$$

Katsayıların başlangıç değeri genelde random olarak atanır. Optimum parametre değerlerini bulmak için çen sık kullanı-ılan yöntemlerden biri Backpropagation yöntemidir. Geri yayılım(Backpropagation), sinir ağlarının öğrenme sürecinin temel mekanizmasını oluşturur ve ağı-ın ağırlıklarını, çıktıların doğruluğunu artırmak için sürekli olarak optimize eder. Bu süreç, ağı-ın daha doğru tahminlerde bulunabilmesini sağlar.

**Delta öğrenme kuralı** (ya da delta kuralı), geri yayılım (backpropagation) algoritmasının bir parçası olarak kullanılabilir ve basit yapay sinir ağlarında ağırlık güncellemeleri için yaygın bir yöntemdir. Delta öğrenme kuralı, hata terimini kullanarak ağırlıkları günceller. Kabul edilebilir bir hata oranına ulaşıncaya kadar gerçek değerler ve tahmin edilen değerler arasındaki farklar minimize edilmeye çalışılır, buna yönelik olarak da girdilerin ağırlıkları değiştirilir.

### verileri standartlaştırmak

Standartlaştırma, verilerin ortalamasını 0 ve standart sapmasını 1 olacak şekilde ölçeklendirme işlemidir. Bu, özelliklerin farklı ölçü birimlerine sahip olduğu veri setlerinde modelin daha iyi performans göstermesine yardımcı olur.

Yapay sinir ağları, homojen veri setleri üzerinde daha iyi çalışan bir algoritmadır. Bu yüzden veri üzerinde standartlaştırma işlemi yaptıktan sonra modeli eğitmek daha sağlıklıdır.



```
scaler = StandardScaler ()  
scaler.fit(X_train)  
X_train_scaled = scaler.transform(X_train)  
X_test_scaled = scaler.transform(X_test)
```

**StandardScaler** sınıfından bir nesne oluşturuluyor.

**fit** metodu, eğitim verisi (**X\_train**) üzerinde çalışır ve veri setinin her bir özelliğinin ortalamasını ve standart sapmasını hesaplar. **Fit işlemi**, modelin verilere nasıl uyum sağladığını öğrenmesi gibidir. Standartlaştırma işlemi için gerekli olan bu bilgileri alır ve saklar.

**transform** metodu, fit ile hesaplanan ortalama ve standart sapma değerlerini kullanarak, **X\_train** verilerini standartlaştırır.

Test verileri üzerinde de aynı dönüşümü uygularız, ancak **fit** işlemi **yalnızca eğitim verileri** üzerinde yapılır.

Yani sadece train seti üzerinde ortalaması veya standart sapma hesaplanır, model eğitilir, test setini model daha önce görmedi bu veriler üzerinden tahmin yapacak. Test seti için sadece transform yapıyoruz çünkü train setindeki veriler mesela 0-1 aralığında ise test setini de bu aralığa getirmemiz gerekiyor. Eğer test seti de aynı ölçeklemeye tabi tutulmazsa, test setindeki veriler eğitim setindeki verilerle uyumsuz hale gelir. Bu durumda model, test verisindeki girdileri eğitimdeki girdilere benzer şekilde göremez ve doğru tahmin yapamaz. İkisini de **ölçekliyoruz**, ancak sadece **train setini fit ediyoruz**.

### **Neden Önce fit, Sonra transform?**

#### **1. fit işlemi:**

- Bu işlem, eğitim verisinin istatistiklerini (ortalama ve standart sapma gibi) öğrenir. Modelin eğitim verisinden öğrenmesi gereken bu bilgilere göre veriyi ölçeklendirmek önemlidir.
- Bu istatistikler, modelin sadece eğitim verisine bakarak veriyi nasıl işleyeceğini belirler.

#### **2. transform işlemi:**

- transform, öğrendiğiniz (fit ettiğiniz) bu ortalama ve standart sapmayı kullanarak veriyi dönüştürür. Yani veriyi aynı dağılımda olacak şekilde yeniden ölçeklendirir.

**Yani öğrenilen standart sapma ve ortalamaya göre veri ölçeklendirilir.**

Model tuning

```
mlp_params = {'alpha': [0.1, 0.01, 0.02, 0.001, 0.0],
```

```
"Hidden_layer_sizes": [(10,20), (5,5), (100,100)]}
```

(10,20) => iki deęer olduęu iin iki katman, birinci katmanda 10, ikinci katmanda 20 hcre var.

## CART – Classification and Regression Tree

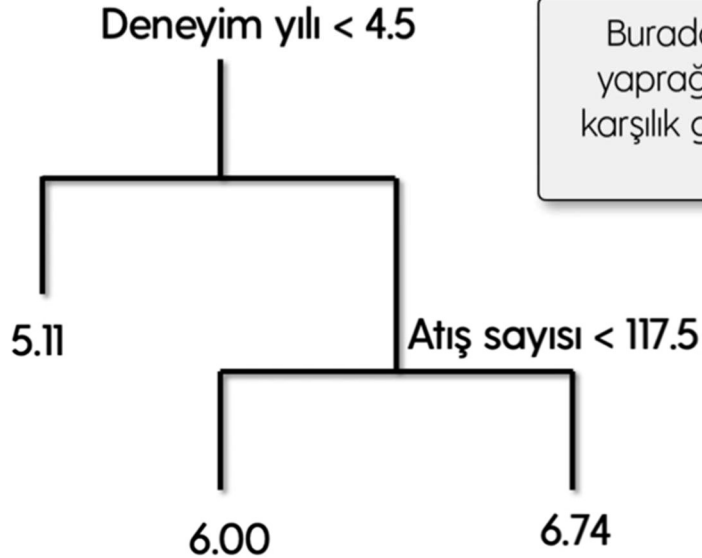
Ama veri seti ierisindeki karmařık yapıları basit karar yapılarına dnřtrmektir.

Heterojen veri setleri belirlenmiř bir hedef deęiřkene gre homojen alt gruplara ayırır.

Karar aęaların (decision tree) yapısını oluřturur.

### nemli Not!

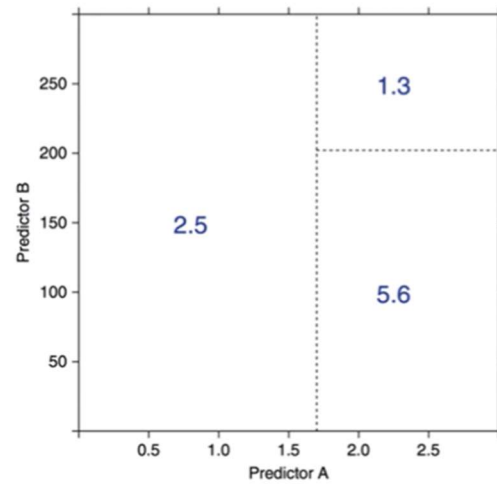
Burada grlen her bir sayı, yapraęın bulunduęu blgeye karřılık gelen yanıt deęiřkeninin ortalamasıdır.



Oyuncunun daha az tecrbeli olduęu bilindięinde, nceki yılda yaptıęı atıř sayısının maařı etkilemedięi grlmektedir.



```
if Predictor A >= 1.7 then  
|   if Predictor B >= 202.1 then Outcome = 1.3  
|   else Outcome = 5.6  
else Outcome = 2.5
```



Elimizdeki bağımlı ve bağımsız değişkenleri bir ağaç yapısı ile gösteriyoruz, bu şekilde bağımsız değişkenler önem yapısına göre dallara ayrılarak, birbirleri ile kesiştirilerek bağımlı değişkenin değerlerini tahmin eder.

**Decision Tree (Karar Ağacı):** sınıflandırma ve regresyon problemlerini çözmek için kullanılan, dallara ayrılan yapısıyla karar verme sürecini modelleyen bir makine öğrenmesi algoritmasıdır. Bir ağaç yapısına benzer şekilde, kararlar adım adım verilir ve her bir düğümde belirli bir soruya cevap verilerek dallara ayrılır. Sonuçta, veri setindeki her bir gözlem, ağacın sonuna (yapraklara) ulaşır ve bu yapraklarda bir sınıf veya tahmin değeri verilir.

# Random forests

**Random Forest** tekniđi **Bagging** ve **Random Subspace** yöntemlerinin geliştirilip birleştirilmesiyle oluşturulmuştur.

**1) Bootstrap Yöntemiyle Gözlem Seçimi:** temel olarak, bir veri setindeki gözlemlerden (satırlardan) **rastgele ve tekrar ederek** örnekler seçmeyi ifade eder.

Bootstrap Nasıl Çalışır?

- Diyelim ki 1, 2, 3, 4, 5 gibi 5 gözlemlili bir veri setiniz var.
- Bootstrap örnekleme yaptığınızda şunları seçebilirsiniz: 1, 3, 3, 5, 2. Gördüğünüz gibi bazı gözlemler (örneğin "3") birden fazla kez seçildi, bazı gözlemler ise (örneğin "4") hiç seçilmedi.

Bu yeni oluşan örnekleme **bootstrap örnekleme** diyoruz. Random Forest yönteminde, her bir karar ağacı bu şekilde farklı bootstrap örnekleme ile eğitilir.

**Bootstrap'in Faydası Nedir?**

- **Model çeşitliliđi** sağlar: Her bir karar ağacı farklı bir bootstrap örnekleme ile eğitildiđi için ağaçlar birbirinden farklı olur.
- **Aşırı uyum (overfitting)** engellenir: Tek bir modelin eğitim verisine çok iyi uyması ve aşırı uyum yapması engellenir. Bootstrap sayesinde farklı alt veri kümeleri kullanıldığından, modelin genellenebilirliđi artar.

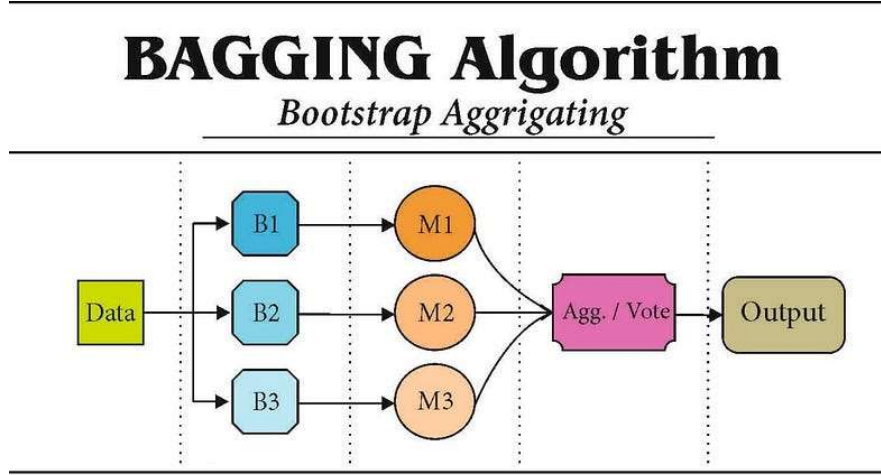
Özetle: Bootstrap, veri setinden rastgele ve tekrar ederek (aynı gözlemin birden fazla kez seçilebileceđi şekilde) gözlemler seçme işlemidir. Random Forest'ta bu yöntem her ağacı farklı bir veri alt kümesiyle eğitmek için kullanılır. **Bootstrap örnekleme**nde **tekrar işlemi (tekrarlı örnekleme)**, yani aynı gözlemin birden fazla kez seçilmesi olasılıđı, yöntemin temel bir parçasıdır ve bu **kesinlikle olmalıdır**.

**2) Rastgele Deđişken Seçimi (Random Subspace):** Normalde karar ağaçları tüm deđişkenleri (veri setindeki sütunlar) kullanarak dallanma kararlarını verir. Ancak Random Forest'ta, her bir karar ağacının her düğümünde (karar noktası) deđişkenlerin sadece bir alt kümesi kullanılır. Örneđin, 100 tane deđişkenin (özelliklerin) bulunduđu bir veri setinde, her bir düğümde sadece rastgele seçilen 20 deđişken arasından en iyi ayırıcı deđişken (örneğin bilgi kazancı gibi bir kritere göre) seçilir.

**Dallanma Kararı:** Karar ağacının her düğümünde, bu rastgele seçilen alt kümedeki deđişkenler üzerinden en iyi ayırıcı deđişken seçilir. Örneđin, veri setinizde 100 deđişken varsa, Random Forest bu 100 deđişken arasından her düğümde 20 tanesini rastgele seçer ve bu 20 deđişkenden en iyi olanı bulur. Bu sayede karar ağaçları aşırı uyum (overfitting) yapmadan daha genellenebilir bir model oluştururlar.

**Rastgelelik ve Genellenebilirlik:** Random Forest yöntemindeki bu rastgelelikler (hem gözlem hem de değişken seçimindeki rastgelelik), modelin genellenebilirliğini artırır. Yani model, sadece eğitim verisine değil, farklı veri setlerine de iyi sonuçlar verir. Çünkü her bir ağaç farklı veri ve değişken alt kümeleri kullanarak eğitilir, bu da ağaçlar arasında çeşitlilik sağlar.

## Bagging: (Bootstrap Aggregating)



**Bagging**, aynı veri seti üzerinde farklı alt kümelerle birçok model eğitmek ve bu modellerin tahminlerini birleştirerek genel performansı artırmak için kullanılan bir yöntemdir. **Random Forest**, Bagging'in karar ağaçları üzerinde uygulandığı bir örnektir.

Bagging yöntemi birçok farklı ağaç yöntemine uygulanabilen bir metodolojidir. M adet gözlem birimi içeren bir veri setinde, n'er adet gözlem ( $n < m$  olacak şekilde) bootstrap örnekleme yöntemi ile seçilerek K tane ağaç ile yeni modeller oluşturulur (Her yeni oluşturulan ağaç önceki ağaçlardan bağımsızdır) ve K tane karar ağacı modelinin ürettiği K adet tahmin değeri bir araya getirilir. **Bagging yöntemi, CART ve birçok tekli ağaç yöntemine göre hata kareler ortalamasını düşüren ve doğru sınıflandırma başarısını artıran ayrıca overfittinge karşı dayanıklı bir yöntemdir.**

Yani farklı veri kümeleriyle eğitilmiş modeller bir arada kullanılır.

*Bagging ile Boosting yöntemlerinin en belirgin farkı, bagging yöntemlerinde ağaçların birbirlerine bağımlılıkları yoktur. Boosting yönteminde ise ağaçlar artıklar üzerine kurulur dolayısıyla ağaçların birbirlerine bağımlılıkları söz konusudur.*

- Rastgele örneklemler çek
- Bu örneklemelerin her birine model kur
- Ve bunları aggrege et yani bir araya getir

## Random Subspace:

Breiman, tıpkı Bagging tekniğinde rastgele gözlem birimleri üzerinden elde ettiği rastgeleliği, değişkenler için de edinmek amacıyla, değişkenlerde rastgele seçimler yaparak **Random Subspace** tekniğini geliştirmiştir.

- Gözlem (**Observation**): Veri setinizdeki satırlar, yani bağımsız ve bağımlı değişkenlerin değerlerini içeren örnekler.
- Değişken (**Variable**): Veri setinizdeki sütunlar (özellikler), yani bağımsız değişkenler (inputlar) ya da bağımlı değişken (hedef).

**Random Subspace** tekniği, bağımsız değişkenler (özellikler) arasından rastgele bir alt küme seçer ve karar ağacı bu seçilen alt küme üzerinden dallanma işlemi yapar.

Gözlemler bootstrap yöntemi ile, değişkenler random subspace yöntemi ile seçilir.

Böylelikle Breiman **Bagging ve Random Subspace yöntemlerini birleştirerek Random Forest tekniğini oluşturdu** ve hem gözlem birimlerinde hem de değişkenlerde rastgeleliği sağlayarak **CART algoritmasındaki aşırı öğrenme meyiline karşı çözüm üretmiş** oldu.

## Random Forest ve Decision Tree Arasındaki Farklar:

### • Model Sayısı:

- **Decision Tree:** Tek bir karar ağacıdır. Verileri tek bir yapıda işler ve sonuçlar bu tek ağacın tahminine dayanır.
- **Random Forest:** Birden fazla karar ağacından oluşur. Genellikle onlarca hatta yüzlerce karar ağacı bir araya getirilir ve bu ağaçların tahminleri birleştirilir.

### • Tahmin Yöntemi:

- **Decision Tree:** Sonuç, tek bir ağaç üzerinden alınır. Bu, modelin aşırı uyum (overfitting) yapmasına neden olabilir, çünkü sadece bir modelin kararlarına dayanır.
- **Random Forest:** Her bir karar ağacı farklı alt veri setleri üzerinde eğitilir ve sonuç olarak her ağaç farklı bir tahmin yapar. Tahminler çoğunluk oylaması (sınıflandırma için) veya ortalama (regresyon için) ile birleştirilir. Bu yöntem, modelin genel performansını artırır ve aşırı uyum riskini azaltır.

### • Genellenebilirlik:

- **Decision Tree:** Tek bir ağaç olduğu için, yeni veriler üzerinde zayıf performans gösterebilir.
- **Random Forest:** Birden fazla ağaç kullandığı için, daha iyi bir genellenebilirlik sağlar. Farklı ağaçların tahminleri birleştirildiğinde, daha kararlı ve doğru sonuçlar elde edilir.

- Bagging (Breiman, 1996) ile Random Subspace (Ho, 1998) yöntemlerinin birleşimi ile oluşmuştur.
- Ağaçlar için gözlemler bootstrap rastgele örnek seçim yöntemi ile değişkenler random subspace yöntemi ile seçilir.
- Karar ağacının her bir düğümünde en iyi dallara ayırıcı (bilgi kazancı) değişken tüm değişkenler arasından rastgele seçilen daha az sayıdaki değişken arasından seçilir.
- Ağaç oluşturmada veri setinin 2/3'ü kullanılır. Dışarıda kalan veri ağaçların performans değerlendirmesi ve değişken öneminin belirlenmesi için kullanılır.
- Her düğüm noktasında rastgele değişken seçimi yapılır. (regresyon'da  $p/3$ , sınıflama'da  $\sqrt{p}$ )

#### **Random\_forest\_model.get\_params ()**

```
{'bootstrap': True,  
'ccp_alpha': 0.0,  
'criterion': 'squared_error',  
'max_depth': None,  
'max_features': 1.0,  
'max_leaf_nodes': None,  
'max_samples': None,  
'min_impurity_decrease': 0.0,  
'min_samples_leaf': 1,  
'min_samples_split': 2,  
'min_weight_fraction_leaf': 0.0,  
'n_estimators': 100,  
'n_jobs': None,  
'oob_score': False,  
'random_state': 42,  
'verbose': 0,  
'Warm_start': False}
```

- **bootstrap**: Modelin bootstrap örnekleme yöntemini kullanıp kullanmadığını belirtir.
- **ccp\_alpha**: Karar ağaçlarının budanmasında kullanılan bir parametre.
- **criterion**: Modelin hata ölçümünü belirler.
- **max\_depth**: Karar ağaçlarının maksimum derinliğini belirler.
- **max\_features**: Her bir ağaçta değerlendirilecek maksimum özellik sayısını belirler.
- **max\_leaf\_nodes**: Ağaçta maksimum yaprak düğüm sayısını belirler.
- **max\_samples**: Her bir ağaç için kullanılacak örnek sayısını belirler.
- **min\_impurity\_decrease**: Düğümün bölünmesi için gereken minimum bilgi kazancı düşüştür.
- **min\_samples\_leaf**: Bir yaprak düğümde bulunması gereken minimum örnek sayısını belirler.
- **min\_samples\_split**: Bir düğümün bölünebilmesi için gereken minimum örnek sayısını belirtir.
- **min\_weight\_fraction\_leaf**: Bir yaprak düğümdeki örneklerin toplam ağırlıklarının toplam ağırlığa oranını belirler.
- **n\_estimators**: Random Forest modelindeki karar ağaçlarının sayısını belirtir.
- **n\_jobs**: Modelin eğitim ve tahmin sürecinde kullanılacak iş parçacığı sayısını belirtir.
- **oob\_score**: Out-of-Bag (OOB) örneklerini kullanarak modelin doğruluğunu değerlendirme seçeneğidir.
- **random\_state**: Rastgelelik için kullanılan bir tohum değeridir.
- **verbose**: Modelin eğitim sürecinde çıktı detay seviyesini belirler.
- **warm\_start**: Modelin önceki öğrenme aşamasından devam edip etmeyeceğini belirler.

## Özellik Önem Derecelerini

```
importance = pd.DataFrame ({
    "Importance": rf_tuned.feature_importances_,
    "index": X_train.columns
})
```

- **rf\_tuned.feature\_importances\_**: Random Forest modelinin, her bir özellik için hesapladığı önem derecelerini içeren bir dizi.
- **X\_train.columns**: Eğitim setindeki özelliklerin isimlerini alır.
- **DataFrame**'de iki sütun bulunur: "Importance" (özelliklerin önem dereceleri) ve "index" (özellik isimleri).

```
importance.sort_values (by="Importance", axis=0, ascending=True)
```

**DataFrame**'i "Importance" sütununa göre artan sırayla sıralar.



## Gradient Boosting Machines (GBM) (boost: artırmak, yükseltmek, desteklemek)

AdaBoost'un sınıflandırma ve regresyon problemlerine kolayca uyarlanabilen genelleştirilmiş versiyonudur.

**AdaptiveBoosting:** Zayıf sınıflandırıcılar bir araya gelerek güçlü bir sınıflandırıcı oluşturabilir.

Artıklar üzerine tek bir tahminsel model formunda olan modeller serisi kurulur. Yani bir ağacın çıktıları üzerine bir başka ağaç kurulur.

Boosting Yöntemleri: **Boost etmek**, modellerin hatalarını adım adım düzeltmek ve toplu olarak güçlü bir model ortaya çıkarmaktır. Bu şekilde, zayıf öğreniciler (örneğin basit karar ağaçları) bir araya gelerek daha güçlü ve daha doğru bir model oluştururlar.

- Gradient boosting tek bir tahminsel model formunda olan modeller serisi oluşturur.
- Seri içerisindeki bir model serisindeki bir önceki modelin tahmin artıklarının/ hatalarının (residuals) üzerine kurularak (fit) oluşturulur.
- GBM diferansiyellenebilen herhangi bir kayıp fonksiyonunu optimize edebilen Gradient descent algoritmasını kullanmakta.
- GB bir çok temel öğrenici tipi (base learner type) kullanabilir. (Trees, linear terms, splines,...)
- Cost fonksiyonları ve link fonksiyonları modifiye edilebilir.
- Boosting + Gradient Descent

1.50

## Gradient Boosting Machines



AdaBoost'un sınıflandırma ve regresyon problemlerine kolayca uyarlanabilen genelleştirilmiş versiyonudur.

Artıklar üzerine tek bir tahminsel model formunda olan modeller serisi kurulur.

**Optimizasyon problemi**, hedefe ulaşmak için en iyi çözümü bulmaya yönelik bir matematiksel problemdir. Bu tür problemlerde amaç, bir **hedef fonksiyonunu** maksimuma çıkarmak (maksimizasyon) veya minimuma indirmek (minimizasyon) olabilir. Genellikle, bir dizi **değişken** ve **kısıtlar** bulunur, ve çözüm bu değişkenlerin en iyi değerlerini bulmayı içerir.

Örnek olarak:

- **Maksimizasyon problemi:** Bir fabrikada üretim kârını en üst düzeye çıkarmak.
- **Minimizasyon problemi:** Bir lojistik sisteminde taşımacılık maliyetini en aza indirmek.

#### **HİPERPARAMETRELER:**

- **alpha:** Quantile regression'da kullanılır, modelin hangi yüzdeyle ilgili hataları minimize edeceğini belirler.
- **ccp\_alpha:** Cost Complexity Pruning (budama) için bir parametre; karar ağacının ne kadar budanacağını kontrol eder.
- **criterion:** Karar ağaçlarındaki bölünme kalitesini ölçmek için kullanılan kriter. *friedman\_mse*, boosted ağaçlar için mean squared error'ü optimize eder.
- **init:** Boosting sürecinin başlangıcında kullanılan model. Varsayılan olarak None (başlangıçta herhangi bir model yok).
- **learning\_rate:** Öğrenme oranını kontrol eder. Her adımda yeni öğrenilen bilginin ne kadarının modele katılacağını belirler.
- **loss:** Modelin optimizasyon için kullanacağı kayıp fonksiyonu. *squared\_error*, en küçük kareler hatasıdır.
- **max\_depth:** Karar ağaçlarının maksimum derinliğini belirler, daha derin ağaçlar daha karmaşık olabilir.
- **max\_features:** Her bölünmede değerlendirilecek maksimum özellik sayısını kontrol eder. Rastgele alt özellik seçimini sağlar.
- **max\_leaf\_nodes:** Maksimum yaprak düğüm sayısını sınırlar. Ağaçların karmaşıklığını sınırlamak için kullanılır.
- **min\_impurity\_decrease:** Bölünme gerçekleşmesi için gereken minimum saf olmayanlık azalmasını belirler.
- **min\_samples\_leaf:** Her yaprakta bulunması gereken minimum örnek sayısını belirtir.
- **min\_samples\_split:** Bir düğümün bölünmesi için gereken minimum örnek sayısını belirtir.
- **min\_weight\_fraction\_leaf:** Yaprak düğümdeki minimum ağırlık oranını kontrol eder, örneklerin ağırlıklı dağılımına dayalıdır.
- **n\_estimators:** Oluşturulacak ağaç sayısını belirtir.
- **n\_iter\_no\_change:** İyileşme olmadığı takdirde kaç iterasyon sonra erken durdurma yapılacağını belirler.
- **random\_state:** Rastgelelik için kullanılan tohum değeri. Aynı sonuçları almak için kullanılır.
- **subsample:** Her ağacı oluştururken kullanılacak veri alt kümesinin oranını belirler.
- **tol:** Öğrenmenin durdurulacağı minimum iyileşme oranını belirler.
- **validation\_fraction:** Early stopping (erken durdurma) için doğrulama setinin oranını belirler.

- **verbose:** Eğitim sürecinde ne kadar bilgi gösterileceğini kontrol eder.
- **warm\_start:** Modelin önceki eğitiminin üzerine devam edilip edilmeyeceğini kontrol eder.

### 1. learning\_rate (Öğrenme Oranı)

- **Ne İşe Yarıyor?** Modelin her adımda ne kadar büyük bir öğrenme adımı atacağını belirler. Küçük bir değer, modelin yavaşça öğrenmesini sağlar ama bu genellikle daha iyi bir doğruluğa ulaşmayı sağlar. Büyük bir öğrenme oranı ise modelin daha hızlı öğrenmesine yardımcı olur, ancak aşırı uyum (overfitting) ya da düşük doğruluk riski taşır.
- **Değerler:** [0.001, 0.01, 0.1, 0.2]  
Daha düşük öğrenme oranları genellikle modelin daha iyi öğrenmesine olanak tanır, ancak eğitim süresini uzatabilir.

### 2. max\_depth (Maksimum Derinlik)

- **Ne İşe Yarıyor?** Modelin her bir karar ağacının maksimum derinliğini belirler. Derinlik arttıkça model daha karmaşık yapılar öğrenebilir, ancak aşırı uyum riski de artar.
- **Değerler:** [3, 5, 8, 10]  
Küçük değerler genellikle daha basit modeller oluşturur ve aşırı uyumu önlemeye yardımcı olur.

### 3. n\_estimators (Tahminci Sayısı)

- **Ne İşe Yarıyor?** Modelde kullanılacak karar ağaçlarının (zayıf tahmincilerin) sayısını belirler. Daha fazla ağaç, modelin daha karmaşık ilişkileri öğrenmesini sağlar, ancak eğitim süresi uzayabilir ve aşırı uyum riski olabilir.
- **Değerler:** [100, 200, 500]  
Daha fazla ağaç, genellikle modelin doğruluğunu artırır, ancak eğitim süresini uzatır.

### 4. subsample (Alt Örneklem Oranı)

- **Ne İşe Yarıyor:** Her bir ağacın eğitimi için kullanılan veri alt kümesinin oranını belirtir. subsample=1.0 kullanıldığında tüm veri kümesi kullanılır, daha düşük bir değer ise veri kümesinin sadece bir kısmının kullanılacağı anlamına gelir. Bu, varyansı azaltarak aşırı uyumun önlenmesine yardımcı olabilir.
- **Değerler:** [0.5, 0.8]  
Genellikle 0.5 ile 0.8 arasında seçilen alt örneklem oranları, modelin daha genelleştirici olmasını sağlar.

### Bu Parametreler Neye Göre Belirleniyor?

- **Deneme Yanılma (Grid Search/Cross Validation):** Parametreler genellikle bir **Grid Search** veya **Randomized Search** ile çapraz doğrulama (cross-validation) yöntemleri kullanılarak en uygun değerler bulunur.

## Kayıp Fonksiyonu Nedir?

Kayıp fonksiyonu, bir modelin **ne kadar kötü tahmin yaptığını** ölçen bir formüldür.

### Örnek:

Diyelim ki sen bir model yaptın ve bu model insanlara "yarın hava sıcaklığı kaç derece olacak?" sorusunu cevaplıyor. Gerçek sıcaklık 20°C, ama modelin "25°C olacak" dedi. İşte **kayıp fonksiyonu**, modelin yaptığı bu tahmin hatasını sayısal olarak ölçer ve sana "bu tahminde şu kadar hata var" der.

### Neden Önemli?

Kayıp fonksiyonu sayesinde, modelin yanlış tahminlerini görebiliriz ve bu sayede modeli daha iyi hale getirebiliriz. Modelin hata yaptığı her seferde, bu hata kaydedilir ve **modelin öğrenmesi** için kullanılır.

### Kısacası:

- Kayıp fonksiyonu, **gerçek sonuç** ile **modelin tahmin ettiği sonuç** arasındaki farkı ölçer.
- Bu fark ne kadar büyükse, model o kadar kötü tahmin yapmış demektir.
- Amaç, **kayıp fonksiyonunu** olabildiğince **küçük** yapmak, yani modelin daha iyi tahminler yapmasını sağlamaktır.
- **squared\_error**: Hataların karesiyle çalışır, daha büyük hataları daha ağır cezalandırır.
- **absolute\_error**: Hataların mutlak değerleriyle çalışır, aykırı değerlere karşı dayanıklıdır.
- **huber**: MSE ve LAD arasında bir denge sağlar.
- **quantile**: Belirli bir kuantili optimize etmek için kullanılır.

## XGBoost (Extreme gradient boosting)

GBM in hız ve tahmin performansını arttırmak üzere optimize edilmiş, ölçeklenebilir ve farklı platformlara entegre edilebilir.

### Light GBM

XGBoost'un eğitim süresi performansını arttırmaya yönelik geliştirilen bir GBM türüdür.

- Daha performanslı
- Level-wise büyüme stratejisi yerine Leaf-wise büyüme stratejisi
- Breadth-first search (BFS) yerine depth-first search (DFS)

#### 1. Level-Wise (Seviye Bazlı)

- Karar ağaçları, her seferinde bir seviyedeki tüm düğümleri eş zamanlı olarak böler. Yani, her seviyede mevcut olan tüm düğümler üzerinde karar verilir.

#### 2. Leaf-Wise (Yaprak Bazlı)

- Karar ağaçları, her seferinde yalnızca en az hata oranına sahip yaprak düğümünü böler. Yani, mevcut yapraklardan birini hedef alır ve onu daha fazla bölerek daha iyi bir model oluşturur.

🚦 **Level-Wise:** Düğümleri seviyede böler, daha hızlı ama potansiyel olarak daha az etkili.

🚦 **Leaf-Wise:** Yaprakları böler, daha iyi performans ama daha fazla kaynak gerektirir.

## Categoric Boosting (CatBoost)

Kategorik veriler ile otomatik olarak mücadele edebilen, hızlı bir GBM türevidir.

Elimizdeki sayısal verileri, ağaçların performansını arttırmak amacıyla kategorik değerlere çevirip dallanmaların daha kolay oluşmasını sağlar.

- Kategorik değişken desteği
- Hızlı ve ölçeklenebilir GPU desteği
- Daha başarılı tahminler
- Hızlı train ve hızlı tahmin
- Rusyanın ilk açık kaynak kodlu, başarılı ML çalışması

