14.09.2025

MAKİNE ÖĞRENİMİ

Gözetimli ve Gözetimsiz Öğrenme

Preprocessing

Algoritmalar

Hyper Parametreler

Ömer Faruk Gülşen

LinkendIn

Kaggle

Github

İçindekiler

MAKİNE ÖĞRENİMİ ALGORİTMALARI

1. GÖZETİMLİ ÖĞRENME (Supervised Learning) :

Gözetimli öğrenme, makine öğrenmesinde hem giriş verilerinin (X) hem de bu verilere karşılık gelen doğru çıktı (Y) etiketlerinin bulunduğu veri setleri kullanılarak algoritmanın eğitilmesi sürecidir. Bu yöntemde algoritma, X verileri ile onlara ait Y etiketleri arasındaki ilişkiyi öğrenir ve daha sonra karşısına çıkan yeni X verilerinin olası Y çıktısını tahmin etmeye çalışır. Gözetimli öğrenme, **regresyon** ve **sınıflandırma** olmak üzere iki farklı yönteme sahiptir.

1.1. Regresyon (Regression) :

Regresyon modelleri, giriş verileri üzerinden eğitilerek çıktı olarak **sürekli (continuous)** bir değer tahmin etmeyi amaçlayan algoritmalardır. Örneğin, bir ev veri setinde oda sayısı, yaşı, ısınma şekli gibi özelliklere bakarak evin fiyatını tahmin etmeye çalışan modeller regresyon algoritmalarına örnektir. **Genellikle fiyat, sıcaklık, ağırlık gibi ölçülebilir büyüklüklerin tahmin edilmesinde kullanılır.**

1.2. Regresyon Değerlendirme Metrikleri :

- **R^2 Skoru :** `from sklearn.metrics import r2_score`

R^2 skoru, bir modelin tahminleri ile gerçek değerler arasındaki farkı ölçerek, modelin gerçek değerlerle tahminler arasındaki ilişkiyi ne kadar iyi yakaladığını gösterir. Başka bir deyişle, regresyon problemlerinde bir tür doğruluk (accuracy) ölçütü olarak kabul edilebilir. R^2 skoru **1'e ne kadar yakınsa**, modelin tahminleri gerçek değerlere o kadar uygundur.

- **Mean Absolute Error :** `from sklearn.metrics import mean_absolute_error`

MAE, regresyon modellerinde tahmin edilen değerler ile gerçek değerler arasındaki farkların mutlak değerlerinin ortalamasını alan bir hata metriğidir. Yani tahminlerin gerçek değerlerden ortalama olarak ne kadar saptığını gösterir.

- **Mean Squared Error :** `from sklearn.metrics import mean_squared_error`

MSE, tahmin edilen değerler ile gerçek değerler arasındaki farkların karesinin ortalamasını alan bir hata metriğidir. Hataları kareye alması, büyük hataların etkisini daha fazla gösterir.

1.3. Sınıflandırma (Classification) :

Sınıflandırma modelleri, giriş verileri üzerinden eğitilerek çıktı olarak **kategorik** bir değer tahmin etmeye çalışan algoritmalardır. Örneğin, bir çiçeğin çeşitli yaprak uzunluklarına bakılarak çiçeğin türünün tahmin edilmeye çalışılması, e-mailin içeriğine bakılarak bu e-mailin spam olup olmadığının kontrol edilmeye çalışılması sınıflandırma algoritmaları ile mümkündür.

1.4. Sınıflandırma Değerlendirme Metrikleri :

- **Accuracy (Doğruluk) :** `from sklearn.metrics import accuracy_score`

Accuracy, sınıflandırma modellerinde kullanılan temel bir değerlendirme metriğidir. Modelin yaptığı **doğru tahminlerin**, toplam tahmin sayısına oranını verir. Yani, modelin ne kadar doğru sınıflandırma yaptığını ölçer. 1'e ne kadar yakınsa model o kadar iyi tahminlerde bulunuyordur.

- **Precision (Kesinlik) :** `from sklearn.metrics import classification_report`

Precision, modelin **pozitif tahminlerinden kaç tanesinin gerçekten pozitif** olduğunu gösterir. Yani, modelin pozitif sınıf tahminlerinin doğruluğunu ölçer.

$$Precision = \frac{True\ Positives\ (TP)}{True\ Positives\ (TP) + False\ Positives\ (FP)}$$

True Positives -> Gerçek değer pozitif ve model tahmini de pozitif.

False Positives -> Gerçek değer negatif ama model tahmini pozitif.

True Negatives -> Gerçek değer negatif ve model tahmini de negatif.

False Negatives -> Gerçek değer pozitif ama model tahmini negatif.

- **Recall (Duyarlılık) :** `from sklearn.metrics import classification_report`

Recall, gerçek pozitiflerin (TP) toplam gerçek pozitiflere (TP + FN) oranını gösterir. Yani, modelin **pozitif sınıfı ne kadar iyi yakaladığını** ölçer.

$$Recall = \frac{True\ Positives\ (TP)}{True\ Positives\ (TP) + False\ Neagtives\ (FN)}$$

- **F1 Score :** `from sklearn.metrics import classification_report`

F1 Score, sınıflandırma modellerinde **precision (kesinlik)** ve **recall (duyarlılık)** ölçütlerini tek bir değer hâlinde birleştiren bir metriktir. Özellikle veri setinde sınıflar dengesiz olduğunda kullanışlıdır.

$$F1\ Score = 2 \cdot \frac{Precision\ \times Recall}{Precision + Recall}$$

- **Confusion Matrix (Karışıklık Matrisi) :** `from sklearn.metrics import confusion_matrix`

Confusion Matrix, sınıflandırma modellerinin tahmin sonuçlarını özetleyen ve modelin hangi sınıfları doğru veya yanlış tahmin ettiğini gösteren bir tablodur. Matriste, **gerçek değerler** satırlarda, **model tahminleri** sütunlarda yer alır ve her hücre TP, FP, TN, FN sayılarını içerir. Bu sayede modelin performansını görsel olarak anlamak kolaylaşır. Aşağıdaki tablo örnek bir confusion matrixi göstermektedir:

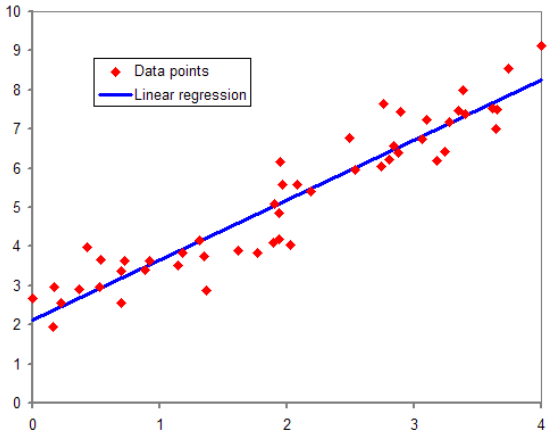
Gerçek / Tahmin	Pozitif Tahminler	Negatif Tahminler
Gerçek Pozitifler	TP	FN
Gerçek Negatifler	FP	TN

2. GÖZETİMLİ ÖĞRENME ALGORİTMALARI :

2.1. Linear Regression :

- **Import :** `from sklearn.linear_model import LinearRegression`

Linear regresyon, bağımlı değişken (çıktı) ile bir veya daha fazla bağımsız değişken (girdi) arasındaki ilişkiyi **doğrusal bir denklem** ile modellemeyi amaçlayan bir regresyon yöntemidir. Model, girdilerle çıktı arasındaki ilişkiyi en iyi şekilde temsil eden **düz bir doğru** bulmaya çalışır ve bu doğru üzerinden tahminler yapar. Özelliklerin farklı ölçeklerde olması durumunda modelin dengeli öğrenebilmesi için girdilerin ölçeklendirilmesi (scaling) önemlidir.



2.2. Ridge Regression (L2):

- **Import** : `from sklearn.linear_model import Ridge`

Ridge regresyon, linear regresyonun bir çeşididir ve modelin **aşırı öğrenme (overfitting)** yapmasını önlemek için katsayılara bir **ceza (regularization)** uygular. Bu ceza, katsayıların büyüklüğünü sınırlayarak modelin daha genel ve stabil tahminler yapmasını sağlar. Girdiler scale edilmelidir.

2.3. Lasso Regression (L1):

- **Import :** `from sklearn.linear_model import Lasso`

Lasso regresyon, lineer regresyonun bir çeşididir ve modelin aşırı öğrenmesini (overfitting) önlemek için katsayılara ceza uygular. Ridge'den farkı, Lasso bazı katsayıları **tamamen sıfıra indirme** yeteneğine sahiptir, böylece gereksiz özellikleri otomatik olarak elemiş olur. Girdiler scale edilmelidir.

2.4. ElasticNet Regression (L1+L2):

- **Import :** `from sklearn.linear_model import ElasticNet`

Elastic Net, Ridge ve Lasso regresyonun bir kombinasyonudur. Modelin aşırı öğrenmesini önlemek için katsayılara ceza uygular ve hem **Ridge'in katsayı küçültme** hem de **Lasso'nun bazı katsayıları sıfırlama** özelliklerini bir arada sunar. Girdiler scale edilmelidir.

2.5. Logistic Regression :

- **Import :** `from sklearn.linear_model import LogisticRegression`

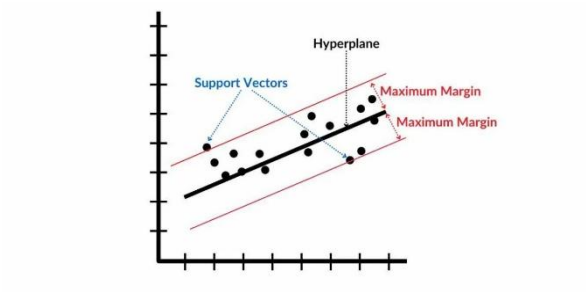
Logistic regresyon, bağımlı değişkenin **kategorik** olduğu (genellikle 0 veya 1 gibi iki sınıflı) sınıflandırma problemlerinde kullanılan bir regresyon yöntemidir. Model, girdiler ile çıktı arasındaki ilişkiyi **sigmoid fonksiyonu** ile doğrusal olmayan bir olasılık değerine dönüştürür ve bu olasılık üzerinden sınıflandırma yapar. Girdiler scale edilmelidir.

2.6. Support Vector Machines :

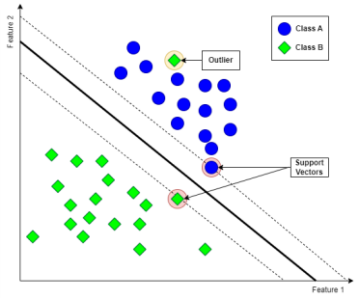
- **Import Regressor :** `from sklearn.svm import SVR`
- **Import Classifier :** `from sklearn.svm import SVC`

SVM, sınıflandırma ve regresyon problemlerinde kullanılan güçlü bir makine öğrenmesi algoritmasıdır. Temel mantığı, farklı sınıfları **en geniş marj ile ayıran bir sınır (hiperdüzlem)** bulmaktır. SVM, hem lineer olarak ayrılabilen hem de lineer olmayan (kernel kullanarak) veriler üzerinde çalışabilir.

SVR



SVC



Parametre	Açıklama	Default	Alternatif Seçenekler
C	Marjın genişliği ve hataya tolerans	1.0	0.1, 10, 100 (sayısal değerler)
kernel	Karar sınırını dönüştürmek için kullanılan fonksiyon	'rbf'	'linear', 'poly', 'sigmoid', 'precomputed'
gamma	Noktaların etkili olduğu alan (RBF, poly)	'scale'	'auto', sayısal değeler (0.01,0.1)

2.7. Naive Bayes :

- **Import :** `from sklearn.naive_bayes import BernoulliNB`
- **Import :** `from sklearn.naive_bayes import GaussianNB`
- **Import :** `from sklearn.naive_bayes import MultinomialNB`

Naive Bayes, olasılık temelli bir **sınıflandırma (classification)** algoritmasıdır ve Bayes teoremi ile çalışır. Eğer girdiler (X) sürekli sayısal verilerden oluşuyorsa **Gaussian Naive Bayes**, sayısal sıklık temelli verilerden oluşuyorsa **Multinomial Naive Bayes**, binary veriler için ise **Bernoulli Naive Bayes** kullanılır. Naive Bayes hızlıdır, büyük veri setlerinde etkilidir, az sayıda veriyle bile iyi sonuç verebilir ve çoğu durumda **scaling gerekmez**. Olasılık temelli çalıştığı için hyperparameter tuning yapılmasına **gerek yoktur**. Girdiler **normal dağılıma** yakın olmalıdır (özellikle Gaussian Naive Bayes).

2.8. K-Nearest Neighbors (KNN) :

- **Import Classifier :** `from sklearn.neighbors import KNeighborsClassifier`
- **Import Regressor :** `from sklearn.neighbors import KNeighborsRegressor`

K-Nearest Neighbors (KNN), hem **sınıflandırma (classification)** hem de **regresyon (regression)** problemlerinde kullanılabilen, **mesafe tabanlı bir algoritmadır**. Tahmin, veri setindeki en yakın **K komşunun** değerlerine bakılarak yapılır: sınıflandırmada çoğunluk oyuna göre sınıf atanır, regresyonda ise komşuların ortalaması alınır. Basit ve yorumlanabilir bir algoritmadır, ancak **özelliklerin ölçeklendirilmesi (scaling)** çok önemlidir, çünkü mesafeye dayalı çalışır.

Parametre	Açıklama	Default	Alternatif Seçenekler
n_neighbors	Karar verirken dikkate alınacak komşu sayısı	5	1, 3, 7, 10, 20 …
metric	Komşular arasındaki mesafe ölçütü	'minkowski'	'euclidean', 'manhattan', 'chebyshev'
weights	Komşuların etkisini belirler	'uniform' (eşit ağırlık)	'distance' (yakın olan daha fazla etki)

2.9. Decision Tree :

- **Import Regressor :** `from sklearn.tree import DecisionTreeRegressor`
- **Import Classifier :** `from sklearn.tree import DecisionTreeClassifier`

Decision Tree (Karar Ağacı), hem **sınıflandırma (classification)** hem de **regresyon (regression)** problemlerinde kullanılan, **ağaç yapısına dayalı bir algoritmadır**. Model, veriyi dallara ayırarak ve belirli kurallara göre karar noktaları oluşturarak tahminler yapar; her düğüm bir özellik üzerinde ayrım yapar ve yaprak düğümler tahmin sonucunu verir. Yorumlaması kolaydır, hem sayısal hem kategorik verilerle çalışabilir ve genellikle özelliklerin **ölçeklendirilmesine gerek yoktur**. Ancak overfitting'e eğilimlidir. İlk tabakaya **root node** aradakilere **internal node** son sonuçlara da **leaf node** denir. Karar ağaçları dallanıp budaklanarak **sub-tree/branch**ler oluşturur. Dallanan internal nodelara **decision node** da denebilir

Parametre	Açıklama	Default	Alternatif Seçenekler
max_depth	Ağacın maksimum derinliği	None (sınırsız)	3, 5, 10, 20 …
min_samples_split	Bir düğümün bölünebilmesi için gereken minimum örnek sayısı	2	5, 10, 20 …
min_samples_leaf	Yaprak düğümde bulunması gereken minimum örnek sayısı	1	2, 5, 10 …
criterion	Ayrım yaparken kullanılacak ölçüt	'gini' (classification)	'entropy' (classification), 'mse' (regression), 'mae' (regression)
max_features	Bölme için kullanılacak özellik sayısı	None	'sqrt', 'log2', int

2.10. Random Forest :

- **Import Regressor :** `from sklearn.ensemble import RandomForestRegressor`
- **Import Classifier :** `from sklearn.ensemble import RandomForestClassifier`

Random Forest, hem **sınıflandırma (classification)** hem de **regresyon (regression)** problemlerinde kullanılan, **çoklu karar ağaçlarından (ensemble) oluşan bir algoritmadır**. Her bir ağaç, verinin rastgele bir alt kümesi ve rastgele seçilmiş özelliklerle eğitilir ve tahminler **oylama (classification)** veya **ortalama (regression)** ile birleştirilir. Bu sayede tek bir karar ağacına göre daha stabil ve genelleme yeteneği yüksek sonuçlar elde edilir. Overfitting riski tek ağaçlara göre **çok daha düşüktür**. Özelliklerin ölçeklendirilmesi genellikle **gerekmez.**

Parametre	Açıklama	Default	Alternatif Seçenekler
n_estimators	Ağacın sayısı	100	50, 200, 500 …
max_depth	Her ağacın maksimum derinliği	None	3, 5, 10, 20 …
min_samples_split	Bir düğümün bölünebilmesi için gereken minimum örnek sayısı	2	5, 10, 20 …
min_samples_leaf	Yaprak düğümde bulunması gereken minimum örnek sayısı	1	2, 5, 10 …
max_features	Bölme için kullanılacak özellik sayısı	'auto'	'sqrt', 'log2', int

2.11. AdaBoost (Adaptive Boosting):

- **Import Regressor :** `from sklearn.ensemble import AdaBoostRegressor`
- **Import Classifier :** `from sklearn.ensemble import AdaBoostClassifier`

AdaBoost, bir dizi zayıf öğreniciyi ardışık olarak eğitir ve her iterasyonda önceki modelin hatalarını daha fazla önemseyerek ağırlıklandırır. Başlangıçta tüm eğitim örneklerine eşit ağırlık verilir, ardından her zayıf öğrenici eğitildikçe hatalı sınıflandırılan örneklerin ağırlıkları artırılır, doğru sınıflandırılanların ağırlıkları azaltılır; böylece bir sonraki model **residual (hata) üzerinde odaklanır**. Sınıflandırmada her modelin katkısı, kendi **hata oranına göre bir ağırlık katsayısı** ile belirlenir. Regresyon için ise hatalar genellikle **bin aralıkları veya residual değerler** üzerinden normalize edilerek ağırlıklandırılır. Bu yöntem, zayıf öğrenicileri birleştirerek güçlü ve doğruluğu yüksek bir tahmin modeli oluşturur. Overfitting riski genellikle düşüktür. Özelliklerin ölçeklendirilmesi genellikle **gerekmez.**

Parametre	Açıklama	Default	Alternatif Seçenekler
n_estimators	Toplam zayıf öğrenici sayısı	50	100, 200, 500,…
learning_rate	Her modelin ağırlığını çarpan katsayı	1.0	0.01, 0.1, 0.5,…

Not: Default olarak kullanılan **Decision Tree** nedeniyle, base_estimator içindeki parametreler de hyperparameter tuning'de kullanılabilir. Örneğin `estimator__max_depth`, `estimator__min_samples_split` gibi parametreler modelin doğruluğunu etkileyebilir.

2.12. Gradient Boosting :

- **Import Regressor :** `from sklearn.ensemble import GradientBoostingRegressor`
- **Import Classifier :** `from sklearn.ensemble import GradientBoostingClassifier`

Gradient Boosting, hem **sınıflandırma (classification)** hem de **regresyon (regression)** problemlerinde kullanılan, **boosting tabanlı bir ensemble algoritmasıdır**. AdaBoost'a benzer şekilde, ardışık zayıf öğreniciler (genellikle karar ağaçları) eğitilir; her yeni model, önceki modellerin **residual (hata) değerleri** üzerine odaklanır. Tahminler, tüm modellerin çıktılarının birleşimiyle elde edilir. Bu yöntem, hataları ardışık olarak düzelterek güçlü ve yüksek doğruluklu bir model oluşturur. Overfitting riski AdaBoost'a göre **biraz daha yüksektir**. Özelliklerin ölçeklendirilmesi genellikle **gerekmez.**

Parametre	Açıklama	Default	Alternatif Seçenekler
n_estimators	Toplam zayıf öğrenici sayısı	100	200, 500, 1000 …
learning_rate	Her modelin katkısını çarpan katsayı	0.1	0.01, 0.05, 0.5 …
max_depth	Her ağacın maksimum derinliği	3	1, 5, 10 …
min_samples_split	Bir düğümün bölünebilmesi için gereken minimum örnek sayısı	2	5, 10, 20 …
min_samples_leaf	Yaprak düğümde bulunması gereken minimum örnek sayısı	1	2, 5, 10 …
subsample	Her ağacın eğitiminde kullanılacak örnek oranı	1.0	0.5, 0.7, 0.9 …
max_features	Bölme için kullanılacak özellik sayısı	None	'sqrt', 'log2', int
loss	Kullanılacak kayıp fonksiyonu	'ls' (reg) / 'deviance' (cls)	'lad', 'huber', 'quantile' (hepsi regression için)

2.13. XGBoost (Extreme Gradient Boosting) :

- **Import Regressor :** `from xgboost import XGBRegressor`
- **Import Classifier :** `from xgboost import XGBClassifier`

XGBoost, hem **sınıflandırma (classification)** hem de **regresyon (regression)** problemlerinde kullanılan, **optimize edilmiş bir gradient boosting algoritmasıdır**. AdaBoost ve klasik Gradient Boosting gibi ardışık zayıf öğreniciler (genellikle karar ağaçları) ile çalışır; her yeni model, önceki modellerin residual (hata) değerleri üzerinde gradient ve **Hessian (ikinci türev) bilgisi** kullanarak öğrenir. Bu sayede düzeltmeler daha adaptif olur ve model hem doğruluk hem de genelleme açısından daha güçlü hale gelir. Özellikle büyük veri setleri için Gradient Boosting den **çok daha hızlıdır.** Özelliklerin ölçeklendirilmesi genellikle **gerekmez. Level-wise** çalışır.

Parametre	Açıklama	Default	Alternatif Seçenekler
n_estimators	Toplam zayıf öğrenici sayısı	100	200, 500, 1000 …
learning_rate	Her modelin katkısını çarpan katsayı	0.3	0.01, 0.05, 0.1 …
max_depth	Her ağacın maksimum derinliği	6	3, 5, 10 …
subsample	Her ağacın eğitiminde kullanılacak örnek oranı	1.0	0.5, 0.7, 0.9 …
colsample_bytree	Her ağacın eğitiminde kullanılacak özellik oranı	1.0	0.5, 0.7, 0.9 …
gamma	Bölme için minimum loss reduction	0	0.1, 0.5, 1 …
reg_alpha	L1 regularization	0	0.1, 0.5, 1 …
reg_lambda	L2 regularization	1	0.5, 1, 2 …

2.14. LightGBM (LGBM) :

- **Import Regressor :** `from lightgbm import LGBMRegressor`
- **Import Classifier :** `from lightgbm import LGBMClassifier`

LightGBM, hem **sınıflandırma (classification)** hem de **regresyon (regression)** problemlerinde kullanılan, **gradient boosting tabanlı hızlı bir algoritmadır**. XGBoost'a benzer şekilde ardışık zayıf öğreniciler (genellikle karar ağaçları) ile çalışır, fakat **leaf-wise (yaprak bazlı) ağaç büyütme stratejisi** sayesinde doğruluğu artırırken hesaplama süresini kısaltır. Büyük veri setlerinde ve yüksek boyutlu verilerde özellikle hızlıdır. Özelliklerin ölçeklendirilmesi genellikle **gerekmez.** LightGBM, **XGBoost'a göre büyük veri setlerinde daha hızlıdır** ve bellek kullanımını azaltır.

Parametre	Açıklama	Default	Alternatif Seçenekler
n_estimators	Toplam zayıf öğrenici sayısı	100	200, 500, 1000 …
learning_rate	Her modelin katkısını çarpan katsayı	0.1	0.01, 0.05, 0.3 …
max_depth	Ağacın maksimum derinliği	-1 (sınırsız)	3, 5, 10 …
num_leaves	Bir ağacın maksimum yaprak sayısı	31	15, 50, 100 …
min_data_in_leaf	Yaprak başına minimum örnek sayısı	20	10, 50, 100 …
subsample	Her ağacın eğitiminde kullanılacak örnek oranı	1.0	0.5, 0.7, 0.9 …
colsample_bytree	Her ağacın eğitiminde kullanılacak özellik oranı	1.0	0.5, 0.7, 0.9 …

3. GÖZETİMSİZ ÖĞRENME (Unsupervised Learning) :

Unsupervised Learning (Gözetimsiz Öğrenme), veri setinde yalnızca giriş verilerinin (X) bulunduğu ve doğru çıktı veya etiketlerin (Y) olmadığı öğrenme yöntemleridir. Bu yöntemlerde algoritmalar, verideki gizli yapıları, desenleri ve ilişkileri keşfederek veriyi anlamaya, benzer örnekleri gruplamaya veya boyutunu azaltarak özetlemeye çalışır. Gözetimsiz öğrenme genellikle iki şekilde kullanılır: **Kümeleme (Clustering)**, verileri benzerliklerine göre gruplar (ör. K-Means, DBSCAN) ve **Boyut indirgeme (Dimensionality Reduction)**, verinin boyutunu azaltırken temel varyansı korur (ör. PCA). Etiketli verinin olmadığı durumlarda veri keşfi, görselleştirme ve özellik mühendisliği için oldukça faydalıdır.

3.1. Gözetimsiz Öğrenme Değerlendirme Metrikleri :

- **Silhouette Score :** `from sklearn.metrics import silhouette_score`

Gözetimsiz öğrenmede özellikle kümeleme (clustering) algoritmalarının başarısını değerlendirmek için kullanılan bir metriktir. Her bir verinin kendi kümesindeki benzerliğini ve diğer kümelerden farklılığını ölçer. Skor -1 ile +1 arasında değişir; +1'e yakın değerler, verinin doğru kümelenmiş olduğunu, 0 civarındaki değerler sınırda kaldığını, -1'e yakın değerler ise yanlış kümeye atandığını gösterir.

- **Davies-Bouldin :** `from sklearn.metrics import davies_bouldin_score`

Davies-Bouldin indeksi, kümeleme performansını değerlendirmek için kullanılan bir metriktir. Küme içi benzerlik ile kümeler arası ayrışmayı birlikte dikkate alır. Her küme için, o kümenin diğer kümelerle olan ortalama benzerliği hesaplanır ve en kötü (yani en fazla örtüşen) değerler üzerinden genel bir skor elde edilir. Sonuç değeri ne kadar düşükse kümeler o kadar iyi ayrılmış ve homojen kabul edilir; yüksek değerler ise kümeler arası ayrımın zayıf olduğunu gösterir.

- **Calinski-Harabasz :** `from sklearn.metrics import calinski_harabasz_score`

Calinski-Harabasz indeksi, kümeleme sonuçlarını değerlendirmek için kullanılan bir metriktir. Bu indeks, kümeler arası ayrışmanın (between-cluster dispersion) kümeler içi benzerliğe (within-cluster dispersion) oranını hesaplar. Değer ne kadar yüksekse, kümelerin birbirinden daha iyi ayrıldığını ve her kümenin kendi içinde daha homojen olduğunu gösterir. Bu nedenle yüksek Calinski-Harabasz skoru, iyi bir kümeleme performansının göstergesidir.

4. GÖZETİMSİZ ÖĞRENME ALGORİTMALARI :

4.1. Principal Component Analysis (PCA) :

- **Import :** `from sklearn.decomposition import PCA`

PCA, herhangi bir etiket bilgisi gerektirmediği için unsupervised öğrenme yöntemleri arasında değerlendirilen bir boyut indirgeme ve **feature extraction** tekniğidir; var olan feature'lardan yeni, daha az sayıda fakat bilgi açısından güçlü feature'lar (principal components) elde eder. PCA, kovaryans matrisini çözümleyerek verinin en çok yayıldığı doğrultuları bulur: burada **eigenvector**'ler verinin yayılma doğrultusunu, **eigenvalue**'ler ise bu doğrultudaki yayılımın (varyansın) büyüklüğünü ifade eder. Bu sayede yüksek boyutlu veriler daha az boyuta indirgenerek hem görselleştirme kolaylaşır hem de bazı makine öğrenmesi algoritmalarının performansı artabilir. PCA yapılmadan önce verinin ölçeklendirilmesi **(scaling) gerekmektedir.** İçerisine verilen `n_components` parametresi ile verinin kaç boyuta indirileceği belirtilir. Daha sonra `fit_transform` yapılır.

4.2. K-Means :

- **Import :** `from sklearn.cluster import KMeans`

K-Means, veriyi kümelere ayırmak için kullanılan bir kümeleme algoritmasıdır ve her küme için bir merkez (centroid) oluşturur. Başlangıçta centroidler rastgele konumlandırılır ve prototip kümeleme yöntemi uygulanır; daha sonra her merkez, kendi kümesindeki verileri temsil edecek şekilde güncellenir ve bu döngü ideal merkezler bulunana kadar devam eder. Küme sayısı (K) genellikle **Elbow Method** ile belirlenir; bu yöntemde **WCSS (Within-Cluster Sum of Squares)**, yani her küme elemanının kendi merkezine uzaklıklarının toplamı hesaplanır. Merkez sayısı arttıkça WCSS azalır, ancak ideal K değeri geçildikten sonra azalma belirgin şekilde yavaşlar. Rastgele başlatma nedeniyle bazen sağlıklı kümeleme yapılamayabilir; bu sorunu önlemek için **K-Means++** geliştirilmiş olup, centroidleri birbirinden olabildiğince uzak konumlarda başlatmayı amaçlar.

4.3. Hiyeraşik Kümeleme :

- **Import :** `from sklearn.cluster import AgglomerativeClustering`

Hiyerarşik Kümeleme, verileri ya küçük kümelerden başlayarak büyütme (**agglomerative clustering**) ya da büyük bir kümeden başlayarak bölme (**divisive clustering**) yöntemiyle organize eden bir kümeleme algoritmasıdır. Başlangıçta küme sayısı verilmesine gerek yoktur ve algoritma birçok ara küme oluşturarak verideki yapıyı keşfeder; küçük veri setlerinde iyi performans gösterirken büyük veri setlerinde etkinliği azalabilir. Agglomerative clustering'de her veri başta kendi kümesi olarak kabul edilir ve en yakın kümeler birleştirilerek daha büyük kümeler oluşturulur. Divisive clustering'de ise tüm veri tek bir küme olarak başlar ve ardından bölünerek küçük kümeler elde edilir. Kümeleme sırasında uzaklık hesaplaması sadece **Euclidean** veya **Manhattan** ile sınırlı olmayıp, **cosine similarity** gibi yöntemlerle de özellikle kategorik değişkenler arasında benzerlik ölçülebilir. Genelde **agglomerative clustering** tercih edilir.

Dendogram:

Dendogram, kümeler arası mesafeyi görselleştirmeye yarar. Üzerine bir **threshold (eşik değeri)** belirleyerek, en ideal cluster sayısını görsel olarak seçmek mümkündür.

4.4. DBScan (Density-Based Spatial Clustering of Applications with Noise):

- **Import :** `from sklearn.cluster import DBSCAN`

DBSCAN, yoğunluk bazlı bir kümeleme algoritmasıdır ve veriyi **core point**, **border point** ve **outlier point** olarak sınıflandırır. **Core point**ler, belirlenen yoğunluğu sağlayarak küme oluşturan noktalar; **border point**ler, bu yoğunluk kümelerinin sınırındaki noktalar; **outlier point**ler ise hiçbir kümeye dahil olamayan noktaları ifade eder. Algoritmanın temel parametreleri arasında **minPts**, bir noktanın core point olabilmesi için gereken minimum komşu sayısını, ve **epsilon**, küme oluşumunda kullanılan sabit yarıçapı belirler. DBSCAN, özellikle **outlier tespiti** ve **non-linear veri yapıları** ile çalışmada başarılıdır, ancak yoğunluğu çok dengesiz veri setlerinde performansı düşebilir.

4.5. HDBSCAN (Hierarchical Density-Based Spatial Clustering of Applications with Noise):

- **Import :** `from hdbscan import HDBSCAN`

HDBSCAN, DBSCAN'ın daha optimize edilmiş bir versiyonudur. Yoğunluk farklarına daha duyarlıdır çünkü DBSCAN'daki gibi sabit bir **epsilon (yarıçap)** kullanmak yerine, küme yoğunluğuna bağlı olarak bu değer dinamik olarak değişir. Bu sayede yoğunluğu farklı veri bölgelerinde daha esnek ve doğru kümeleme yapabilir.

5. VERİ ÖN İŞLEMESİ (Preprocessing):

Veri Ön İşleme (Preprocessing), makine öğrenmesi modellerinin doğru ve verimli çalışabilmesi için verinin temizlenmesi, dönüştürülmesi ve uygun hâle getirilmesi sürecidir. Bu aşamada eksik veriler işlenir, kategorik değişkenler sayısallaştırılır, özellikler ölçeklendirilir veya dağılımları normalize edilir. Doğru preprocessing adımları, hem model performansını artırır hem de algoritmaların varsayımlarına uygun veri sağlar.

5.1. Ölçeklendirme (Scaling) :

Scaling, makine öğrenmesi modellerinde farklı ölçeklerdeki özelliklerin eşit ağırlıkta değerlendirilmesini sağlamak için kullanılan bir ön işleme yöntemidir. PCA, SVM, KNN gibi algoritmalar feature ölçeklerine duyarlıdır; bu nedenle veriyi normalize etmek veya standartlaştırmak performans açısından önemlidir. Eğitim verisine `fit_transform`, test verisine ise `transform` uygulanmalıdır, böylece test verisi eğitim sırasında öğrenilen ölçeklendirmeye uygun hâle gelir.

- **StandardScaler :** `from sklearn.preprocessing import StandardScaler`

Veriyi ortalama 0 ve standart sapma 1 olacak şekilde standartlaştırır; özellikle normal dağılıma yakın verilerde etkilidir.

- **MinMaxScaler :** `from sklearn.preprocessing import MinMaxScaler`

Veriyi 0-1 aralığına sıkıştırır ve **uç değerlerden çok etkilenir**. Yani dataset'de çok büyük veya çok küçük outlier'lar varsa, geri kalan veriler çok dar bir aralığa sıkışır. Bu nedenle outlier'lı veri için MinMax genellikle önerilmez.

- **RobustScaler :** `from sklearn.preprocessing import RobustScaler`

Medyan ve interquartile range kullanarak ölçeklendirme yapar; uç değerlerden **daha az etkilenir**.

5.2. Encoding :

Encoding, makine öğrenmesi modellerinde kategorik verilerin sayısal değerlere dönüştürülmesi sürecidir. Algoritmalar genellikle yalnızca sayısal verilerle çalışabildiği için, kategorik sütunlar sayısal forma çevrilir. Bu sayede model, kategorik bilgiyi matematiksel olarak işleyebilir. Scaler'da olduğu gibi encoding de yaparken **train** ve **test** verisini ayırdıktan sonra yapmak önemlidir.

- **LabelEncoder :** `from sklearn.preprocessing import LabelEncoder`

  Kategorik değerleri tamsayıya çevirir. Sıralama duyarlı değildir.

- **OrdinalEncoder :** `from sklearn.preprocessing import OrdinalEncoder`

  Kategorik değerleri sıralama gözeterek (kötü-iyi-çok iyi) numaralandırır. Sıralamayı kendimiz de `categories` değişkenine bir liste vererek belirleyebiliriz.

- **OneHotEncoder :** `from sklearn.preprocessing import OneHotEncoder`

  Her kategori için ayrı bir sütun oluşturulur. Örneğin cinsiyet sütunundaki erkek ve kadın kategorileri erkek ve kadın olarak iki ayrı sütuna bölünür. **Pandas** `get_dummies` ile uygulanabilir veya daha büyük veri setleri ve pipeline ile entegresyon için **Sklearn OneHotEncoder** ile de uygulanabilir.

- **Frequency Encoding:**

  Her kategoriye veri setinde **görünme sıklığına göre bir değer** atayan bir kodlama yöntemidir. Özellikle çok fazla kategori olan sütunlarda kullanışlıdır ve modelin kategorik bilgiyi sayısal olarak işlemesini sağlar.

```
freq = df['kategorik_sutun'].value_counts() / len(df)
df['kategorik_freq'] = df['kategorik_sutun'].map(freq)
```

5.3. Power Transform :

- **Import :** `from sklearn.preprocessing import PowerTransformer`

  Power Transform, verinin dağılımını **daha normal hâle getirmek** için kullanılan bir dönüşüm yöntemidir. Özellikle lineer modellerde ve bazı istatistiksel testlerde, normal dağılıma yakın veri performansı artırır**. Train** ve **test** verisini ayırdıktan sonra yapmaya dikkat edilmelidir.

- **Yeo-Johnson :** `PowerTransformer(method='yeo-johnson')`

  Hem pozitif hem de negatif değerler için transform yapabilir.

- **Box-Cox :** `PowerTransformer(method='box-cox')`

  Yalnıcaz pozitif değerler üzerinde uygulanabilir. Değerler 0 veya negatif olmamalıdır.

Not: Eğer hedef değişken (y) PowerTransformer ile dönüştürüldüyse, model tahminlerini **orijinal ölçeğe çevirmek için** `inverse_transform` kullanılmalıdır. Bu sayede tahminler, gerçek değerlerle doğru şekilde karşılaştırılabilir.