

1- Temel Kavramlar

→ ML Giriş

→ Bilgisayarların insanlara benzer şekilde öğrenmesini sağlamak

→ Değişken Türleri

→ Sayısal → Kesikli (10.5 değil 10)
→ Sürekli (boy)

Kategorik (Nominal → Sınıflar arası fark yok
Ordinal → fark var)

→ Bağımlı değişken → target, dependent
output, response

Bağımsız değişken → feature, independent
input, column
predictor, explanatory

→ Öğrenme Türleri

→ Denetimli (Supervised) → veride labeller var
(bağımlı değişken var)

Denetimsiz (Unsupervised) → bağımlı değişken yok
(label yok)
benzerliklere göre veriler gruplandırılır

Pekiştirmeli (Reinforcement) → Deneme-yanılma
Hatalardan ceza

→ Problem Türleri

→ Bağımlı sayısal → regresyon problemi
değişken kategorikse → sınıflandırma "

→ Model Başarı Değerlendirme Yöntemleri

→ Regresyon Modelleri

$$\bullet \text{MSE} = \frac{1}{n} \sum_{i=0}^n (y_i - \hat{y}_i)^2$$

(mean square error) $\left\{ \begin{array}{l} \text{gerçek değer} \\ \text{tahmin} \end{array} \right.$

$$\bullet \text{RMSE} = \sqrt{\text{MSE}}$$

(root mean square error)

$$\bullet \text{MAE} = \frac{1}{n} \sum_{i=0}^n |y_i - \hat{y}_i|$$

(mean absolute error)

→ Sınıflandırma Modelleri:

$$\bullet \text{Accuracy} = \frac{\text{Doğru Sınıflandırma Sayısı}}{\text{Toplam Sınıflandırılan Gözlem Sayısı}}$$

→ Model Doğrulama Yöntemleri


→ Bir modelin başarısı, o modeli eğitmek için kullanılan veri seti üzerinden değerlendirilir

⚠ Sıkıntısı: model veriyi tanıyo
→ hataları yanlış değerlendirme
→ aşırı öğrenme

Gözüm: Hold-out yöntemi

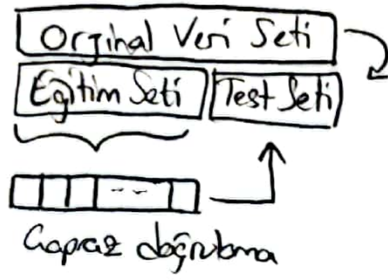
→ Original veri setini eğitim ve test seti olarak ikiye bölmek

→ K Katlı Çapraz Doğrulama Yöntemi (K-Fold Cross Validation)

 → k-1 ile model kur
1 ile test et → tekrarla

→ Gelen hataların ortalamasını al

→ Hold-out ve K-foldu Birlikte Kullanmak



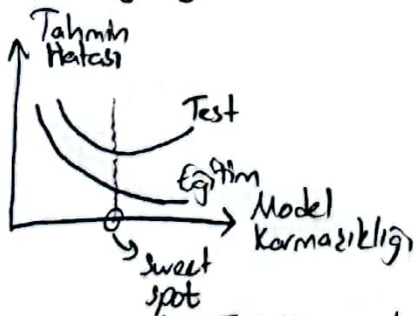
→ Yanlılık - Varyans - Değiş-Tokuş
(Bias - Variance - Trade off)

→ Overfitting → Modelin veriyi öğrenmesi/ezberlemesi
(Yüksek Varyans) ↓
aslen örüntüyü öğrenmedi

Underfitting → Modelin öğrenememesi
(Yüksek yanlılık)

Doğru model → Düşük Yanlılık
Düşük Varyans

→ Overfittingin yakalanması:



→ Test ile model hatasının model karmaşıklığının artmasıyla azaldığı nokta

Çözüm: Model karmaşıklığını düşürmek
modeli daha detaylı tahminler yapmak
hale getirmek
↓
aslen iyi ama bi yordansa
train setini çok iyi öğrendiği
için overfitting oluyor

2- Doğrusal Regresyon

→ Doğrusal Regresyon (Linear Regression)

Amaç: bağımlı ve bağımsız değişkenler arasındaki ilişkiyi doğrusal olarak modellemek

$y \rightarrow$ bağımlı $x \rightarrow$ bağımsız


$b \rightarrow$ sabit, intercept, bias, beta

$w \rightarrow$ katsayı (coefficient)
ağırlık (weight)

$$\hat{y}_i = b + w x_i \rightarrow \text{veri}$$

$$\hat{y}_i = b + w_1 x_1 + \dots + w_p x_p \rightarrow \text{model}$$

→ Ağırlıkların Bulunması


$$\text{Cost}(b, w) = \frac{1}{2m} \sum_{i=1}^m ((b + w x_i) - y_i)^2$$

→ MSE'ye benziyor

→ Parametrelerin Tahmin Edilmesi

1) Analitik Çözüm

Normal Denklemler Yöntemi (OLS)
(En küçük kareler yöntemi)

Simple Linear Regresyon

+ Multiple Linear Regresyon

⚠ Kısmi türevler kullanılıyor

2) Optimizasyon Çözümü

Gradient Descent

→ Doğrusal Regresyon için Gradient Descent

⚠ Gradient Descent bir ML değil, optimizasyon yöntemi, verilen fonksiyonun min değerini bulur

⚠ Learning rate iyi ayarlanmalı

↳ Basit Doğrusal Regresyon
reg_model.intercept_[0]
reg_model.coef_[0][0]

↳ Doğrusal Regresyonda Tahmin İşlemleri
g=sns.regplot(x=X, y=y, --)
 ↓ ↓
 bağımsız bağımlı
 deg. deg.

↳ Doğrusal Regresyonda Tahmin Başarısı

y_pred = reg_model.predict(X)
mean_squared_error(y, y_pred)
y.mean() } bakarak error
y.std() } başarımlı anla

R-KARE değeri → reg_model.score(X, y)

↳ modelin başarısına ilişkin
bir metrik

↳ bağımsız değişken, bağımlı
değişkenin yüzde kaçını açıklayabiliyor

↳ veri arttıkça yükselir

↳ Çoklu Doğrusal Regresyon Modeli
(Multiple Linear Regression)

↳ X_train, X_test, y_train, y_test
= train_test_split(X, y, test_size=0.2,
 train_test
 ← random state=1)
ayrımlarının
aynı olmasını istiyorsan, aynı sayı olmalı

reg_model = LinearRegression().fit(X_train,
 y_train)

↳ Çoklu Doğrusal Regresyonda Tahmin
Başarısı

⚠ Train hatasının test hatasından
daha düşük olması beklenir

↳ Gradient Descent ile Doğrusal Regresyon

↳ cost function (Y, b, w, X)
 ↓ ↓ ↓
 MSE değerini sabit bağımsız
 hesaplıyor bağımlı ağırlık

update_weight(Y, b, w, X, learning_rate)
train(Y, initial-b, initial-w, X, learning_rate)

↳ parametre = train setinden bulunabilen
ağırlıklar

hiperparametre: train setinden bulunamaz
userın belirleyeceği
hatalar

(3- Lojistik Regresyon)

↳ Lojistik Regresyon

" Amaç sınıflandırma problemi için
bağımlı ve bağımsız değişkenler
arasındaki ilişkiyi doğrusal olarak
modellenecek "

↳ Log loss değerini min yapacak
ağırlıklar bulunur

↳ Lojistik Regresyon için Gradient Descent

↳ Entropi ne kadar düşüklse
gelişlilik de o kadar düşüktür,
biz de bunu isteriz

↳ Sınıflandırma Problemlerinde Başarı Değerlendirme

		Tahmin		
		1	0	
Gerçek	1	TP	FN	
	0	FP	TN	

T → true
 F → false
 P → positive
 N → negative

I. tip hata
 ↓
 çok sıklıkla değil

II. tip hata
 ↓
 kritik hata

$$\text{Accuracy} = \frac{(TP + TN)}{(TP + TN + FP + FN)}$$

$$\text{Precision} = \frac{TP}{(TP + FP)} \Rightarrow \text{pozitif sınıf tahminlerinin başarı oranı}$$

↓
tahminlerin başarı

$$\text{Recall} = \frac{TP}{(TP + FN)} \Rightarrow \text{Pozitif sınıfın doğru tahmin edilme oranı}$$

↓
gerçekleri yakalama başarı

$$\text{F1 Score} = \frac{2 * (\text{Precision} * \text{Recall})}{(\text{Precision} + \text{Recall})}$$

↓

Precision ve Recall'in harmonik ortalaması

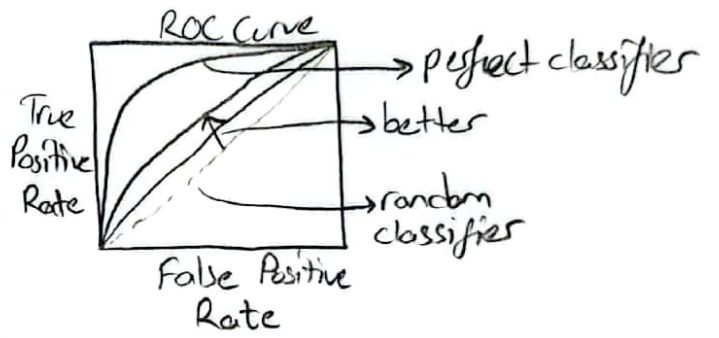
↳ Karmaşıklık Matrisi (Confusion Matrix)

		Tahmin		
		1	0	
Gerçek	1	a	b	a+b
	0	c	d	c+d
		a+c	b+d	

↳ Classification Threshold

⚠ Thresholdu yükselttikçe, accuracy düşebilir

↳ ROC Curve (Receiver Operating Characteristic Curve)



AUC → Area Under Curve

↳ → bunun alanı

(4-KNN)

↳ K-En Yakın Komşu (K-Nearest Neighbor)

↳ K en yakın gözleme göre tahmin yapma

↳ Keşifçi Veri Analizi → EDA (Exploratory Data Analysis)

↳ Model Başarı Değerlendirme

⚠ Veri dengesizse başarı metriği → F1-Score

dengelyse → Accuracy

⚠ Cross-Validation başarımın gerçek durumunu gösteriyor, genelde daha düşük oluyor

↳ Hiperparametre Optimizasyonu

↳ Cross-Validationla hataya x katlı bakılıyor ve buna göre hiperparametreler belirleniyor


5-CART

↳ CART (Classification and Regression Tree)

- ↳ Tahmin başarısı yüksek
- ↳ Random Forestin temeli
- ↳ Amaç veri setindeki karmaşık yapıları basit karar yapılarına dönüştürmek

birbir karar ağacı yapılarının temeli

- ↳ Heterojen bir veri seti, belirlenmiş değişkenlere göre homojen alt gruplara ayrılır

- ↳ Ağaçtaki bölün noktalarının iç düğüm noktaları denir → 

↳ CART

- ↳ RSS (SSE) → Objective fonksiyonun min olması gerekiyor (toplam hatalar)

↳ CART

- ↳ Ne kadar dallanacağız? (min-samples-split) (argümanı)

! Ağaç modelleri overfit etmeye meyillidir bunu engellemek için dallanmayı iyi belirlemek lazım

! Ağaç modellerinde veriyi standartlaştırmaya gerek yok

↳ CART

- ↳ Sınıflandırma problemi için
 - Gini katılığı
 - Entropy
 - Varyans
 - Bilgi
 - Gelişlilik

ne kadar düşük o kadar iyi Gerçekleşen tahminleri değerlendirek bize başarıyla ilgili bilgi verir

$$\text{weighted gini} = \frac{\text{gözlem}}{\text{toplam gözlem}} \cdot \text{onun gini'si} + \frac{\text{diğer gözlem}}{\text{toplam gözlem}} \cdot \text{onun gini'si}$$

↳ Modellere

- ↳ Veri gözlemi artırılarak X Fold yöntemi ile Parametre eklenerek } başarı artırılabilir

↳ Hiperparametre Optimizasyonu

- ↳ `cort_best_grid = GridSearchCV`
(`cort_model`,
`cort_params`,
`cv=5`,
`n_jobs=1`,
`verbose=True`).`fit(X,y)`

Rapor yazdırıyor → `best_params_` → `max_depth`, `min_sample_split` veriler

`best_score_` → en iyi accuracy verilecek olan parametrelere `scoring = "f1"` yaparak f1 score getirebiliyosun

↳ Final Model

`cort_final = cort_model.set_params`
`direk set ← (*) cort_best_grid.best_params`
edilmesini sağlayarak `fit(X,y)` modele

↳ Öğrenme Eğilimlik Model Karmaşıklık
Analiz Etme

↳ Overfittingin önüne geçmek için
model karmaşıklığı azaltılmalı.

↳ Karar Kurallarının Python Kodları

↳ `print(kompik(cart-final-predict).`

`to('python/code'))`

`.. to('sqlalchemy/sqlite'))`

`.. to('excel'))`

↳ Python Kodları ile Tahmin İşlemleri

↳ `predict_with_rules(x)`

`return(—)`

`x=[3, 5, 8, 15, 1]`

`pred = ..(x) → O yada 1 direkt
tahmin ediyö`

⚠ Veri değiştirme hızı yüksek
Yüksek performanslı
Veri tabanına en uygun
Basit

Çözümler
üret

6-Gelişmiş Ağaç Yöntemleri "Leo Breiman"

↳ Random Forests

↳ Birden çok karar ağacının ürettiği
tahminlerin bir araya getirilerek
değerlendirilmesi

↳ Bagging + Random Subspace



Bootstrap + aggregation



Ağaçlar için gözlemler rastgele
seçilir, tekrar edenler olabilir,
seçilmeyenler olabilir

↳ Bu sayede rassallık ve
genellenabilirlik korunmuş dur

↳ Ağaç oluştururken verinin 2/3ü
kullanılır. Geri kalan veri ağaçların
performansını ve değişken önemini
belirlemek için kullanılır.

↳ Her düğüm noktasında rastgele
değişken seçimi yapılır (regresyonda
p/3, sınıflamada \sqrt{p})

⚠ CART'a göre rastgeleliği koruyo,
bu yüzden overfittinge düşmüyö
ve başarıları daha yüksek oluyo

⚠ Baggingde ağaçlar birbirinden
bağımsız; Boostingde ise
bağımlılık var, ağaçlar ardışık
üzere kuruluyo

↳ Gradient Boosting Machines (GBM)

↳ AdaBoost → Adaptive Boosting



Zayıf sınıflandırıcılar bir araya
gelerek güçlü bir sınıflandırıcı
oluşturuyor (Power Rangers :D)

7- Dengesiz Veri Seti Nedir? Nasıl Baza Çıkarılır?

↳ Dengesiz Veri Seti

↳ Sınıflandırma problemlerinde gürültü ve sınıf dağılımlarının birbirine yakın olmadığı zamanlarda çıkar

↳ Çoğunluk sınıfı, azınlık sınıfı domine eder

↳ Model çoğunluk sınıfına yakınlık gösterir

⚠ Veri seti dengesizse accuracy performans ölkümü için doğru bir metrik değildir

↳ Dengeyi Sağlamak İçin:

↳ Doğru Metrik Seçimi

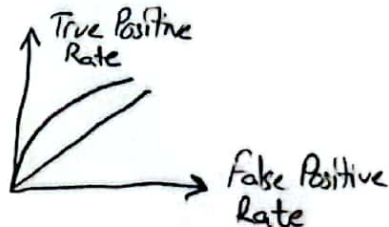
↳ Precision ↓ False Positive ↑

↳ Recall ↓ False Negative ↑

↳ F1 Score → precision ve recallın harmonik ortalaması

↳ Support → sınıfların ölçüm sayısı

↳ ROC Curve



↳ AUC → Area Under ROC

↳ Resampling

↳ Azınlık sınıfına yeni örnekler ekleyip veya çoğunluk sınıftan örnekler çıkartarak veri setinin dengeli hale getirilmesi

↳ Oversampling

↳ Azınlık sınıfının örnekleri kopyalanır ve veri seti dengelenir

↳ Random Oversampling

↳ Rastgele örnekler seçilir

↳ Veri seti küçülece kullanılabılır

↳ Overfittinge neden olabilir

↳ sampling_strategy = minority
↳ sınıfın sayısı artırılır

sampling_strategy = 0.5

↳ azınlık çoğunluğun yansı olur

↳ SMOTE Oversampling

↳ Overfittingi önlemek için azınlık sınıfından sentetik örnek oluşturulur

↳ Azınlık sınıfından rastgele örnek seçilir

En yakın k komşusu alınır
Bunlardan biri rastgele seçilir ve orijinal rastgele örnekle birleştirilip özellik uzayında bir çizgi parçası oluşturularak sentetik örnek oluşturulur

↳ GBM

↳ Gradient Descentin Boosting'e uyarlanmış hali

↳ Hatalar/artıklar üzerine tek bir tahminsel model formunda olan modeller serisi kurulur

↳ Herdeki model, kendinden önceki modelin hataları üzerine fit edilir



Tek bir tahminsel model formunda olan modeller serisi additive şekilde kurulur

↳ GBM

$$\hat{y} = f_0(x) + \sum_{m=1}^M \Delta m(x)$$

↳ base learner

$$\underbrace{f_0 \quad \Delta_1 \quad \Delta_2 \quad \frac{\Delta_3}{\Delta_4}}_{= F_M(x)}$$

$$F_0(x) = f_0(x)$$

$$F_m(x) = F_{m-1}(x) + \Delta m(x)$$



Additive yöntem bunu daha hassas hale getirmek için

$$y = c +$$

↳ x 'e bağımlılığı artırır

↓
 $\sin x, x^2$ gibi

↳ XGBoost (Extreme Gradient Boosting)

↳ GBM'in hız ve tahmin performansı artmış optimize versiyonu

↳ Ölçeklenebilir ve farklı platformlara entegre edilebilir

↳ Light GBM

↳ XGBoostun eğitim süresi performansını arttırmaya yönelik geliştirilen bir diğer GBM türü

↳ Level-wise büyüme yerine leaf-wise büyüme stratejisi

⚠ Veri çok büyük olunca XGBoost yavaş kalır

⚠ XGBoost geniş kapsamlı ilk arama yapıyor, LightGBM derinlemesine ilk arama yapıyor

↳ n-estimator sayısını 10.000'e sığır

↳ Random Search CV (CV=cross validation)

↳ Verilen hiperparametre seti içinden rastgele seçim yapar

↳ Grid Search CV'den daha hızlı çünkü setin içinden rastgele seçiyor, dolayısıyla bütün olasılıkları almamış oluyor

↳ Under Sampling

↳ Çıkarılan örnekler rastgele seçilir

↳ Büyük veri setleri için uygun

↳ Rastgele seçimden dolayı bilgi kaybı yaşanabilir

↳ Near Miss Undersampling

↳ Bilgi kaybını önler

↳ KNN algoritmasına dayanır

↳ Çoğunluk sınıfın örneklerinin, azınlık sınıfın örneklerine olan uzaklığı hesaplanır

↳ Belirtilen k değere göre uzaklığı kısa olan örnekler korunur

↳ Undersampling (Tomek link)

↳ Farklı sınıflara ait en yakın iki örneğin arasındaki çoğunluk sınıfının örnekleri kaldırılarak iki sınıf arasındaki boşluk artırılır

↳ Undersampling (Cluster Centroids)

↳ Örneksiz örneklerin veri setinden çıkarılması. Bu durum kümelemeyle belirlenir

↳ Veri setini dengelemek için daha fazla veri toplanabilir. Class-weight parametresiyle, sınıflardan esit şekilde öğrenilebilen model yaratılması. Anomaly detection veya change detection yapılması. Diğer model performanslarına bakılır

(8- Denetimsiz Öğrenme)

↳ Denetimsiz Öğrenme (Unsupervised Learning)

↳ Bağımlı değişken yok

↳ K-Ortalamalar (K-Means)

↳ Amaç gözlemleri birbirine olan benzerliklerine göre kümelere ayırmak

1) Küme sayısı belirlenir (k)

2) Rastgele k merkez seçilir

3) Her gözlem için k merkezlere olan uzaklıklar hesaplanır

4) Her gözlem en yakın olduğu k merkeze atanır

5) Atama işlemlerinden sonra oluşan kümeler için tekrar merkez hesaplamaları yapılır

6) Bu işlemler belirlenen bir tekrar sayısından sonra tekrar edilir ve küme içi hata koreler toplamının (total within cluster variation) minimum olduğu durumdaki gözlemlerin küme yapısı nihai sayılır (SSE, SSR, SSD)

"Kümler kendi içinde homojen, birbirlerine heterojen olsun"

↳ $k\text{-means} = K\text{Means}(n\text{-cluster} = k) \cdot J(k, df)$

$k\text{-means.inertia} \rightarrow \begin{matrix} SSE \\ SSD \\ SSR \end{matrix}$

\uparrow
X olarak
df alıyo

→ Optimum Küme Sayısını Belirleme

⚠ Küme sayısı arttıkça hatanın düşmesi beklenir, hatta küme sayısı gözlem sayısına eşit olursa hata sıfır olur → overfitting

⚠ Küme sayısı olarak dirseklenmenin en ziddetlendiği nokta seçilir (test ve model hata sayısı)
elbow = KElbowVisualizer(kmeans, k=(2, 20))
elbow = elbow.value

→ Final K-Means Model "dosyanın adı"
df.to_csv("clusters.csv") → dosya yaratır

→ Hiyerarşik Kümeleme Analizi (Hierarchical Cluster Analysis)

↑ Agglomerative (Birleştirici) ↓ Divisive (Bölmeleyici)

⚠ K-Means işlem sırasında dışardan müdahale edemiyorsun ama bu yöntemde müdahale edip kümesi belirleyebiliyorsun

→ Hiyerarşik Kümeleme

⚠ Uzaklık temelli çalıştığı için veriyi standartlaştırmak lazım
 $sc = \text{MinMaxScaler}((0,1))$ yapar
 $df = sc.fit_transform(df)$ bitirsin

→ Temel Bileşen Analizi (PCA) (Principal Component Analysis)

→ Temel fikir; çok değişkenli verinin ana özelliklerini daha az sayıda değişken ile temsil etmek

⚠ Küçük bir bilgi kaybı olur, veri setinin boyutu azaltılır / indirgenir

→ Değişkenler bilginin totalde %90'ını kapsıyorsa kadar tutup, gerisini atabilirsin

→ Temel Bileşen Regresyon Modeli (PCR)

→ Datasetimizde label yok ama sınıfları sınıflandırmak istiyorsak, unsupervised clusterla sokarız ve kümelere label olarak çıkarıp bunu sınıflandırıcıya sokarsın

9- Makine Öğrenmesi - Pipeline

→ Pipeline: Veya ya bunları ekleyerek bir aktarma işlemi

→ Stacking & Ensemble Learning

voting classifier → soft → sınıf güçleşme dahilinde
hard → çoğunluğun sonucu

→ Pipeline :

- 1) Veriyi oku/çık
- 2) Veriyi ön işleme scriptinden geçir
- 3) Genel Modellere bak
- 4) Hiperparametre optimizasyonu
- 5) Voting classifier oluştur
- 6) Bu modeli kaydet

→ Prediction / Scoring

⚠ Yeni veri eklenirse onun da değiştirilmiş geliştirilmiş datasetine göre düzenlenmesi lazım, yoksa hata alırsın