

MACHINE LEARNING

(MACHINE ÖĞRENMESİ)

Miuul

2.08.2023

- Temel Kavramlar
- Doğrusal Regresyon ✓
- Lojistik Regresyon ✓
- KNN ✓
- CART ✓
- Geliştirilmiş Ağaç Yöntemi ✓
- Denetimsiz Öğrenme -
- Yapay Öğrenme Pipeline -

MAKİNE ÖĞRENMESİ NEDİR?

Bilgisayarların insanlara benzer şekilde öğrenme ve hatırlama yeteneklerini geliştirmek amacıyla geliştirilen algoritmalar ve çalışma alanıdır.

REGRESYON

Regresyon Problemleri: Bağımlı değişken yani hedeflediğimiz değişken sayısal olan problemlere regresyon problemleri denir.

Değişken Türleri (Variable Types)

- Sayısal Değişkenler
- Kategorik Değişkenler (Nominal, Ordinal)
- Bağımlı Değişken (target, dependent, output, response)
 - ↳ Denetimli Öğrenme Problemlerinde var.

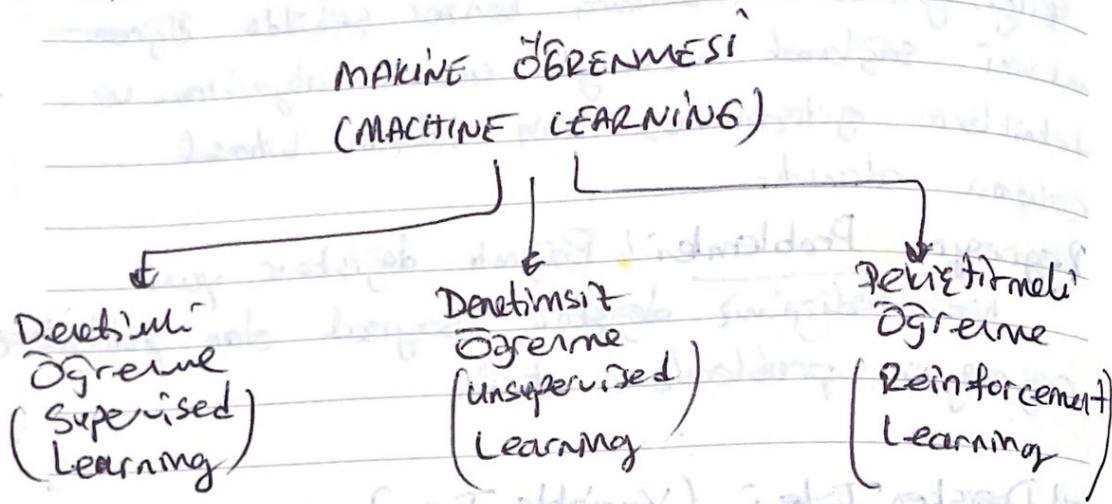
Çıkış, output, target } Bağımlı Değişken Değişkenidir.

Bağımlı değişkenimiz sayısal bu bir REGRESYON PROBLEMİDİR.

Eğer bağımlı değişken kategorikse bu bir SINIFLANDIRMA PROBLEMİDİR.

0-1'den oluşan müşteri sınıfları kategoriktir. Sayısal değildir. 0-1 Tercih Etti, 1-0 Tercih Etti demektir.

Öğrenme Türleri (Learning Types)



Denetimli Öğrenme (Supervised Learning)

Eğer veri setinde label (etiketler) yer alıyorsa bu durumda veri gözetimli, denetimli öğrenme problemine sahiptir. Üzerinde çalışılan veride bir bağımlı değişken varsa bu bir denetimli öğrenme problemidir.

Denetimsiz Öğrenme (Unsupervised Learning)

İlgili problemlerde labeller yoksa yani bağımlı değişken yoksa bu gözetimsiz öğrenmedir.

Problem Türleri (Problem Types)

İlgilenilen makine öğrenmesi problemi regresyon türündemi? Yoksa sınıflandırma problemi mi? diye sorgularız.

- Regresyon Problemi ↔ Bağımlı Değişken Sayısal
- Sınıflandırma Problemi ↔ Bağımlı Değişken Kategorik

Regresyon Modellerinde Başarı Değerlendirme

Ne kadar yüksekse o kadar iyidir.

→ Hata Kareler Ortalaması (MSE)

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

y_i : Gerçek Değer

$\hat{y}_i$: Model Tahmin Değeri

n : Model Sayısı

Kurmuş olduğumuz modeldeki gerçek değer ile model tahmin değerleri arasındaki farkları tahmin etmeyi sağlar.

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

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

MSE'nin kare alma kısmındaki geri döndürmesi gibi düşünebiliriz.

Mutlak hata, gerçek değer ile tahmin edilen değer arasındaki mutlak fark alınarak toplanıp ortalaması alınır.

Veride tahmin sonucu elde edilen başarıyı ölçmek için doğru tahmin / tüm tahminler yaparız.

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

Toplam Sınıflandırılan Örnek Sayısı

→ Sınıflandırma metrikleri için kullanılır.

Ne kadar yüksekse o kadar iyidir.

Katlı Çapraz Doğrulama

Model Doğrulama (Model Validation) Yöntemleri

~~MSE~~

Regresyonda MSE değeri düşük olsun bekliyoruz
Sınıflandırmada Accuracy yüksek olsun bekliyoruz

Holdout Yöntemi (Sınama Seti Yöntemi)

Veri seti D

egitim seti,

test seti olarak ikiye bölünür.

Eğitim seti üzerinde modelleme işlemi,
train işlemi, eğitim işlemi gerçekleştirir.

Model burada öğrenir

Daha sonra o modele test setinden
sorular sorular. Kendini burada test eder.

Original Veri Seti

Eğitim Seti

Test Seti

K-Katlı Çapraz Doğrulama (K fold Cross Validation)

Her zaman geçerli değildir. 100 veri varsa
%80-%20 bölünümünde 17 veri olduğu için
öğrenme gerçekleştiremeyebilir. 17 veride bu
yönteme başvurulmaz. Eğer fazla veri
varsa %80-%20 bölünümünde Holdout yöntemi
kullanılabilir.

Original Veri Seti

Eğitim Seti

Test Seti

1 2 3 4 5

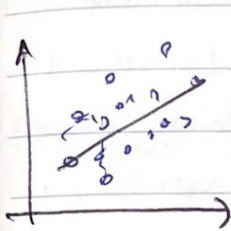
ÇAPRAZ DOĞRULAMA

Örneğin eğitim verisi 5 parçaya bölünür. Kendi içerisinde $\frac{1}{4}$ oranında test ve eğitim verisi olarak ayrılır. Kendi içinde eğitim ve test yapar. Orandaki test için bölünen diğer veriler ile test yapar.

Yanlılık - Varyans Denge Tutusu

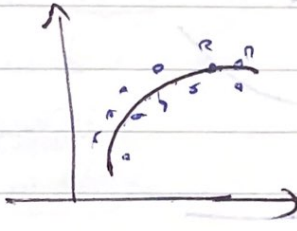
(Bias - Variance Tradeoff)

Pratikte yaygınca karşılaşılan aşırı öğrenme konusunu ele alıyor olacağız.



Underfitting

(Yüksek Yanlılık)



Doğru Model

(Düşük Yanlılık
Düşük Varyans)



Overfitting

(Yüksek Varyans)

Overfitting Nedir?

(ezberlenmesidir)

Aşırı öğrenmedir. Modelin veriyi öğrenmesidir. Model veriyi değil model içerisindeki örüntüyü, yapıyı öğrenmektedir. Aşırı öğrenme verinin ezberlenmesidir.

Underfitting Nedir?

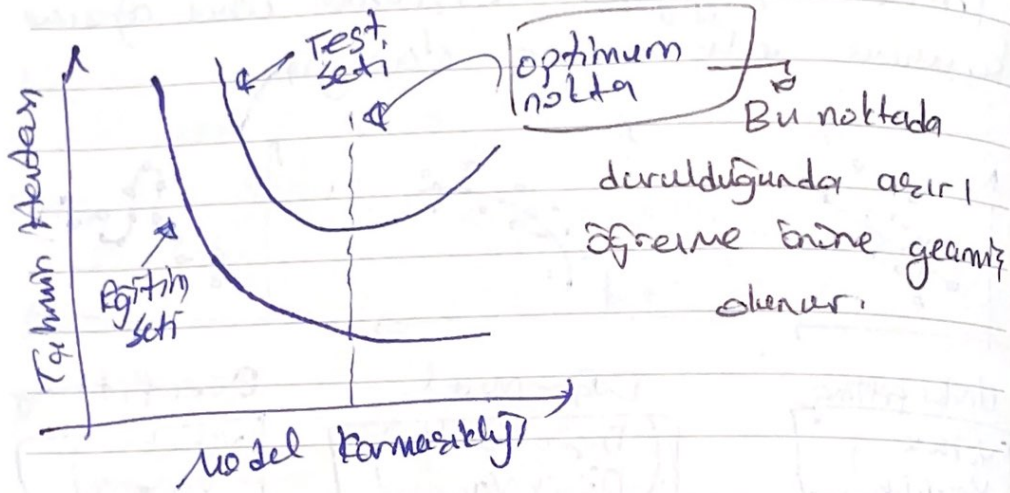
Modelin veriyi öğrenememesidir.

Yüksek Varyans $\Rightarrow$ Değişkenlik fazla, çok iyi öğren

Yüksek Yanlılık $\Rightarrow$ Bazı gözlemlere daha yakın, daha yakın, genellenebilirlik kabiliyeti katanmamış

Azırı Öğrenme Nasıl Yakalanır?

Eğitim ve test seti incelenir. Bu iki verinin birbirinde ayrılmaya başladığı noktada azırı öğrenme başlamıştır denir. Dolayısıyla o noktayı yakalayabiliyorsa nasıl önüne geçebileceğimizi de gösteririz. Modeli burada durdurursak azırı öğrenme önüne geçebiliriz.



Model Kurmak Ne Demek?

Bağımlı ve bağımsız değişkenler arasındaki ilişkiyi, ilişkinin özetini çıkarmak demektir. Bu ilişkiyi öğrenmek demektir.

Model Karmaşıklığı Nedir?

Bunun cevabı her modelde farklı olacaktır.

DOĞRUSAL YÖNTEMLERDE modelle istediğimiz terimleri eklemek, yani modeli hassaslaştırmak, yani modeli daha detaylı tahminler yapabilir hale getirmek demektir.

$$y = b + wx$$

te x yerine x^2, x^3 yazdığımızda ortaya çıkan durumlar.

AĞACI YÖNTEMLERİNDE, Dallanma sayısını artırarak karmaşıklığı etkiler.

Karmaşıklıkla kötü değildir. Bir süreye kadar hataları düşürüyor olacaktır. Ancak bir noktadan sonra train setini çok iyi öğrendiğinden dolayı fazla karmaşıklıklaştırmamak gerekir.

Doğrusal Regresyon (Linear Regression)

Bağımlı ve bağımsız değişkenler arası ilişkisi doğrusal bir şekilde modellemeye çalışır.

$$\hat{y}_i = b + w \cdot x_i$$

model bu sabitler ve ağırlıklardan oluşmaktadır.

$\hat{y}_i$: Bağımlı Değişken

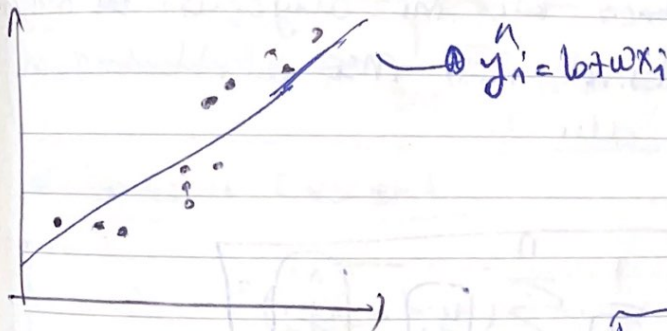
x_i : Bağımsız Değişken

b : Beta, bias, sabit

w : Katsayı, ağırlık

Ağırlıkların Bulunması

Graph'teki çizgiyi nereye kayaracağımıza karar verelimiz. Herhangi bir noktaya ne çok yakın ne de çok uzak olacak.



Doğruya nereye kayaracağımızı matematiksel formülle buluyoruz.

$$\Rightarrow \text{cost}(b, w) = \frac{1}{2m} \sum_{i=1}^m (\hat{y}_i - y_i)^2$$

MSE'dir. Tek fark 2'dir.

Gerçek değer ile tahmini değerler birbirinden çıkarılmış ve kareleri alınmıştır. ve ortalaması alınmıştır.

Örnek Bias ve weight'a göre 80 metre
kare evin fiyatını hesaplayınız.
 $b=210$, $w=4$

$$\hat{y}_i = b + w \cdot x_i \rightarrow 210 + 4 \cdot 90$$
$$\boxed{\hat{y}_i = 570}$$

Sabit (bias) ve ağırlık (weight) değişince
fiyatlar değişecektir. Dolayısıyla hatalar
değişecektir. Hata hesaplamak $(e_i = y_i - \hat{y}_i)$
kullanırız. Bazen (-) bazen de (+) hata
çıkabilir.

Regresyon Modellerinde Başarı Değerlendirme (MSE, RMSE, MAE)

Yukarıda $e_i = y_i - \hat{y}_i$ sonucu eksili veya
artılı sonuç bulmuştuk. Ortalama hatayı
bulmak için hepsini toplayıp kaç tane varsa
ona böleriz. Bu bir yaklaşım tabii ki işin
değerler olduğu için sonuç sıfır da çıkabilir. Bu
ifadeyi çıkartınca karesini alıyoruz. Bu işlem
problemimizi gidermek için MSE çıktıktan
sonra kare alır.

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

Hata
Kareleri
Ortalama

Görüş
Sayısı

Gercek
Değer

Tahmin
Edilen
Değer

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

Her bir kareler
Ortalamanın
karekökü

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

Parametrelerin Tahmin Edilmesi (Ağırlıkların Bulunması)

Yapılacak olan işlem bana en küçük ~~olan~~ hatayı verecek olan sabiti ve ağırlığı bulmaktır. Belirli (w) ve (b) kombinasyonlarını karşılaştırmak hatanın ne olabileceğini gözlemleyip bu sabit ve ağırlığı optimum değerleriyle yani mse'yi en küçük verecek şekilde seçmeye çalışmaktır. (w : ağırlık, b : sabit)

$$MSE = cost(b, w) = \frac{1}{2n} \sum_{i=1}^n ((b + wx_i) - y_i)^2$$

Gradient Descent

$$\theta_0 := \theta_0 - a \cdot \frac{1}{n} \sum_{i=1}^n (h_0(x^{(i)}) - y^{(i)})$$

($i=$) Yeniden atama

$$\theta_1 := \theta_1 - a \cdot \frac{1}{n} \sum_{i=1}^n (h_0(x^{(i)}) - y^{(i)}) \cdot x^{(i)}$$

θ_0 : b , sabit, w_0

θ_1 : b_1 , w_1

#1 Doğrusal Regresyon için Gradient Descent

makine öğrenmesinde bağımsız olarak bir optimizasyon yöntemidir. Bunun amacı bir fonksiyonun minimum yapabilecek parametre değerlerini bulmaktır. Bu bir optimizasyon yöntemidir. Şu şekilde gerçekleşir!

İlgili fonksiyonun ilgili parametreye göre kısmi türevini alır. Bu sonuca göre güncellenir.

Elde edilen türev, dişe itadesi ile gradian değeri ilgili fonksiyonun maksimum artış yönünü verir. Dolayısıyla ilgili fonksiyonun maksimum artış yönünü veren gradyanın tersine doğru belirli bir siddet ile giderek parametrenin eski değerinde değişiklik yaparak her iterasyonda hatanın azalmasını sağlar.

$$\theta_j \leftarrow \theta_j - \alpha \cdot \frac{\partial}{\partial \theta_j} J(\theta)$$

Gradyanın negatifine doğru giderek ilgili parametreyi günceller.

Gradyanın negatifini olarak tanımlanan "en dik iniş" yönünde iteratif olarak parametre değerini güncelleyerek ilgili fonksiyonun minimum değerini verebilecek parametreleri bulur.

Cost fonksiyonunu minimize edebilecek parametreleri bulmak için kullanılır.

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

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^m (\underbrace{h_\theta(x^{(i)})}_{\text{Tahmin edilen de\u011ferler}} - \underbrace{y^{(i)}}_{\text{Ger\u00e7ek De\u011fer}})^2$$

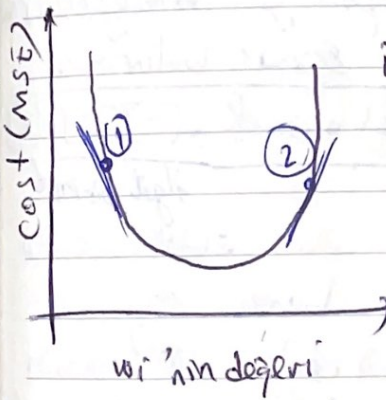
minimize $J(\theta_0, \theta_1)$
 θ_0, θ_1

T\u00fcrev sonucu bu gelir.

$$\theta_0 := \theta_0 - \alpha \left[\frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) \right] \rightarrow \theta_0 \text{'a g\u00f6r\u00fc k\u00fcmt\u00fcrev}$$

$$\theta_1 := \theta_1 - \alpha \left[\frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) \cdot x^{(i)} \right] \rightarrow \theta_1 \text{'e g\u00f6r\u00fc k\u00fcmt\u00fcrev}$$

\u00d6nce k\u00fcmt\u00fcrevler ile fonksiyonun maksimum artış $\hat{x}$ bulunur. Tesme g\u00fc\u00e7ler minimum noktaya gideriz.



① noktada e\u011fim negatif oldu\u011fu i\u00e7in toplama sonucu sağa kayar ve mse azalır.

② noktada e\u011fim pozitif oldu\u011fu i\u00e7in \u00e7ıkartma i\u015lemi yapılır ve sağa kayar, mse azalır.

$$\theta_0 := \theta_0 - \alpha \left[\frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) \right]$$

eski de\u011fer

eski de\u011feri parametresi (de\u011ferme oranı)

Gradyan de\u011feri

Kullanılacak K\u00fct\u00fchaneler

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.model_selection import train_test_split, cross_val_score
```

Model Kurmak

Bir veri setinden iki sütun alalım ve x, y olarak iki ~~data~~ parametreye atayalım.
Ardından model oluşturalım.

① $\text{reg-model} = \text{linearRegression}().\text{fit}(x, y)$
Matematiksel olarak $\hat{y} = b + w * x$ 'ti.

Parametreler görelim.

→ # Sabit (b - bias) için ($\text{bias} = \text{beta} = \text{intercept}$)

② $\text{reg-model.intercept} = [0]$

→ # İlgili yerm katsayısı (w) için ($\text{teta} = \text{rotasyon} = \text{rotas.} = \text{coef}$)
katsayı

③ $\text{reg-model.coef} = [0][0]$

model kurunca elimizde b ve w olacak.

Tahmin

Yukarıdaki kodlar ile sabiti ve ağırlığı elde ettiğimizde göre tahmin yapmaya başlayabiliriz.

④ Belirlediğimiz birime göre sonuç buluruz.

$\text{reg-model.intercept} = [0] + \text{reg-model.coef} = [0][0] * x$
İlgili birim

→ Verinin özetini öğrendiğimiz için elde ettiğimiz veride olmayan bir değer de olsa bunun sorgusu yapılabilir. Veri setinde olmayan değeri bile tahmin edebiliriz. $\text{df.describe}()$ ile verinin özetini görebiliriz.

Tahmin Başarısı

MSE: Bana gerçek değerleri ve tahmin edilen değerleri verir. Bunlara işlem uyguladım. Elimizde gerçek değerler var ama tahmin edilen değerler yoktur.

$$(5) \quad y\text{-pred} = \text{reg-model.predict}(x)$$

↳ Bağımsız değişkenleri verip bağımlı değişkenleri tahmin etmesini bekleriz. Ardından mse'ye gerçek ve tahmin edilen değerleri gönderirsek ortalama hatayı verecektir.

$$(6) \quad \text{mean-squared-error}(y, y\text{-pred})$$

↳ Minimum olduğuna dairmesini bekleriz. Bu hatanın iyi mi kötü mü olduğuna bakmak için:

$$(7) \quad \begin{matrix} y.\text{mean}() \\ y.\text{std}() \end{matrix} \text{ 'ye bakarız.}$$

Büyükliği neye göre değerlendirmen gerektiğini bilmediğimiz durumda bağımlı değişken ortalamasının bakarız $\Rightarrow y.\text{mean}() \pm y.\text{std}()$ ile kıyaslarız.

(RMSE) $(\text{MSE})^2$ 'dir.

$$(8) \quad \text{np.sqrt}(\text{mean-squared-error}(y, y\text{-pred}))$$

$$(MAE) \quad (9) \quad \text{mean-absolute-error}(y, y\text{-pred})$$

$$(R\text{-KARE}) \quad (10) \quad \text{reg-model.score}(x, y)$$

↳ Verisetindeki bağımsız değişkenlerin bağımlı değişkeni açıklama yütüdesidir.

R-kare ile ilgili

1) Değişken sayısı arttıkça R-kare sismeye meyillidir. Burada düzeltilmiş R-kare değerinin de göz önünde bulunulması gerekir.

Çoklu Doğrusal Regresyon Modeli

Örneği gibi bir tane değişken de birden fazla değişken olacak ve bütün veriyi modellemiş olacağız.

Bağımsız Değişken = X ,

Bağımlı Değişken = Y olarak atayıp model kuruyoruz.

X -train, X -test, y -train, y -test = train-test-split

$X, y, \text{test-size} = 0.20, \text{random-state} = 1$

80'li 1.20'si test seti
80'li 1.80 train seti

Ardından modeli eğitime aşamasına geçiyoruz.

(11) reg-model = LinearRegression(X_train, Y_train)

Sabit (b-bias) ← sabit

(12) (reg-model.intercept_)

coefficients (w-weights) ← katsayılar

reg-model.coef_

SORU :

Aşağıdaki gözlem değerlerine göre satış beklentisi değeri nedir?

TV = 30

radio = 10

newspaper = 40

2.90

0.04, 0.17, 0.002

a) model Denklemini yaz

$$\text{SALES} = 2.90 + \text{TV} * 0.04 + \text{radio} * 0.17 + \text{newspaper} * 0.002$$

↳ Değerleri yerine yazınca tahmin edilen satış elde edebiliriz.

ör Fonksiyon için yermat istesek:

13) yeni-veri = [[30], [10], [40]]

yeni-veri = pd.DataFrame(yeni-veri).T

reg = model.predict(yeni-veri)

↳ listeye göre tahmin yapmasını sağlar.

Tahmin Başarı Değerlendire

Train Setine ilişkin:

RMSE:

$$y_pred = \text{reg_model.predict}(X_train)$$
$$\text{np.sqrt(mean-squared-error}(y_train, y_pred))$$

TRAIN R2ARE $\text{reg_model.score}(X_train, y_train)$

↳ Bağımsız değişkenlerin bağımlı değişkeni etkileme, açıklama oranıdır.

↳ 3 farklı değişken çalıştığımız için hata düşecektir.

• YENİ DEĞİŞKEN ELENİNCE BAŞARI ARTAR, HATA DÜŞER

Doğrusal Regresyonu Gradient Descent ile Yapma

Ana Amaç : MSE'yi minimuma getirmeye çalışıyoruz. Bias değerleri ile oynayarak hatanın en düşük olduğu noktaya gitmeye çalışıyoruz.

Lojistik Regresyon (Logistic Regression) $f(x|y)$

Bir sınıflandırma problemi için bağımlı ve bağımsız değişkenler arasındaki ilişkiyi doğrusal olarak modellemektir.

$$\hat{y}_i = \frac{1}{1 + e^{-z}}$$

$$z = b + w_1x_1 + w_2x_2 + w_3x_3 + \dots + w_px_p$$

↳ Doğrusal regresyonun aynısı fakat buradan çıkan sonucu Sigmoid fonksiyonunda yatıyor olacağız. Çünkü işlem sonucu tahmin edilen sonuç 0-1 arasında olur ve bunu belirli bir eşik değere göre değerlendirebiliriz.

Çeşitli değerler ile tahmin edilen değerler arasındaki farklara ilişkin log loss değerini minimum yapabilmek için ağırlıkları bularak gerçekleştirilir.

$$\text{log loss} = \frac{1}{n} \left(\sum_{i=1}^n -y_i \cdot \log(p(\hat{y}_i)) - (1-y_i) \cdot \log(1-p(\hat{y}_i)) \right)$$

↓
Bağımlı değişkenin
bir sınıfın
gerçekleşme
olasılığı

Lojistik Regresyon için Gradient Descent

$$J(\theta) = \frac{1}{n} \sum_{i=1}^n \text{cost}(h_{\theta}(x^{(i)}), y^{(i)})$$

Tahmin
edilen
değerler

Gerçek
değerler

$$J(\theta) = \frac{1}{n} \left[\sum_{i=1}^n -y^{(i)} \cdot \log(h_{\theta}(x^{(i)})) + (1-y^{(i)}) \log(1-h_{\theta}(x^{(i)})) \right]$$

Gerçek değerler ile gerçek değerlerin gerçekleşmesi olasılığı ($h_{\theta}(x^{(i)})$) birbirine ne kadar yakınsa bu durumda loss (kayıp) değeri o kadar küçük olacaktır. Ardından Gradient Descent hesaplanır.

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

↳ Bu güncellemeler sonucunda hatanın değeri gözlemlenir.

Sınıflandırma Problemlerinde Başarı Değerlendirmesi

~~Lineer~~ Regresyon problemlerinde başarı MSE ya da RMSE ile değerlendirilir.

Confusion Matrix (Karışıklık matrisi)

Elimizdeki veri setlerinde modellenen başarıya göre geçmedi önce ilgilendirdiğimiz probleme göre 1 ve 0 sınıfları var. ve daha sonra model kararını bizim tahmin edeceğimiz sınıf var. Bunların karşılaştırması yapılır.

ÖNEMLİ!

1'e 1 $\Rightarrow$ True, 0'a 0 $\Rightarrow$ True

Gerçek Sınıf	Tahmin Edilen Sınıf	
	Sınıf=1	Sınıf=0
Sınıf=1	True Pozitif (TP)	False Negatif (FN)
Sınıf=0	False Pozitif (FP)	True Negatif (TN)

Xanlık sınıfla pozitif işlem yapılmış

Doğru sınıfla negatif işlem yapılmış

Accuracy: Doğru sınıflandırma oranıdır. Doğru yaptığı işlemlerin toplamı

$$(14) \quad \text{ACCURACY} = \frac{(TP + TN)}{(TP + TN + FP + FN)}$$

Precision: Pozitif sınıf (1) tahminlerinin başarı oranıdır. Tahmin ettiğimizde ne kadar başarıyla doğru tahmin yapıldığını gösterir.

$$(15) \quad \text{PRECISION} = \frac{TP}{TP + FP}$$

Örn: Sahte olarak tahmin edilenlerin gerçekte ne kadar sahte olduğu tahmini

Recall: Pozitif sınıfın (1) doğru tahmin edilme oranıdır.

$$(16) \quad \text{RECALL} = \frac{TP}{TP + FN}$$

Örn: Gerçekte sahte olarak işlemi olan işlemlerin ne kadarını doğru tahmin ettiğimizi gösterir.

Precision'da bir işlemde sahtekarlık yok ama sahtekarlık var demiyse oluruz FP demiş olduğumuz. Bu durumda yanlış alarm üretmiyoruz. FP, yanlış alarm üretmek demektir. Bu I. Tip hatadır. Gerçekte sahtekar olmayanları sahtekar dedik.

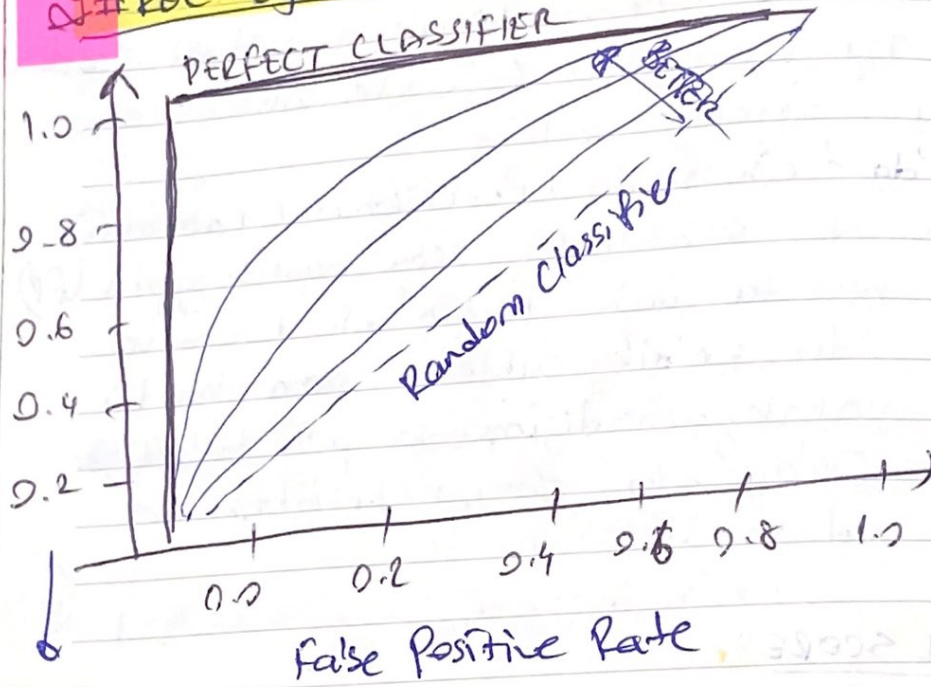
Recall'da - Öncesinde (Precision'da) sahtekarlık olması da sahtekarlık deme hatası yaptık (FP) fakat burada daha kritik bir durum var. Burada da FN ile sahtekar olan bir işlemi yakalayamadığımızda para kaybediyor oluruz. Dolayısıyla durum kritiktir ve II. Tip hatadır.

F1 SCORE

Precision değeri tahmin başarısına, Recall değeri gerçekleri yakalama başarısına odaklanmıştır. F1 SCORE, Precision ve Recall değerlerinin harmonik ortalamasıdır.

$$F1 \text{ SCORE} = 2 \cdot \frac{(PRECISION * RECALL)}{(PRECISION + RECALL)}$$

.....
Eğer değeri, classification threshold değeri
değer bütün istatistikler değişiyor olacak.
ROC Eğrisi (ROC Curve)



True
Positive
Rate

Yukarıda eğriyi görüyoruz.
Bu eğride altta kalan alan
ne kadar fazla alan kaplıyorsa
o kadar modelin başarılı olduğunu
anlamına gelir.

ROC eğrisinin altında kalan alanın
integrali alırsa bu alana Area
Under Curve (AUC) denir. Eğri altında
kalan alan demektir.

Area Under Curve (AUC)

ROC eğrisinin tek bir sayısal değer
ile ifade edilebilir.

ROC eğrisinin altında kalan alandır.

.....
AUC, F1m gibi sınıflandırma ölçütleri için
toplu bir performans ölçüsüdür.

Veri seti değiştiğinde de değişir de olsa
F1 SCORE'ye bakıyoruz.

LOG Loss

LOG Loss bir başarı metriğidir. Bunu hem
modelin başarısını hem de optimizasyon
yapmak üzere gerek hiperparametrelerde,
ağırlıklarda optimizasyon yapmak üzere kullanır.

$$\text{Log Cross} = \frac{1}{n} \left(\sum_{i=1}^n -y_i \cdot \log(p(\hat{y}_i)) - (1-y_i) \log(1-p(\hat{y}_i)) \right)$$

Geri Deger Tahmini Deger

Log Cross entropi metriğidir. Entropi, ne kadar
düşük o kadar iyidir. Ne kadar yüksek o
kadar kötüdür.

Yol Haritası Şu Şekildedir.

- 1) Problem Veri Seti Hazırlama
- 2) Kesitici Veri Analizi
- 3) Veri Ön İşleme
- 4) Modelleme
- 5) Model Başarısını Değerlendirme
- 6) Model Doğrulama
- 7) 10 Kattı Çapraz Doğrulama

Lineer Regresyon sürekli bir çıktı verirken,
Logistik regresyon sabit bir çıktı üretir.

19) Problem Veri Seti Hkayesi

Bu aşamada verisetinin ne olduğu
anlaşılmalı ve anlaşılır. Değişkenler tanınır,
Veriseti tanınır. İlgili kütüphaneler import
edilir. Kendi tanımladığımız fonksiyonlar
eklenir.

- outlier-thresholds
- check-outlier
- replace-with-thresholds

2) Kesifai Veri Analizi

Veriseti okunur ve incelenmeye başlanır.
Target analizi ile hangi sınıflara kaç tane
olduğunu inceleriz ve görselleştirebiliriz.

- 190) `value-counts()` gibi
- `describe().T` gibi → sadece sayısal verileri gösterir
- `plot- numerical-col` fonk. tanımlanır

3) Veri İnceleme

Eksik değerler incelenebilir. (`isnull()` `sum()`)
Aykırı değerlere bakılır. (`check-outlier` fonksiyonu)
Var olan aykırı değerleri eşit değerlerle değiştiririz
- `replace-with-thresholds`
Değişkeni ölçeklendiririz. Modellerde eşit
faktörler için. Kullanılan parametre değerlerinin
daha hızlı ve daha doğru şekilde bulunması
için gerçekleştirilir.

- 20) `RobustScaler()`: Aykırı değerlerden etkilenmez
Her bir gözlem biriminde ortalamayı değil median'ı
alır.

4. Modelleme

Amacımız bağımsız ve bağımlı değişkenler arasındaki ilişkiyi modellemektir. Bağımsız ve bağımlı değişkenler ayrı ayrı ifade edilir. Modelleme böyle ister. Ardından ikisi arası modelleme yöntemi ile ilişki kurulur.

21) $\log_model = \text{LogisticRegression}().\text{fit}(x,y)$
model sabitleme işlemek istenir.

22) $\log_model.\text{intercept_}$
 $\log_model.\text{coef_}$

Regresyon modelini göğünip direceğiz ki al ser en gözlem biriminin değerleri bir tahminde bulun.

23) $y_pred = \log_model.\text{predict}(x)$
tahmin et.

5. Model Başarı Değerlendirme

Gerçek değerler ve tahmin edilen değerlere dair bir başarı değerlendirilmesi yapılabiliriz. Karmaşıklık matrisi gösterilir. Bir fonksiyon şeklinde isı haritası ile birlikte eklenmiştir.

24) $\text{plot_confusion_matrix}$

Accuracy: Doğru sınıflandırma oranı

Precision: Doğru olarak yapılan tahminlerin başarı oranı

Recall: 1 olanları ne kadar başarılı tahmin ettik

F1 SCORE: Bunlar üzerinden harmonik ortalama ile F1 SCORE hesaplanır.

Örnek: Doğrudan Gibi Netiye

1 oluk 1000
değerler 1.744
bazen 1.744

1 sinif
göre değerlendirilir

0.80
10.74

accuracy
macro avg
weighted avg

0.77
0.78

Accuracy and
odds her sonuç
1 satır oluklar

her 0.2 her
1.744 içinde
bulunduran istedi
aritmetik ort
değerleri ort
var.

1 oluk 1.58 bazen 1.6
değerler siniflandırılır

recall f1-score support
0.89 0.84 500
10.08 0.65 268

0.74
0.78

10.78
0.75
0.78 768
768

Doğru siniflandırma
oran değeri

Ardından ROC eğrisine bakarız.

23) $y_prob = \text{log_model.predict_proba}(X)[1,1]$
 $\text{roc_auc_score}(y, y_prob)$

↳ 1 sınıfının gerçekleme oranına bakarız.

6) Model Doğrulama (Holdout)

Bu yaklaşımın çalışma şekli şöyledir: Verisetini iki parçaya böl, birisi ile modeli eğit, diğeri ile modeli test et seklindedir. Bu şekilde veriseti bölünür.

26) $x_train, x_test, y_train, y_test = \text{train_test_split}(x,$

x , bağımsız

y , bağımlı değişkendir ve 1.20 - 1.80

olarak verisetini böldük.

$y,$
 $\text{test_size}=0.20,$
 $\text{random_state}=17)$

27) Ardından modeli kuruyoruz.

$\text{log_model} = \text{LogisticRegression}().\text{fit}(x_train, y_train)$

train ile modeli kurduk. Test ile modele sormalyız.

~~log_model~~

28) $y_pred = \text{log_model.predict}(x_test)$
tahmin et!

Tahmin edilen değerlerle birlikte bir sınıfın ait olma olasılıklarını da hesaplayalım.

29) $y_prob = \text{log_model.predict_proba}(x_test)[1,1]$

↳ Bağımsız değişken değerleri gönderdik. Buna göre elimizde olasılık değerleri gelecek.

Sıra başarı değerleridir ne de.

30) $\text{print}(\text{classification_report}(y_test, y_pred))$

.....
5. Adım ile 6. Adımı kıyaslayacak olursak:
5. Adımda modeli sadece train ile eğitirken
6. Adımda modeli train ve test ile eğitilir.
Çokta farkta olmasa da ufak farklılıklar
olduğu model test veri seti ile görmediği
sonuçlara dayandığında tahmin sonucu
değişir gibi gözüküyor. Böylece model
doğrulama işlemi gerektiği sonucu çıkarılabilir.

ROC grafiğini görüntüleriz. Burada AUC
değerini elde ederiz.

(31)

```
plot-roc-curve(log-model, x-test, y-test)
plt.title("ROC curve")
plt.plot([0,1],[0,1], "r--")
plt.show()
# AUC
roc_auc_score(y-test, y-prob)
```

Buradan şöyle bir kıyaslama yapabiliriz.

5. Adımda hesaplanan AUC score ile 6. Adımda
hesaplanan AUC değeri farklıdır. 6. Adımda
daha yüksektir.

random-state değeri ile aynıysa farklı
sonuçlar elde ederiz. Öyle bir işlem yapmamız
lazım ki bu olan farklılıklar da göz
önünde bulundursun. On kati kopya
doğrulama yöntemini kullanmamız lazım.

7 Adım

10 Katlı Çapraz Doğrulama

6. Adımda değeri değiştirildikçe doğruluğun değiştiğini gördük. Öyle bir işlem yapmalıyız ki modelin doğrulama sürecini en doğru şekilde ele almış olalım. Bunun yolu da Çapraz Doğrulama işlemidir.

Bunun anlamı; Veriseti 10 parçaya bölünür. 9 parçası ile model kurulur 1 parçası ile test yapılır. Daha sonra farklı (9)-(1) oranları ile model kurulur ve test yapılır ve bu on katlı olarak yapıldıktan sonra bütün test hatalarının, validation hatalarının ortalaması alınır. Böylece model verisetinin farklı parçaları ile test edilmiş olduğundan bazı senaryolar ile karşılaşır ve ortalama alınınca tüm senaryolara yönelik bize bir fikir veriyor olur. Böylece on katlı bir şekilde doğrulama işlemleri gerçekleştirilmiş olur.

12. $y = df["Outcome"]$

$x = df.drop(["Outcome"], axis=1)$

$log_model = LogisticRegression().fit(x, y)$

$cv_results = cross_validate(log_model,$

$x, y,$

5 katlı cross
validation yapar

$cv=5,$

$scores = ["accuracy", "precision", "recall", "f1", "roc_auc"]$

2. Ardından her birinin ortalamasına bakalım.
33. Son sağlıklı sonuç görebiliriz.

```
cv_results["test-accuracy"].mean()  
cv_results["test-precision"].mean()  
cv_results["test-recall"].mean()  
cv_results["test-f1"].mean()  
cv_results["test-roc-auc"].mean()
```

On katlı aşırı doğrulama yaptığımız için artık daha doğru bir oran elde ediyoruz.

K-En Yakın Komşuluk (K-Nearest Neighbors) (KNN)

Görüntülerin birbirlerine olan benzerlikleri üzerinden tahmin yapar.

Bana arkadaşımı söyle sana kim olduğunu söyleyeyim.

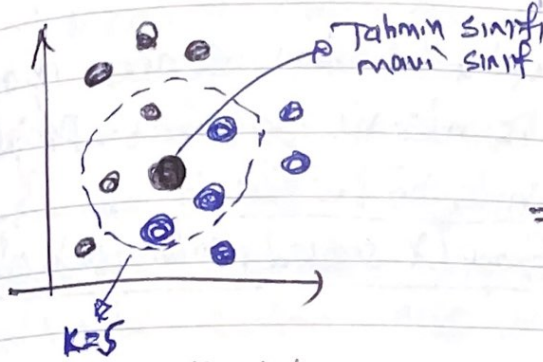
Bir gözetim biriminin kendine en yakın olan diğer gözetim birimleri hesaplanır (k adet) ve bu diğer kendisine yakın k adet gözetim biriminin bağımlı değişkeni neyse bunun üzerinde ilgili gözetim için tahminde bulunulmuş olur.

Öklid ya da başka bir uzaklık hesabı ile her bir gözetim uzaklığı hesaplanır.

$$\sqrt{\sum_{i=1}^k (x_i - y_i)^2}$$

K-En Yakın Komşu Sınıflandırma Problemi

En yakın k -adet komşuluğun toplayıp adet sayısına böleriz yani ortalamasını alırız. Eğer 1 ve 0'lerden oluşuyorsa en çok tekrar eden ortalamayı olarak kabul edeceğiz çünkü ortalamayı en çok tekrar edene yakın olacak.



En yakın k -adet gözlemin y -değerlerinin en sık gözlenen frekansı tahmin edilen sınıf olur.

Mavi = 1
⇒ Sıkak = 0 ve $k=5$
derset ortadaki
gözlem birimi sınıfı
mavi frekandan olduğu
için 1'dir.

1 # Kesirli Veri Analizi (EDA)

Bu aşamada veriyi okuduktan sonra;

- df.head()
- df.shape
- df.describe().T
- df["Outcome"].value_counts(),

2 Veri Ön İşleme (Standardizasyon yaparız.) bakabiliriz.

KNN yöntemi uzaklık tabanlı bir yöntemdir. Uzaklık tabanlı yöntemlerde ve Gradient Descent temelli yöntemlerde değişkenlerin standart olması elde edilecek sonuçların ya daha hızlı ya da daha doğru olmasını sağlayacaktır. Eğer bir ifade ile daha başarılı olması sağlanabilecektir.

.....
Bu sebeple elimizdeki bağımsız
değişkenler standartlaştırma işlemine sokulur.

(34)
$$\begin{aligned} y &= df["Outcome"] \\ x &= df.drop(["Outcome"], axis=1) \\ x_scaled &= StandardScaler().fit_transform(x) \end{aligned}$$

Standartlaştırma
yaptık.

X-scaled çıktılarından sonra np.array
döner ve sütun isimlerini taşıması. Aşağıdaki
işlem ile sütun isimlerini ekleriz.

(35)
$$X = pd.DataFrame(X_scaled, columns=X_columns)$$

3. Modelleme

Bu bölümde KNN modelimizi oluşturacağız.

(36) ~~Knn model~~

$$knn_model = KNeighborsClassifier().fit(X, y)$$

Veriden rastgele bir kullanıcı seçelim
ve kurulan (36)'daki modele diyabet olup
olmama ihtimalini soralım. a random bir satır al.

(37)
$$\begin{aligned} random_user &= X.sample(1, random_state=45) \\ knn_model &= predict(random_user) \end{aligned}$$

Modeli fit etmek, train etmek ayrı bir
süreçtir. Train edilen modelle kullanarak
tahminlerde bulunmak ayrı bir süreçtir.

4. Model Başarı Değerlendirme

Az önce tek bir gözlem için tahmin yaptık. Şimdi knn-model'i kullanarak bütün gözlem birimleri için tahmin yapıp bunları bir yerde saklamamız lazım önce.

$$(38) \quad y_pred = knn_model.predict(X)$$

↳ Bunun üzerinde confusion matrix hesaplanır. Bunun üzerinde de accuracy ve f1 gibi metrikler hesaplanıyor olur.

Bir de AUC hesaplanabilir. Bunu da olasılık değerleri üzerinde yapabiliriz. ROC eğrisi altında kalan AUC hesaplamak için:

ROC eğrisini de classification thresholdları değiştirerek, farklı olasılıklara göre bu farklı olasılıklar üzerinde TP ve FP'lere göre ROC eğrisini oluştuyorduk ve altında (eğrim) kalan alandan AUC hesaplanır.

$$(39) \quad y_prob = knn_model.predict_prob(X)[1]$$

↳ AUC için y-prob

↑ signifying
all other olasılıkları
gettiyoruz.

Madem karmaşıklık matrisi için tahmin edilen değerler elimizde hemen classification report methodunu getirerek hesaplama işlemi gerçekleştirelim.

$$(40) \quad print(classification_report(y, y_pred))$$

Precision
Accuracy
F1 score
Recall

değer tablosunu verir

...../.....
Accuracy değeri başarılı sınıflarımızın oranıdır. $1/83$ accuracy olsa bu, 100 bireyi 83'üne doğru tahmin ediyor demektir. Ancak her zaman doğru bilgi veremezdir. Dağılımı varsa başka metriklerle de bakmak lazım. Bu metrikler Precision (1) olarak tahmin ettiklerimizin başarıları.

RECALL: Gerçekte (1) olanları (1) olarak tahmin etme başarıları

F1 SCORE: Bunların harmonik ortalaması

AUC skoru için:

(1) roc-auc-score(y, y-prob)

Yüksek çıkarsa
çok iyi

Gerçek
Değer

y-lerde yer alan (1) sınıfın
asıl olma olasılıklarını
ifade eden
y-prob

Yukarıya kadar modeli kurduğumuz
veride test ettik.

Fakat asıl yöntem modeli görmediği
veride test etmektir.

Bunun için iki yol vardır.

(1) Holdout Yöntemi (Sinema Seti Yaklaşımı)

↳ $1/80$ de $1/20$ gibi veriyi train ve test olarak bölmektir

(2) Cross Validation, Holdout'un ortaya çıkardığı
cegi bazı sınıflardaki dezavantajı gidermek için önerildi
↳ Örneğin 5 kere tekrar doğrulama

5 katlı ya da 10 katlı ağırlar doğrulama yöntemi ile hatamızı değerlendirmek istiyorsak

```
cv_results = cross_validate(knn_model,  
                             X, y,  
                             cv=5,  
                             scoring=["accuracy", "f1", "roc_auc"])
```

model nesnesi

Bağımsız, Bağımlı Değişkenler

5 katlı Ağırlar Doğrulama Yaparken

Kullanmak istediğin metrikleri ver.

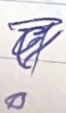
Burada 5 katlı ağırlar doğrulama yaptığımız için 5 farklı analiz yapar ve 5 farklı değer verir. Bunların ortalamasını alacağız.

```
cv_results["test-accuracy"].mean()  
cv_results["test-f1"].mean()  
cv_results["test-roc_auc"].mean()
```

↳ Buradan direkt değerler çıktı.
Buradan ne anlamalıyız?

Modeli kurduğumuz veriyi, modelin performansını ~~doğru~~ değerlendirmek için kullandığımızda, ortaya bir miktar yanlılık çıkar. Bu yanlılık sonuçları doğru değerlendirmemizi engeller.
Peki Başarı Skoru Nasıl Arttırılabilir?

- 1) Veri boyutu arttırılabilir.
- 2) Veri ön işleme işlemleri detaylandırılabilir.
- 3) Özellik mühendisliği, yeni değişkenler türetilir.
- 4) İlgili algoritma için optimizasyonlar yapılabilir.



④ maddeyi biraz açıklayalım. KNN yönteminin bir dısal parametresi var. Komşuluk sayısının hiperparametresi. Bu parametre değiştirilebilir. Parametreleri görmek için;

④④ `knn-model.get_params()`

↳ `n-neighbors`: 5 olduğu görülür.

Parametre: Modellerin veri isimden öğrendiği ağırlıklardır. Özetle veri içerisinde öğrenilmektedir.

Hiperparametre: Kullanıcı tarafından tanımlanması gereken, dısal ve veri seti içerisinde öğrenilemeyen parametrelerdir.

⑤ Hiperparametre optimizasyonu

Kullanıcının discordan eklemesi gereken parametreler vardır. Özellikle KNN parametreleri vardı. Onu ve ön tanımlı parametrelere bakalım

④⑤ `knn-model = KNeighborsClassifier()`
`knn-model.get_params()`

↳ Buradan komşuluk sayımızı 5 olarak görüyoruz. Şimdi amacım bu komşuluk sayısını değiştirecek olmanın gerekten en optimum komşuluk sayısını bulmak. Şu şekilde

④⑥ `knn-params = {"n-neighbors": range(2, 50)}`

↳ `knn-model.get_params()` ile isimlendirme ile aynı oluyor

↳ 2'den 50'ye kadar sayılar oluşturduk. Bu sayıları tek tek deneyip en iyi değeri bulmaya çalışacağız

range(2, 50) aralığındaki en değer sonucu
hergi konsult verecek? Onu arıyoruz.
Bunun için:

(47) `kn-gs-best = GridSearchCV ( knn-model, knn-params, cv=5, n_jobs=-1, verbose=1 )` fit(x,y)

Bana modeli ver
Parametreleri ver
rapor bekliyorum
Bu ayrı hiperparametre için yapılan cross validation farklıdır.

1290'dan cross validation'ın ara demektir.
2-50 arasındaki tüm değerleri dener.

Çalıştırınca ekrana bir bilgi dözer.
Bu dener parametreler için 5 katlı cross validation yapılacağından ve range(2, 50) arası olduğundan dolayı 48 değer $\times 5 = 240$ tane fit etme işlemi yapılır.
En iyi model kurma işlemi

İşlemleri olan en yüksek performansı kullanır.

Buradan sonra GridSearchCV kullanarak KNN algoritması için optimum konsult sayısının ne olduğunu bulduk. Şimdi en iyi parametreyi çağıralım

(48) `kn-gs-best.best_params_`

↳ En iyi konsult sayısını verir (v)
↳ Başlangıç parametresi 5 idi. Şimdi bu en iyi konsult sayısından tekrar model kurarsak bu modelin başarısının daha iyi olması beklenir.

16. Final model

Elimizdeki KNN algoritmasının en iyi hangi parametrelerle çalışacağını bulduk. Bunu model abnormalizit. Daha öncesinde onun en tanımlı değerini bulmuştuk. Tutar model kuralım.

49 ~~knm = knn_model.set_params(**knm_gs.best_params_)~~
knm-final = knn_model.set_params(**knm_gs.best_params_)

Ardından modeli 'fit(x_{iy})' yzdr. Normalde key-value çifti ile yazmalıydık fakat birde forlar parametre olduğunda tek tek yazmamız gerekmekteydi. ** ile direkt atayabiliyoruz.

Şimdi ise bu final modelin test hatasını bakmamız gerekiyor.

50 `cv_results = cross_validate(knm-final,
X,
y,
cv=5,
scoring=["accuracy", "f1", "roc-auc"]
cv_results["test-accuracy"].mean()
cv_results["test-f1"].mean()
cv_results["test-roc-auc"].mean()`

→ KNN algoritması hem sınıflandırmada hem de regresyon problemlerinde kullanılabilir.

→ KNeighborsClassifier (n-neighbors=7)

↳ Gözlem birimlerini sınıflandırmak için kullanılarak kime sayısını ifade eder. Arttıkça hata da artar.

Classification and Regression Tree

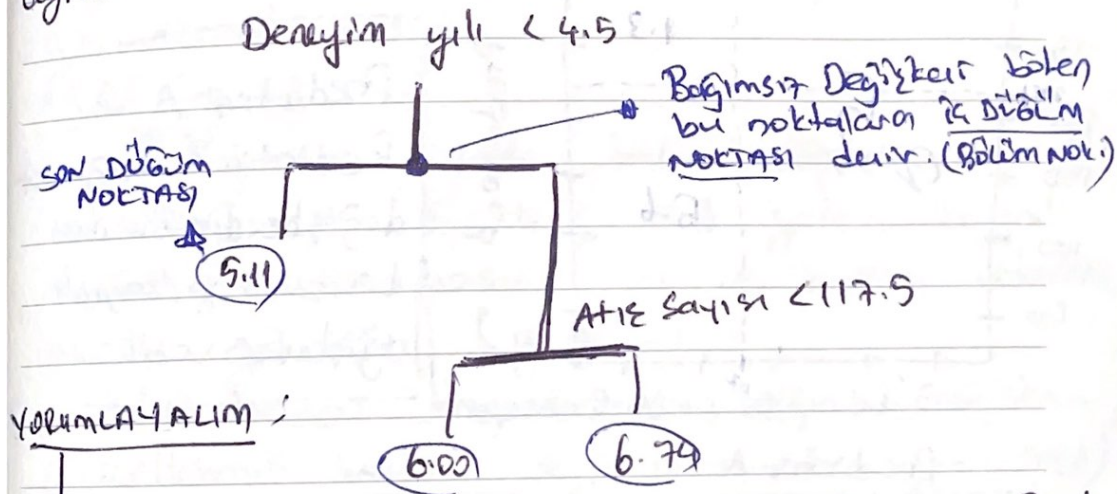
(CART)

(Karar Ağacı Yöntemi)

Random Forest'in temelini oluşturur.

AMACI: Veri seti içerisindeki karmaşık yapıları basit karar yapılarına dönüştürmektir. Bütün karar ağaçlarının aslında temel amacı budur.

Heterojen bir veri seti belirlenmiş bir hedef değişkere göre homojen alt gruplara ayrılır.



- Deneyim yılı ve Atış sayısı Bağımsız Değişken
- Değerler Bağımlı Değişken
- Deneyim Yılı < 4.5 ise 5.11, değilse Atış Sayısı < 117.5 'e bakılır.

↳ Bir Karar Ağacı Şemasında en tepede görülen değişken en önemli değişkendir.

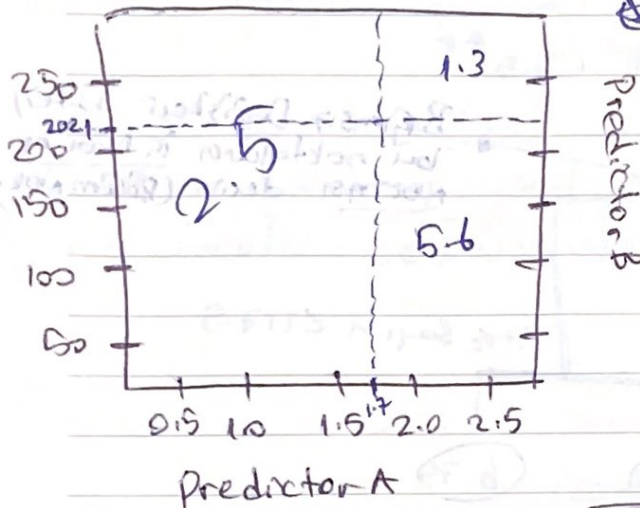
↳ Bu ağacın iki tane 15 bölüm noktası vardır.

↳ Bu ağacın iki tane de son düğüm noktası vardır.

...../...../.....
Regresyon problemlerinde bu dallanma işlemi, derin sarma kalan bölgelerin ortalaması alınarak bu nihai tahmin sonucu olarak verilir. (örneğin 1.3 veya 5.6)

Kod kısmına bakalım

```
if Predictor A >= 1.7 then  
|   if Predictor B >= 2.021 then Outcome = 1.3  
|   else Outcome = 5.6  
else Outcome = 2.5
```



Temel Karar Kuralı

Predictor A ve Predictor B bağımsız değişkenlerdir. Bu ilişki keskinliğinde bağımlı değişkenler vardır.

EXCEL'de örneklendirelim. X_1 ve X_2 bağımsız değişken ise Y bağımlı değişken olur. Bu bir regresyon problemi olsun. X_1 yani Predictor A eğer 1.7'den düşük olması durumunda bağımlı değişkenin ortalaması alınacak. X_1 ve X_2 bağımsız değişkenlerinin sınırlarının Y bağımlı değişkendeki ortalaması alınıp yazılır.

Karar ağaçları heterojen veri setini bu şekilde bölerek ~~karar~~ tahmin sonuçları elde eder.

Görüntü işlemede Derin Öğrenme, Ağaç Algoritmalarında da bu

Regresyon Problemleri için Cost / Loss / Objective Fonksiyonu

Bu ağacı nasıl oluşturmamızın sonucu RSS veya SSE (dijer ifadesi)'dir. Bu değerin minimum olduğu noktalardaki bölgenmeler, yaprak bölgeler kutulanmalardır.

$$\text{RSS (SSE)} = \sum_{j=1}^J \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2$$

(HATA) Gerçek Değerler Tahmin edilen Bölge'deki Değerler

: R_j Bölge Yaprak kutu

Tahmin edilen değerler, bölme işlemleri neticesinde ortaya çıkan kutularda kalan gözlem birimlerinin ortalamasıdır.

OLAY ŞU ŞEKİLDE :

- 1) Bağımsız değerleri belirli değerde böl (x)
- 2) Bağımsız değerlere karşı gelen bağımlı değerin (y) ortalamasını al ve hepsini ortalama değere güncelle (y-pred)
- 3) Ardından ~~bağımlı~~ bağımlı değişkeni güncellenmiş bağımlı değişkenler çıkar. (y-y-pred)
- 4) Bu sonuçların karesini al
- 5) Toplamını toplar. Bu değer SSE'dir.
↳ Hata kareler Toplamı

SSE en düşük olduğu noktayı seçmeliyiz. En iyi bütün noktaları veren noktayı seçmeliyiz. En düşük hatayı veren değişken en önemli değer olup en tepede yer alır. Bu önem düzeyine göre değişkenlerin dallanmaları gerçekleştirir.

$$MSE = \frac{\text{Minimum SSE}}{\text{Görlem Birimi Sayısı}}$$

$$RMSE = \sqrt{MSE}$$

$$MAE \text{ (mutlak Artık Ortalaması)} = \frac{|y - y_{pred}|}{\text{Görlem Birimi Sayısı}}$$

Birinci dallanmayı tamamladıktan sonra (SSE minimum olduğu yerden kesildi) veri iki farklı veri olacak düşünelim. Ardından aynı işlem tekrar tekrar yapılır. Çeşitli noktalardan bölünür ve bu bölme işlemi sonucu en küçük hatayı veren noktaya göre bir kez daha dallanacaktır. Böylece bölme işlemiyle belirli bir derinliğe kadar gidilecek olacaktır.

Burada problemler oluşur

(min Sample Size) 1) Ne kadar dallanacağız?

(max Depth) 2) Nereye, ne kadar bölünmeye devam edeceğiz?

Ağac yöntemleri çok basardır. Overfit (Aşırı Öğrenme) 'e çok meyillidir çünkü çok iyi öğrenirler. Aynı anda aynı anda en dişe kadar inerler. Dezavantajı, genellebilirlik kabiliyetini kaybetmeleridir, aşırı öğrenme popüler olmasıdır.

Sınıflandırma Problemleri için Cost/Loss/Objective Fonksiyonu

Bir sınıflandırma problemi için Gyny ve Entropi kullanılabilir.

$$L_G(N_m) = \sum_{k=1}^K \hat{p}_{mk} (1 - \hat{p}_{mk}),$$

$$L_E(M_E) = - \sum_{k=1}^K \hat{p}_{mk} \log \hat{p}_{mk}$$

Benzersiz değerleri gibi düşünebiliriz. Ne kadar düşük o kadar iyidir.

Ağırlıklı şekilde $\Rightarrow L(S_m) = f_1 \cdot L(C_m^L) + f_2 \cdot L(C_m^R)$ kayıp hesaplanır.

Dalların Sol
Tarafları

Dalların Sağ
Tarafları

ENTROPİ: Ne kadar düşük o kadar iyidir. Çünkü entropi gesihtililik demektir. Gesihtililiğin az olmasını bekleriz.

CA2D Kütüphane Kurulumları

- (51) pip install pydotplus
- pip install skompiler
- pip install astor
- pip install joblib

Kurulumdan sonra çalışma dosyası yeniden başlatılmalı.

Kütüphaneler

(52)

```
import warnings
import joblib
import pydotplus
import numpy as np
import pandas as pd
import seaborn as sns
from matplotlib import pyplot as plt
from sklearn.tree import DecisionTreeClassifier, export_graphviz,
export_text
from sklearn.metrics import classification_report, roc_auc_score
from sklearn.model_selection import train_test_split,
GridSearchCV, cross_validate, validation_curve
from skompiler import skompiler
```


53) ~~Gösterim ile ilgili.~~
pd.set_option("display.max_columns", None)
warnings.simplefilter(action="ignore", category=Warning)

#modelleme

Verisetimizin yskledikleri sonra bağımlı (y) ve bağımsız (x) değişkenleri seçeriz. Ardından modeli oluşturmaya başlayabiliriz

54) cart_model = DecisionTreeClassifier(random_state=1).fit(x,y)

↳ CART modeli kuruldu.

55) y_pred = cart_model.predict(x)

↳ Confusion matrix için y_pred (Karmaşıklık matrisi)

56) y_prob = cart_model.predict_proba(x)[i, 1]

↳ AUC için y_prob (ROC eğrisi için birinci sınıfa ait olma olasılığı)

57) print(classification_report(y,y_pred))

↳ Confusion matrix için

58) roc_auc_score(y,y_prob)

↳ AUC score için

57 ve 58'de değerler 1.0 (Y 100)

çıkarsa kusulanmıyız. Acaba adam

öğrenmeye mi destek?

Holdout Yöntemiyle Başarı Değerlendirme

(1.0) elde etsek bile bunu doğrulamaya

59) ihtiyacımız vardır.

x_train, x_test, y_train, y_test = train_test_split(X,

Y,

test_size=0.30

random_state=17)

Train setiyle model kurduk

60) Şimdi ise modeli kuralım.

```
cart_model = DecisionTreeClassifier(random_state=17).fit(X_train,
```

Bu noktada:

y_train)

1) Train hatasına bakabiliriz

2) Test hatasına bakabiliriz

Train Hatası için

61) `y_pred = cart_model.predict(X_train)`

`y_prob = cart_model.predict_proba(X_train)[ :, 1]`

`print(classification_report(y_train, y_pred))`

`roc_auc_score(y_train, y_prob)`

Eğer hala (1.0) almaya devam ediyorsa kuskular devam ediyor. Test setine de bir bakalım

`X_train` → `X_test` olarak değiştirmek yeterli.

Model eğitildiği veride çok iyi sonuç verebilir ancak aynı sonuç test verilerinde gözlemlenemeyebilir.

Eğer train setinde (1.0), test verisinde (0.57) gibi birbirinden alakasız değerler gözleniyorsa (accuracy, f1, recall değerleri) bu model aşırı öğrenme olmuş diyebiliriz. Hiç görmediği veriyi gördüğünde basarsın diye.

Aşırı öğrenme olduğu için.

Bu problemleri kaçabilmek için yapraz dağıtma yapmamız lazım.

CV (Yapraz Dağıtma) ile Başarı Değerlendirme

Buradan elde edilecek sonuçlar en doğru, en

2) Sağlıklı sonuçlar olacaktır.

```
cart_model = DecisionTreeClassifier(random_state=17).fit(X,y)
```

```
cv_results = cross_validate(cart_model, X,y,
```

`cv=5,`

`scoring=["accuracy","f1","roc_auc"])`

5 katlı yapraz dağıtma

63

Ardından skorlara bakalım

```
cv_result["test-accuracy"].mean()
```

```
cv_result["test-f1"].mean()
```

```
cv_result["test-roc-auc"].mean()
```

Şimdi Neler Yaptık? (Olası Senaryolar)

① → Bütün veriyi modellemek için kullanıp bütün veriyi test ettiğimizde (1.0) sonucunu aldık. Çok başarılıyız. (!)

② → Bu işte bir test seti var. O yüzden doğrulamamız lazım.

③ → Daha sonra train ve test olarak ayırıp Holdout yöntemine geçtik. Burada da train seti tekrar (1.0) çıktı. Bu da garip. Test hatalarına bakınca çok düşektir çıktı. Başka bir 1.70-1.30 dedüğümüzde de hatalar değişti. Veriseti az olduğundan Holdout sevdeleri. Normalde

✓ veri seti yeterli kadar zengin olduğunda sadece direkt train-test ayrımı da tercih edilebilir.

④ → Ardından ~~train~~ test kullandığımız doğrulamayı yapalım dedik. Ardından elde ettiğimiz hatalar en doğru, en geçerli hatalardır.

Baradaki değerlendirme kriteri başarının yetersiz değildi. Doğrulama taygımız var. Literatürde de adı model validation'dur. Amacımız da bu zaten. Elde edilen hata daha doğru elde etmek. Daha büyük başarı elde etmek ya da daha düşük hata elde etmek değil.

Hiperparametre Optimizasyonu

mevcut modelin parametrelerini şu şekilde

öğreniriz $\Rightarrow$ `cart_model.get_params()` (64)

AYARLANACAK PARAMETRELER

`min-samples-split` : a tane kalma
OVERFIT'e sebep olabilir.

durumuna göre bölme
işlemine karar verin.

`max-depth`

Bu iki değer OVERFIT'e doğrudan etki eder.

Bu iki değer ile oynayarak AŞIRI ÖĞRENMEYEN kurtaklılık.

(5) `cart_params = { "max-depth": range(1,11),
"min-samples-split": range(2,20) }`

• Bu değerleri ön tanımlı parametrelerin
• etrafındaki değerleri denemeliyiz.

Daha sonra GridSearchCV methodunu getirerek bu parametre

(6) bere göre arama yapılır.

`cart_best_grid = GridSearchCV ( cart_model,`

modeli göster

Hangi parametreleri
gezetmek olduğunu
göster

`cart_params,`

`cv=5`, kaç katlı
kopyalar doğrulamak

`n_jobs=-1,`

`verbose=True).fit(X,y)`

İzlenimin tam
performans
kullanma ayarı

Raporlama
yap

Su anaya
`score="f1"`

• Ya da belki
• bir sorgulama
• yapabiliriz

Toplamda iki hiperparametrenin kaç tane kombinasyonu
olduğunu raporlaması yapar.

En iyi parametreleri görüntülemek için:

(7) `cart_best_grid.best_params =`

En iyi değerlere karşı en iyi skoru belirtmek için:

(8) `cart_best_grid.best_score =`

Arama işlemleri ile bulunan en iyi parametreler GridSearchCV metodu bir modeli hazır eden (fit eder) Onu da (cart-best-grid)'e kaydedtik. Ardından random bir veri seti uygulamayı yapalım.

(69) `random = X.Simple(1, random_state=45)`
`cart_best_grid.predict(random)`

Yani GridSearchCV metodu; en iyi parametreleri vermek için kullanacağımız metoddur.

Final Model

Daha önceden alışık olduğumuz için direkt yapar
(70) `cart_final = DecisionTreeClassifier(**cart_best_grid.best_params_)`
↳ model kuruldu.

Bu hiperparametre değerleri ile kullandığımız `fit(X,y)`
(71) `cart_final.get_params()`
↳ Parametreleri kontrol ederiz.

Bir diğer parametreleri modele atamanın yolu şudur.

(72) `cart_final = cart_final.set_params(**cart_best_grid.best_params_)`

↳ (70)'teki ile aynı iş. Var olan model üzerine set_params yaparak parametreleri günceller. Ardından cross-validate hatalarını gözlemleyebiliriz. (62)'deki gibi. Tek fark cart_model yerine cart_final yazacak olmamız.

Buraya kadar sadece modellerin en iyi şekilde, en performanslı şekilde nasıl kullanıldığını görüyoruz. geliştirilmeler, optimizasyonlar yapıyoruz.

Özellik Önerisi (Feature Importance)

Hangi değişkenin hangi önem skoruna sahip olduğunu görmek istiyoruz. Öncelikle özelliklerin önem sırasına göre sıralamak istersek

(73) `cart - final.feature_importances =`

↳ Bu bize sadece sayısal değerleri önem sırasına göre verir.

(74) `def plot_importance(model, features, num=len(X), save=False):`

Bu fonksiyon ile önem sırasına ~~göre~~ göre verilen sayısal değerleri anlamlandırılır ve barplot ile grafiği verilir. (önem düzeyine göre)

Öğrenme Farklılıklarıyla Model Karmaşıklık Analizi Etme (BONUS)

Model karmaşıklığı arttıkça:

→ Train seti hatası düşer

⁴
Bütün testler
başarılı olabilir.

→ Test seti hatası da bir noktaya kadar

düşükler sonra tekrar artmaya başladığı gözlemlendi.

Train ile test setinin ayrılmaya başladığı noktada OVERFIT'e düşen noktadır.

Görselleştirme

conda install graphviz

import graphviz

• Bunları indirmeli

(75) Kullanmayı görselleştiririz. fonksiyon ile yapılır.

`def tree_graph(model, col_names, file_name):`

`tree_str = export_graphviz(model, feature_names=col_names,
filled=True, out_file=None)`

`graph = pydotplus.graph_from_dot_data(tree_str)`

`graph.write_png(file_name)`

(75) numaralı fonksiyonla ağaç yöntemi parametrelerine göre dallanma görseli yazdırarak png olarak kaydedecek. Bu fonksiyonu çağıracağız.

(76) `tree-graph (model=cart-final, col-names=X.columns, file-name="cart-final.png")`

↳ `cart-final` modelin ağaç görselini `cart-final.png` olarak dosyaya kaydeder. Üzerinde çalışmış olduğumuz modelin görsel çıktısını verir.

(77) `cart-final.get-params()` 'daki `max-depth=5` olduğu için 5 tane dallanma işlemi olacaktır.

ÖRNEK
GİSTERİM

Glucose değeri
küçüktür birliğe sınırlanmış
bu noktadan bölünce
en düşük gmi'yi
vermiş.

Glucose <= 127.5
gmi = 0.454
samples = 768
value = [500, 268]

768 değer var.
112'yi sınıfı
268 tane var.

True (500 tane)
False

Diye DEVAM EDER!

Karar kuralları

Hemen yukarıda blok şeklinde olan karar kurallarını konsolda görüntülemek için

(78) `tree-rules=export-text (cart-final, feature-names=list(X.columns))`
`print (tree-rules)`

Dallanmalardan sonra tüm değişkenler göz önünde bulundurulur.

Karar Kurallarının Python Kodlarını Çıkartma

Kurulan herhangi bir karar ağacı yönteminin kurallarını çıkartıp bir SQL sorgusu haline getirebilirsek aslında database seviyesinde bir model deployment yani modeli canlıya alma işlemini tamamlamış oluruz.

sklearn "0.23.1" versiyonu ile yapılabilir

pip install scikit-learn==0.23.1

↳ Böyle olmalı yoksa kodlar çalışmaz -
Versiyon sorgulamak için

79 `import sklearn`
`sklearn.__version__` → Versiyon sorgusu yapılabilir.

Görsel tekniklerle elde ettiğimiz karar ağacının fonksiyonlaştırılabilir olan karar kurallarını görmek istersek;

80 `print(skompile(cart_final.predict).to("python/code"))`

↳ yani model derlenen şeyi python'da kuradan elde edilen kodlarla çalıştırılabilir olacaktır.
SQL kodları için:

Öncelikle → `pip install sqlalchemy` 81

81 Sonra;

`print(skompile(cart_model.predict).to("sqlalchemy/sqlite"))`

↳ SQL kodlarını verecek. Çok basit şekilde.

↳ Bu kod sonrası yeni bir gözetim birimi kaydı veritabanına düşüyor andı bu kodu çalıştırdığımızda ilgili kişi ile ilgili tahmini çıkarır.

EXCELL kodları için

82) `print(skompile(cart-final-predict).to("excel"))`

Python kodları ile Tahmin işlemi

Az önce skompile ile elde ettiğimiz Python kodları ile tahmin yapmaya geçelim. Bu noktada bir fonksiyon oluşturabiliriz.

83) `def predict-with-rules(x):
 return skompile...`

↳ skompile ile elde ettiğimiz

Ardından columns gidelim Python kodları

`X.columns`

Değer atayalım:

`x = [12, 13, 20, 23, 4, 55, 12, 7]`

`predict-with-rules(x)`

0,1 çıktı

↳ Bu bize tahmini verecektir.

Modelleri Kaydetme ve Gagırma

İlgili işlemleri yaptığımız cart-final'i dosya olarak kaydetmek için:

84) `joblib.dump(cart-final, "cart-final.pkl")`

↳ Bu artık bizim modelimiz. Bu modeli gagırıp

85) test etmek için;

`cart-model-from-disk = joblib.load("cart-final.pkl")`

`x = [12, 13, 20, 23, 4, 55, 12, 7]`

`cart-model-from-disk.predict(pd.DataFrame(x).T)`

↳ Ardından çıktı verir.

predict, DataFrame ile gelir.

Rastgele Ormanlar

Random Forest

(Random Forests)

1) Temeli birde çok karar ağacının ürettiği tahminlerin bir araya getirilerek değerlendirilmesine dayanır.

2) Bagging ile Random Subspace yöntemlerinin birleşimi ile oluşmuştur.

3) Ağaçlar için gözlemler bootstrap rastgele örnek seçim yöntemi ile değişkenler random subspace yöntemi ile seçilir.

4) Karar ağacının her bir düğümünde en iyi dallara ayırıcı (bilgi kazancı) değişken tüm değişkenler arasından rastgele seçilen daha az sayıdaki değişken arasından seçilir.

Rastgeleliği Nereden Alıyor?

1) Gözlem birimlerinden rastgele seçerek n tane ağaç oluşturur.

2) Oluşturduğu ağaçlarda bölme işlemine başlamadan önce değişkenlerin içerisinde rastgele değişken seçer.

3) Ağaç oluşturmada veri setinin $2/3$ 'ü kullanılır. Dışarıda kalan veri ağaçların performans değerlendirilmesi ve değişken önemini belirtenmesi için kullanılır.

(fit etmede $2/3$,
test etmede $1/3$ 'ü kullanılır)

4) Her düğüm noktasında rastgele değişken seçimi yapılır.

(regresyonda $p/3$,
sınıflandırmada $\sqrt{p}$ 'dir.)

.....
Tek başına olan CART yöntemini 'gagaltan'
Bagging Yöntemi ortaya çıkarmıştır. Bagging
yöntemindeki bu rastgelelikte gözetlediği
başarı ile Random Subspace bulunmaktadır.

■ Gözetimlerde rastgele seçimler yaptım ve
yeni ağaçlar kurdum, Random Subspace
yöntemi ile değişkenlerde de rastgele
seçimler yaparsa bu da başarılı olur diye
düşünmüştür.

■ Gözetimlerde ve Değişkenlerde rastgelelik
kavramını ortaya getirerek Rastgele
Ormanlar yöntemini ortaya çıkarmıştır.

CART çok başarılı ancak Azırı Öğrenme
ye meyilli olduğu için bu yöntemle ihtiyas
duyulmuştur.

* Bagging Yöntemi ? Elimizde m adet gözlem
birimi var. Bu m adet gözlem
birimini içerisinde n 'er adet ($n < m$) basit
Örnekleme Yöntemi ile ağaçlar oluşturulur.

TÖREMEĞİN 1000 gözlem var. Bu 1000
gözlemden rastgele 750 tane gözlem
(sıra) seçilir. Bununla bir tane ağaç
modeli kurulur. Daha sonra 1000
gözlem biriminden rastgele yeni bir
750 tane daha gözlem seçilir. Bunun
başka bir model kurulur. Burada

T adet karar ağacı modelinin ürettiği
 T adet tahmin değerini bir araya getirir.
ve değerlendirir. Bu Bagging Yöntemidir.

Ağaçlar birbirinden bağımsızdır. Birbirleri ile alakaları yoktur.

Bagging ile Boosting Farkı Nedir?

Bagging yönteminde ağaçların birbiri ile bağlantısı yoktur. Boosting yönteminde ise ağaçlar artıklar üzerine kuruludur. Dolayısıyla ağaçların birbirine bağımlılıkları vardır.

Yapılacak Kurulumlar

Random Forests, GBM, XGBoost, LightGBM, CatBoost

import warnings

import numpy as np

import pandas as pd

import seaborn as sns

from matplotlib import pyplot as plt

from sklearn.ensemble import RandomForestClassifier,

GradientBoostingClassifier, VotingClassifier

from sklearn.model_selection import GridSearchCV,

cross_validate, RandomizedSearchCV, validation_curve

from xgboost import XGBClassifier

from lightgbm import LGBMClassifier

from catboost import CatBoostClassifier

!pip install catboost

!pip install xgboost

!pip install lightgbm

pd.set_option("display.max_columns", None)

warnings.simplefilter(action="ignore", category=Warnings)

.....
y - bağımlı, x - bağımsız değişken

Random Forests

(87) Model oluşturma.

rf_model = RandomForestClassifier(random_state=)

rf_model.get_params()

İlgileneceğimiz Parametreler

max_features: Bölümlerde göz önünde
Veri setindeki ~~değişken sayısını~~ ^{bulundurulacak değişken sayı}
daha fazla olmaması.

max_depth: maximum derinlik

min_samples_split: Bir düğümün dallanması için
kaç tane düğüm olacağını ifade

n_estimators: Fit edilecek bağımsız
ağac sayısıdır.

Yukarıdaki parametreler önemlidir. Tek
tek denemekten önce GridSearch ile aramaya
başlarız. Değerleri de ön tanımlı değerlerin
etrafındaki değerler olarak belirtiriz.

(88)

```
rf_params = {"max_depth": [5, 8, None],  
             "max_features": [3, 5, 7, "auto"],  
             "min_samples_split": [2, 5, 8, 15, 20],  
             "n_estimators": [100, 200, 500]}
```

Ön tanımları da girilerek ki araştırmanın
en başında elde edilen hatanın daha kötü
hata elde etmemiş oluruz.

(89) (87) ile (88) arasında ağırlıklı olarak yaparız.

```
cv_results = cross_validate(rf_model, x, y, cv=10, scoring="roc_auc")  
cv_results["test-accuracy"].mean()  
cv_results["test-f1"].mean()  
cv_results["test-roc-auc"].mean()
```


Örneğin şöyle bir durum olabilir.

(1) Hiperparametre optimizasyonu yaptım ama öncesinden daha düşük hata aldım?

(v) Hiperparametre optimizasyonu sonrası elde edilen hatanın daha düşük olmasını bekleriz. Düşük olmamasının sebepleri:

1) Fastgelelit,

2) On tanımlı argümanları arama setinde bulundurmuyor olmaktan kaynaklanıyor olabilir.

90 Şimdi tarama yapalım.

`rf-best-grid = GridSearchCV(rf-model, rf-params,`
En iyi parametreler için: `cv=5, n-jobs=-1, verbose=True)`

91 `rf-best-grid.best-params-` `fit(x,y)`
Parametreleri güncellemek için

92 `rf-final = rf-model.set-params (**rf-best-grid.best-params,`
`random-state=17)`

Ardından yeni parametreler ile tekrardan değerlendirme yapılır. 89'daki formülde `rf-model = rf-final` olarak değiştirilecek.

93 Karşılaştırma yapılır.

`def plot-importance`
ve

`def val-curve-params` Fonksiyonları önemli.

Gradient Boosting Machines (GBM)

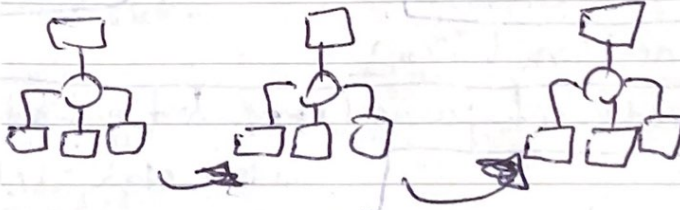
Artık optimizasyonuna dayalı çalışan bir ağaç yöntemidir. Ağaç yöntemlerine boosting veya Gradient Descent'in uygulanmasıdır.

AdaBoost (Adaptive Boosting)

GBM'in temelini diyebileceğimiz yöntemdir. Zayıf sınıflandırıcıların bir araya getirilerek güçlü bir sınıflandırıcı oluşturması fikrine dayanır.

Getelim GBM'i

- Hatalar/artıklar izleme tek bir tahminsel model formunda olan modeller serisi kurulur.



Boosting yöntemlerinin Gradient Descent ile optimize edilmesi işlenmiştir.

$$\boxed{\text{GBM} = \text{Boosting} + \text{Gradient Descent}}$$

Yani zayıf öğrenicileri bir araya getirerek güçlü bir öğrenici oluşturabilir miyim?

Ağaçlar tahminde bulunduktan sonra onların hatalarına odaklanarak bu hataların izleme gideret Boost edebilir miyim bu ağaçları? düşüncesi sonucu GBM çıkmıştır.

- Gradient boosting tek bir tahminsel model formunda olan modeller serisi oluşturur.

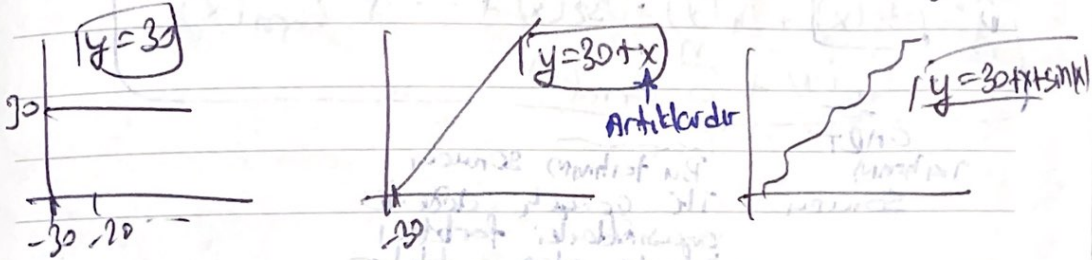
■ Seri içerisindeki bir model serideki bir önceki modelin tahmin artıklarının/hatalarının (residuals) üzerine kurularak (fit) oluşturulur.

■ GBM diferansiyellenebilen herhangi bir kayıp fonksiyonunu optimize edebilen Gradient Descent algoritması kullanılmaktadır.

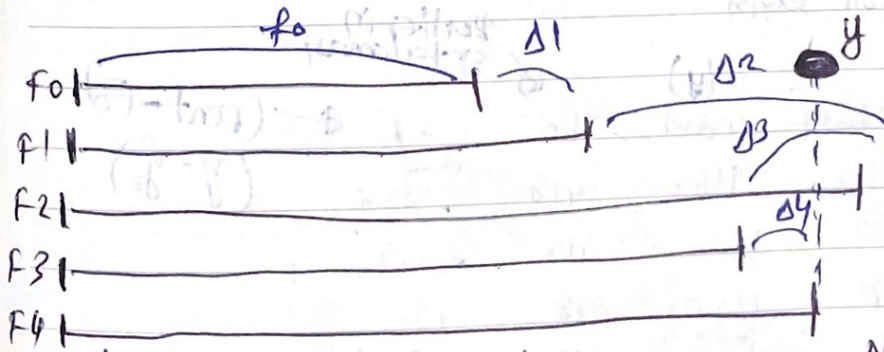
■ Tek bir tahminsel model formunda olan modeller serisi additive şekilde kurulur.

Additive Modeling (Eklendi Model)

Bu aşamada ayrılan bölgeleri ~~ekleyerek~~ daha hassas ayırmaya çalışacağız. Artıklar için yine gireceğiz.



Burada artıkları modelleyeceğiz. Daha hassas olan yaratmaya çalışıyoruz. tahmin sonucuna birşeyler ekleyerek veya çıkararak optimum sonuçlara gitmeye çalışacağız.



f_0 ilk modelimizdir, tahmin sonucumuz. $\Delta_1, \Delta_2, \Delta_3, \Delta_4$ ile seke seke y 'nin gerçek değerine yaklaşılmaya çalışılıyor.

Yani ilk kurulan model f_0 olacak.

Model: Belirli filtrelere göre ortalama değer

Yani 5 tane bile y gerçek değeri olsa hepsi ortalaması alındıktan sonra elde edilen değere eşitlenir. Bu bölündükten sonra belirli alanda kalan değerlerin ortalamasıdır. Bu ilk modeldir. Bu ilk modelden sonra hata elde edeceğiz (Δ) Bunu eski tahminlere kırıttan elde ettiğimiz tahmini çıkaracağız. Böylece y 'nin gerçek değerine yaklaşıyoruz.

→ CART'tan gelen tahmin sonuçları

$$\hat{y} = f_0(x) + \Delta_1(x) + \Delta_2(x) + \dots + \Delta_m(x)$$

$$= f_0(x) + \sum_{m=1}^m \Delta_m(x)$$

CART
Tahmin
Sonuç

Bu tahmin sonucu
ile gerçek değerler
arasındaki farkları
kırda eden artıklar.

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

$$f_m(x) = f_{m-1}(x) + \Delta_m(x)$$

$$f_1(x) = f_0(x) + \Delta_1(x)$$

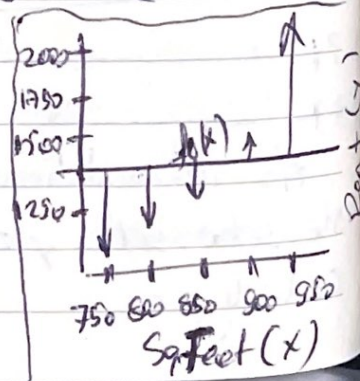
$$f_2(x) = f_1(x) + \Delta_2(x) = f_0(x) + \Delta_1(x) + \Delta_2(x)$$

iterasyon sayısı

rentlerin
ortalaması

Örnek

Sqfeet	rent	f_0	$y - f_0$	$(rent - f_0)$ $(y - \hat{y}_0)$
750	1160	1418	-258	
800	1200	1418	-218	
850	1280	1418	-138	
900	1450	1418	32	
950	2000	1418	582	



Rent (Y)

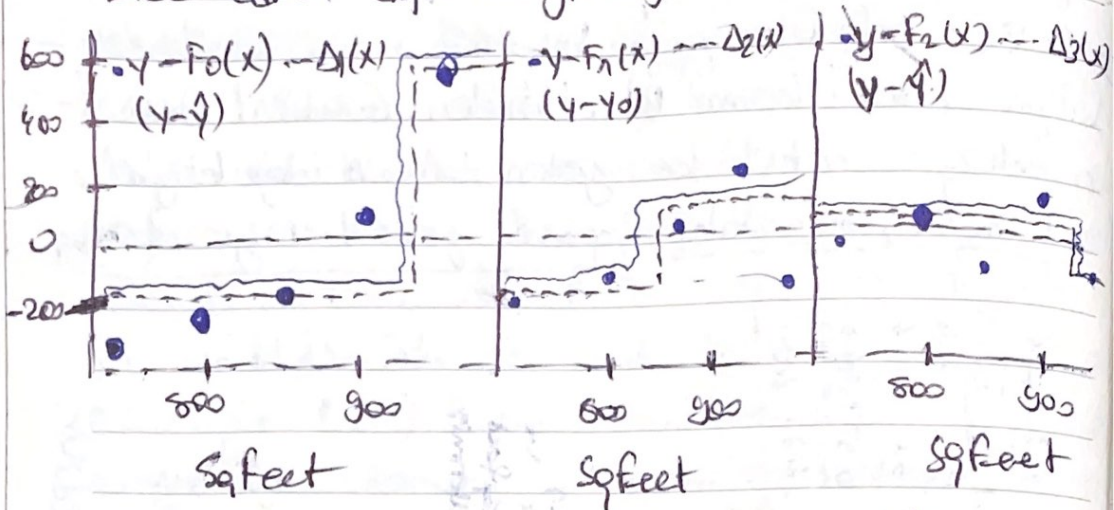
① $\rightarrow$ $\frac{1}{2}$ $\rightarrow$ $\frac{1}{4}$ $\rightarrow$ $\frac{1}{8}$ $\rightarrow$ $\frac{1}{16}$ $\rightarrow$ $\frac{1}{32}$ $\rightarrow$ $\frac{1}{64}$ $\rightarrow$ $\frac{1}{128}$ $\rightarrow$ $\frac{1}{256}$ $\rightarrow$ $\frac{1}{512}$ $\rightarrow$ $\frac{1}{1024}$ $\rightarrow$ $\frac{1}{2048}$ $\rightarrow$ $\frac{1}{4096}$ $\rightarrow$ $\frac{1}{8192}$ $\rightarrow$ $\frac{1}{16384}$ $\rightarrow$ $\frac{1}{32768}$ $\rightarrow$ $\frac{1}{65536}$ $\rightarrow$ $\frac{1}{131072}$ $\rightarrow$ $\frac{1}{262144}$ $\rightarrow$ $\frac{1}{524288}$ $\rightarrow$ $\frac{1}{1048576}$ $\rightarrow$ $\frac{1}{2097152}$ $\rightarrow$ $\frac{1}{4194304}$ $\rightarrow$ $\frac{1}{8388608}$ $\rightarrow$ $\frac{1}{16777216}$ $\rightarrow$ $\frac{1}{33554432}$ $\rightarrow$ $\frac{1}{67108864}$ $\rightarrow$ $\frac{1}{134217728}$ $\rightarrow$ $\frac{1}{268435456}$ $\rightarrow$ $\frac{1}{536870912}$ $\rightarrow$ $\frac{1}{1073741824}$ $\rightarrow$ $\frac{1}{2147483648}$ $\rightarrow$ $\frac{1}{4294967296}$ $\rightarrow$ $\frac{1}{8589934592}$ $\rightarrow$ $\frac{1}{17179869184}$ $\rightarrow$ $\frac{1}{34359738368}$ $\rightarrow$ $\frac{1}{68719476736}$ $\rightarrow$ $\frac{1}{137438953472}$ $\rightarrow$ $\frac{1}{274877906944}$ $\rightarrow$ $\frac{1}{549755813888}$ $\rightarrow$ $\frac{1}{1099511627776}$ $\rightarrow$ $\frac{1}{2199023255552}$ $\rightarrow$ $\frac{1}{4398046511104}$ $\rightarrow$ $\frac{1}{8796093022208}$ $\rightarrow$ $\frac{1}{17592186044416}$ $\rightarrow$ $\frac{1}{35184372088832}$ $\rightarrow$ $\frac{1}{70368744177664}$ $\rightarrow$ $\frac{1}{140737488355328}$ $\rightarrow$ $\frac{1}{281474976710656}$ $\rightarrow$ $\frac{1}{562949953421312}$ $\rightarrow$ $\frac{1}{1125899906842624}$ $\rightarrow$ $\frac{1}{2251799813685248}$ $\rightarrow$ $\frac{1}{4503599627370496}$ $\rightarrow$ $\frac{1}{9007199254740992}$ $\rightarrow$ $\frac{1}{18014398509481984}$ $\rightarrow$ $\frac{1}{36028797018963968}$ $\rightarrow$ $\frac{1}{72057594037927936}$ $\rightarrow$ $\frac{1}{144115188075855872}$ $\rightarrow$ $\frac{1}{288230376151711744}$ $\rightarrow$ $\frac{1}{576460752303423488}$ $\rightarrow$ $\frac{1}{1152921504606846976}$ $\rightarrow$ $\frac{1}{2305843009213693952}$ $\rightarrow$ $\frac{1}{4611686018427387904}$ $\rightarrow$ $\frac{1}{9223372036854775808}$ $\rightarrow$ $\frac{1}{18446744073709551616}$ $\rightarrow$ $\frac{1}{36893488147419103232}$ $\rightarrow$ $\frac{1}{73786976294838206464}$ $\rightarrow$ $\frac{1}{147573952589676412928}$ $\rightarrow$ $\frac{1}{295147905179352825856}$ $\rightarrow$ $\frac{1}{590295810358705651712}$ $\rightarrow$ $\frac{1}{1180591620717411303424}$ $\rightarrow$ $\frac{1}{2361183241434822606848}$ $\rightarrow$ $\frac{1}{4722366482869645213696}$ $\rightarrow$ $\frac{1}{9444732965739290427392}$ $\rightarrow$ $\frac{1}{18889465931478580854784}$ $\rightarrow$ $\frac{1}{37778931862957161709568}$ $\rightarrow$ $\frac{1}{75557863725914323419136}$ $\rightarrow$ $\frac{1}{151115727451828646838272}$ $\rightarrow$ $\frac{1}{302231454903657293676544}$ $\rightarrow$ $\frac{1}{604462909807314587353088}$ $\rightarrow$ $\frac{1}{1208925819614629174706176}$ $\rightarrow$ $\frac{1}{2417851639229258349412352}$ $\rightarrow$ $\frac{1}{4835703278458516698824704}$ $\rightarrow$ $\frac{1}{9671406556917033397649408}$ $\rightarrow$ $\frac{1}{19342813113834066795298816}$ $\rightarrow$ $\frac{1}{38685626227668133590597632}$ $\rightarrow$ $\frac{1}{77371252455336267181195264}$ $\rightarrow$ $\frac{1}{154742504910672534362390528}$ $\rightarrow$ $\frac{1}{309485009821345068724781056}$ $\rightarrow$ $\frac{1}{618970019642690137449562112}$ $\rightarrow$ $\frac{1}{1237940039285380274899124224}$ $\rightarrow$ $\frac{1}{2475880078570760549798248448}$ $\rightarrow$ $\frac{1}{4951760157141521099596496896}$ $\rightarrow$ $\frac{1}{9903520314283042199192993792}$ $\rightarrow$ $\frac{1}{19807040628566084398385987584}$ $\rightarrow$ $\frac{1}{39614081257132168796771975168}$ $\rightarrow$ $\frac{1}{79228162514264337593543950336}$ $\rightarrow$ $\frac{1}{158456325028528675187087900672}$ $\rightarrow$ $\frac{1}{316912650057057350374175801344}$ $\rightarrow$ $\frac{1}{633825300114114700748351602688}$ $\rightarrow$ $\frac{1}{1267650600228229401496703205376}$ $\rightarrow$ $\frac{1}{2535301200456458802993406410752}$ $\rightarrow$ $\frac{1}{5070602400912917605986812821504}$ $\rightarrow$ $\frac{1}{10141204801825835211973625643008}$ $\rightarrow$ $\frac{1}{20282409603651670423947251286016}$ $\rightarrow$ $\frac{1}{40564819207303340847894502572032}$ $\rightarrow$ $\frac{1}{81129638414606681695789005144064}$ $\rightarrow$ $\frac{1}{162259276829213363391578010288128}$ $\rightarrow$ $\frac{1}{324518553658426726783156020576256}$ $\rightarrow$ $\frac{1}{649037107316853453566312041152512}$ $\rightarrow$ $\frac{1}{1298074214633706907132624082305024}$ $\rightarrow$ $\frac{1}{2596148429267413814265248164610048}$ $\rightarrow$ $\frac{1}{5192296858534827628530496329220096}$ $\rightarrow$ $\frac{1}{10384593717069655257060992658440192}$ $\rightarrow$ $\frac{1}{20769187434139310514121985316880384}$ $\rightarrow$ $\frac{1}{41538374868278621028243970633760768}$ $\rightarrow$ $\frac{1}{83076749736557242056487941267521536}$ $\rightarrow$ $\frac{1}{16$

$\Delta y = y_2 - y_1$
 $(F_2 - F_1) \Delta y$
 Birinci order model
 yeni tabanlar

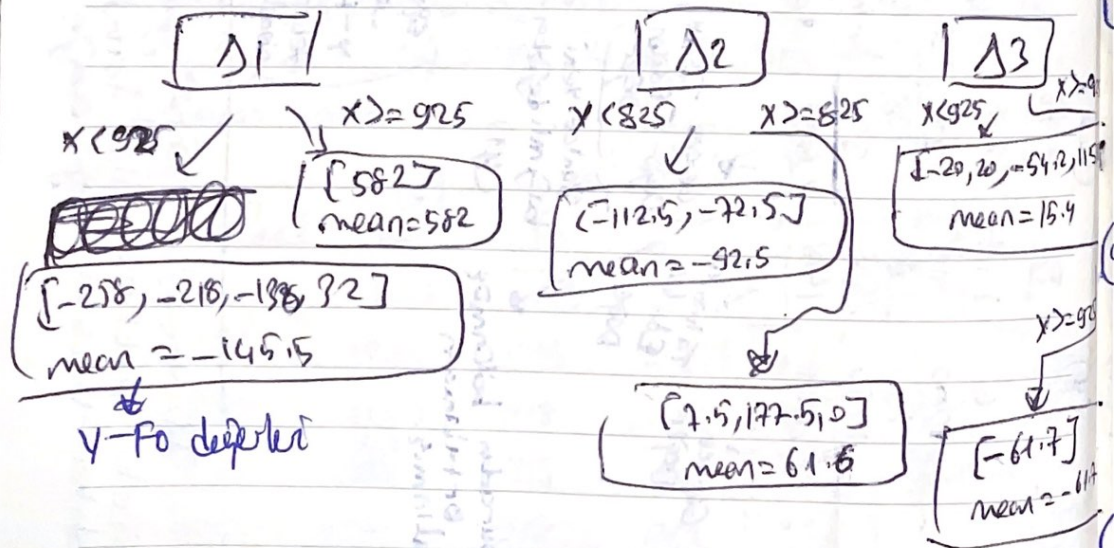
Buradan bölünmüş ve ortalaması alınmış.

Bu ktrafa dogru girdilice gesece daha da
cek yatkindir. 11r tahmin dogrum (1418) yeme
daha nasasa, gesece yakin dogerler tahmin edilir.

Elde edilen değerleri grafikte gösterelim



y -ekseninde bağımlı değişken yani artiklar var
 x -ekseninde bağımsız değişken var
 Amaç artikları (gerçek değerler tahmini değerler) minimize etmektir ki grafikte de (eğer sağ) görülebiliyor. Artiklar optimize edilerek küçültülmüş. Hatalar küçülmüş ve tahmin başarıları arttırılmış. Görsel ile de gösterilebilir istiyorsanız.



GBM uygulama

Modeli oluşturalım.

94) `gbm-model = GradientBoostingClassifier(random-state=17)`

Ardından parametrelerine bir göz atalım.

95) `gbm-model.get_params()`

Ua Gradient Descent temelli olduğumuz için

"learning-rate" var. (öğrenme oranı) Sabit tahminler
artıkla güncellenirken hızlar.

96) "n_estimators": Optimizasyon sayısıdır. Kaç
defa boost ettirmize kararlı gelir.

Hatalarımıza İnelik Bir Bakalım

`cv-results = cross_validate(gbm-model, X, y, cv=5,
scoring=["accuracy", "f1", "roc_auc"])`

`cv-results["test-accuracy"].mean()`

`cv-results["test-f1"].mean()`

`cv-results["test-roc_auc"].mean()`

GBM için aranacak olan hiperparametreleri sorguluyor.

97) `gbm-params = {"learning-rate": [0.01, 0.1],
"max-depth": [3, 8],
"n_estimators": [500, 1000],
"subsample": [1, 0.5, 0.7]}`

98) En iyi parametreyi elde etmek için:

`gbm-best-grid = GridSearchCV(gbm-model, gbm-params,
cv=5, n-jobs=-1,
verbose=True).fit(X, y)`

En iyi parametrelere için

99) `gbm-best-grid.best-params -`

Final Modelin Oluşturulması:
`gbm-final = gbm-model.select_params(xgbm-best-grid,`
`random-state=17)`
`fit(X17)`

Kurulan modelin hatırlarını tekrar değerlendirmek için 97'deki kodda `gbm-model` → `gbm-final` olarak değiştirip çalıştırır ve karşılaştırırız.

XGBoost (Extreme Gradient Boosting)

XGBoost, GBM'in hız ve tahmin performansını arttırmak üzere optimize edilmiştir; ölçeklenebilir ve farklı platformlarda entegre edilebilir versiyonudur.

Model nesnesini oluşturulur:

(10) `xgboost-model = XGBClassifier(random-state=17)`

96'daki gibi hiperparametre optimizasyonu önesi hatırlar değerlerini öncelikle görelim. Ardından XGBoost için de hiperparametre sorgulamasını yapalım.

(12) `xgboost-params = { "learning-rate": [0.1, 0.01, 0.001], "max-depth": [5, 8, 12, 15, 20, 25], "n-estimators": [100, 500, 1000], "colsample-bytree": [0.5, 0.7, 1] }`

* Değerlerden alınacak olan yöntem sayıları ile ilgili parametre

Ardından her zaman ki gibi `GridSearchCV` yaparız.

Ardından `xgboost-best-grid`'den gelen en iyi parametreleri final modelimize entegre ederiz: (`xgboost-best-grid.best_params_`)

Ardından 97'deki gibi parametreleri karşılaştırırız.

LightGBM

Ağac yöntemlerinin temeli; çok dallanıp eziyet yapabiliriz ya da biraz daha yukarıda kalıp genellenebilirliğini koruyabiliriz'e dayanmaktadır.

LightGBM

Birçok kaggle ve benzeri yarışmalarda başarısız sonuçlar, en önemli yöntemlerden birisidir.

LightGBM, XGBoost'un eğitim süresi performansını arttırmaya yönelik geliştirilen bir diğer GBM türüdür. Bu temel sebepten dolayı daha hızlıdır.

Level-wise büyüme stratejisi yerine Leaf-wise büyüme stratejisi ile daha hızlıdır. bunu tercih eder.

XGBoost değişkenleri bölme işlemi sütunlar üzerinde olduğunda bu noktada Level-wise (seviyeye göre büyüme) izlerken, LightGBM bölme noktaları odaklanır, yapraklara odaklanır.

XGBoost geniş kapsamlı bir ilk arama yaparken, LightGBM derinleşmeye ilk arama yapmaktadır.

LightGBM Modelleme

1. Model nesnesini oluşturalım.

```
lgbm_model = LGBMClassifier(random_state=17)
```

```
lgbm_model.get_params()
```

Çoğu Parametreleri gözden geçiriyoruz ve hiperparametre optimizasyonuna geçmeden önce hatalarımıza

(96)'daki gibi bakalım.

Ardından aramak için parametre verelim.

```
lgbm_params = {"learning-rate": [0.01, 0.1],  
               "n-estimators": [100, 300, 500, 1000],  
               "colsample-bytree": [0.5, 0.7, 1]}
```

Daha fazla parametre sürdürülebiliriz. Montajını işi anlatalım da!

TEORİK OLARAK

TECİRLER

LightGBM, XGBoost'ten çok daha hızlıdır. Random Forest'te, doğasında ağırlı öğrenme, online öğrenme eğitimi var, anlaması ve uygulaması basit, değişken ve gözetim seçiminde de iyi yaklaşım etkileyici.

LightGBM, hızlıdır.

Pratik olarak.

Ardından (104)'te atadığımız lgbm-parametrelerimizi (98)'deki gibi GridSearcher gibi sorgulattırız. Ardından (100)'deki gibi final modelimizi kurarız.

LightGBM'de temel olarak diğer hiperparametreleri belirledikten sonra buradaki tahminci sayısını (n-estimators) sayısını 10.000'lere kadar denemeliyiz.

Ve Dikkat! Bu anda geliştirmiş modelleri kullanıyoruz ve elimizde küçük bir veri seti olabilir. Bazı yöntemler karakteristiğini bu küçük veri setlerinde ortaya çıkaramaz. Dolayısıyla aslında daha karmaşık, daha yoğun veri setlerinde bu yöntemlerin daha faydalı olabileceğini görebiliriz.

LightGBM'in en önemli parametresi tahmin sayısı (n-estimators) dur.

Tahmin sayısı = iterasyon sayısı'dir.
Yani boosting sayısıdır.

Diğer parametrelere makul bir değer bulduktan sonra n-estimators'a bakalım. Ya da LightGBM'in fit metodunu kullanarak daha farklı bir açıdan tek başına LightGBM'e gidilerek n-estimator sayısını denemeliyiz. Zaman yeni model oluşturalım. Öncelikle hiperparametreleri sabitleyelim.

```
lgbm-model = LGBMClassifier (random-state=17,  
                             colsample-bytree=0.9,  
                             learning-rate=0.01)
```

Sabitledikten sonra "n-estimators"ları daha sağlıklı bir şekilde değerlendirmeliyiz. Sadece n-estimators'ı inceleyelim.

```
lgbm-params = {"n-estimators": [200, 300, 400, 1000, 2000, 10000]}
```

Yeni parametre setini girildikten sonra bunları aramasını istiyoruz.

```
lgbm-best-grid = GridSearchCV (lgbm-model,  
                                lgbm-params,  
                                cv=5, n-jobs=-1,  
                                verbose=True).fit(X,y)
```

Buradan sonra tekrar final modeli yapar ve (105)'teki gibi yeni parametreleri güncelledikten sonra modeli tekrar değerlendiririz.

```
cv-results = cross-validate (lgbm-final, X, y, cv=5,  
                             scoring=["accuracy", "f1", "roc-auc"])  
cv-results["test-accuracy"].mean()  
cv-results["test-f1"].mean()  
cv-results["test-auc-roc"].mean()
```

107

CatBoost

Eğer gözlem sonuçları aynıysa, değeri -
mesmi su şekilde yaparız. Bunun sebebi

1) Veriseti ile ilgili olabilir. A9 gözlem varsa
zaten a7 gözlem varsa çok fazla derine

gidemez çünkü gözlem sayısı biten
2) Diğer yandan çok fazla gözlem

olmadığı ram optimize edemeyebilir, forkla
alayım diyemez.

Ö zaman su eğilimde olmalıyız!
LightGBM kullanıyorsa 5000, 10000,

15000 gibi değerleri mutlaka denemeliyiz.
Bu değerlerde bir şey çıkmıyorsa denem-

diğine kadarat getirir yolumuza devam
ederiz.

CatBoost (Kategorik Boosting)

Kategorik değişkenler ile otomatik olarak
mücadele edebilen, hızlı, başarılı bir diğer
GBM türüdür.

Kategorik değişkenler aslında bir miktar
baş belası.

CatBoost Modelleme

catboost - model = CatBoostClassifier (random-state=17
CatBoost lu file yap. gözlem için ~~verbose = False~~

Parametre sergisini yapalım.

catboost - params = {"iterations": [200, 500],
"learning-rate": [0.01, 0.1],
"depth": [3, 6]}

Bu nedir?

muhtemelen ağaç sayısıdır,
Boosting sayısıdır,
n-estimators, denebiliriz.

Ardından GridSearchCV ve catboost_model.set_params yaparız.

Feature Importance

Bu aşamada başlangıçta plot-importance fonksiyonundan verileri önem sırasına göre sıraladık.

Hiperparametre Optimizasyonu için RandomSearchCV (BONUS)

Daha önce GridSearchCV öğrenmiştik. GridSearchCV demekti. Verilen bir setin olası tüm kombinasyonunu tet tet deniyordu.

RandomSearchCV ise verilecek bir hiperparametre seti içerisinde rastgele seçimler yapar ve bu rastgele seçimleri arar. Bütün olası senaryoları denediğinden dolayı, daha uzun sürer.

```
rf_model = RandomForestClassifier(random_state=17)
rf_random_params = {
    "max_depth": np.random.randint(5, 50, 10),
    "max_features": ["auto", "sqrt"],
    "min_samples_split": np.random.randint(2, 50, 20),
    "n_estimators": [int(x) for x in np.linspace(start=200, stop=1500, num=10)]
}
```

np.random.randint(2, 50, 20) : 2'den 50'ye kadar

20 tane sayı oluşturur.

[int(x) for x in np.linspace(start=200, stop=1500, num=10)] :

200'den 1500'ye kadar 10 tane sayı oluşturur.

GridSearchCV bütün olası parametre kombinasyonları denediğinden dolayı daha uzun sürer ancak olası en iyiye yaklaşma ihtimali daha yüksektir.

Random Search CV yöntemi ise Grid Search CV'ye göre daha azla, daha geniş bir hiperparametre aralığını arasından rastgele seçim yapar. Bu seçimleri içerisinde tek tek deneme yapar.

(11) rf-random = RandomizedSearchCV (estimator=rf-model
param-distributions = rf-random-param
n_iter=100, # denenecek parametre sayısı
cv=3,
verbose=True,
random-state=42,
n_jobs=-1)
rf-random.fit(X,y)

(112) rf-random.best_params_

En iyi parametreyi barmetmi sordugumuz
Ardından final-model kurup Cross-validation
ile degerleri karşılastırıyoruz

- Karar kurallarının hassaslaştırılarak kullanıldığı ağaç modellerinin genel adı GRADIENT BOOSTING'dir.
- Bootstrap yöntemini en iyi RASTGELE ÖRNEK SEÇİM yöntemi olarak tanımlar.
- Kategorik değişkenlerle otomatik olarak mücadele eden CATEGORY BOOSTING hali bir GBM türüdür.
- XGBoost, Gradient Boosting algoritmasını geliştirilmiştir. İle optimize edilmiş yüksek performanslı haldedir.

Denetimsiz Öğrenme (Unsupervised Learning)

Karşıma denetimsiz öğrenme olabılır. Öncelikle denetimli öğrenmenin ne olduğunu hatırlayalım. Elimizde bir kaç feature'lere karşılık bu featurelarda birşeyler olduğunda ortaya çıktığı varsayalım bir çıktı, yanıt, bağımlı değişken olduğunda buna denetimli öğrenme problemleri deriz.

Eğer ilgilenilen problemde bir bağımlı değişken, hedef değişken yoksa yani ilgili gözlem birimlerinde meydana gelen gözlemlere karşılık ortaya ne çıktığı bilgisi yoksa, diğer bir deyişle bir label yoksa, etiket yoksa bu durumda bunlarda denetimsiz öğrenme problemleridir.

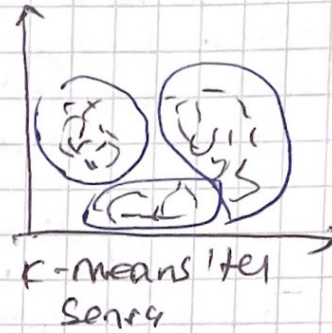
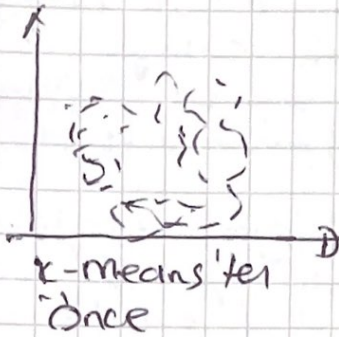
■ K-Ortalamlar (k-Means)

■ Hiyerarşik Kümeleme Analizi
(Hierarchical Cluster Analysis)

■ Temel Bileşen Analizi
(Principal Component Analysis)

K-Ortalamlar (k-means)

Amaç, gözlemleri birbirlerine olan benzerliklerine göre kümelere ayırmaktır.
Temel amaç, gözlem birimlerini kümelere ayırmaktır.



Yani: nasıl yapılır?

Adım 1: Küme sayısı belirlenir.

Adım 2: Rastgele k adet merkez seçilir. k -küme sayısı için k merkez seçilir.

Adım 3: Her bir gözlem için k -merkezlere uzaklıklar hesaplanır.

Adım 4: Her gözlem en yakın olduğu merkeze yani kümeye atanır.

Adım 5: Atama işleminden sonra oluşan kümeler için tekrar merkez hesaplamaları yapılır.

Adım 6: Bu işlem belirlenen bir iterasyon adedince tekrar edilir ve küme için hata kareler toplamının toplamının (total within-cluster variation) minimum olduğu durumdaki gözlemlerin kümelendirme yapısı nihai kümelendirme olarak seçilir.

Elimizde şimdi ise bazı kümeler var. Bunların merkezleri var. Merkez etrafında gözlem birimleri var. Şimdi öyle bir şey yapmalıyız ki bu kümeler kendi içinde homojen, birbirlerine göre heterojen olsun.

Bunu matematiksel bir birimde SSE (SSE) ile karsımla gelebilir.

Gözlem biriminin merkezindeki gözlem biriminin değerleriyle etrafındaki diğer gözlem birimlerinin değerlerinin farklarının karelerini alıp topladığımızdır. Bu kümenin kendi içindeki her bir karesini buluruz.

⇒ $SSE_1 + SSE_2 + SSE_3 = SSE$ hesaba sonucu SSE en düşük olursa o merkez seçilir.

Amaçımız kümeler ~~bir~~ bir homojenliği çok yüksek yapmak, birbirine yakın gözlemler olması demektir.

Kullanılacak Kütüphaneler.

```
#pip install yellowbrick
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import MinMaxScaler
from yellowbrick.cluster import KElbowVisualizer
from scipy.cluster.hierarchy import linkage
from scipy.cluster.hierarchy import dendrogram
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
```

Elimizde USArrests.csv verisi var. Amaçımız bu verideki eyaletleri sınıflara ayırmak, gruplara ayırmak.

```
df = pd.read_csv("dataset/USArrests.csv", index_col=0)
```

Bu indexle index bilgisi olduğundan dolayı bunları yapabiliriz.

Hızlı bir şekilde gösterelim.

df.head() → İlk 5 satır.
df.isnull().sum() → Eksik değer var mı?
df.info() → Kaç değişken var? Tipleri ne?
df.describe().T → Sayısal değerlerin özet

Şimdi uzaklık temelli bir yöntem kullanacağız. Uzaklık temelli ve Gradient Descent temelli yöntemlerin kullanımındaki süreçlerde değişkenlerin standartlaştırılması önem arz edilir. Yani buradaki değişkenleri de standartlaştırmamıza gerekmektedir.

① `sc = MinMaxScaler((0,1))` → Bu araç standartlaştırır.
`df = sc.fit_transform(df)`
`df[0:5]`
↓
standartlaştırma işlemi
→ Neden "head()" kullanarak incelemiyoruz?
→ Çünkü fit_transform'den çıktıktan sonra bunlar numpy array'leri döndürür. Bu yüzden böyle belirttik.

Şimdi modeli kuralım. Ancak gözetimsiz öğrenme gibi bir bağımlı değişken yok.

② `kmeans = KMeans(n_clusters=4).fit(df)` → random_state=17
③ `kmeans.get_params()` → Bağımsız değişkenlere sadece

↳ Parametrelerine bakabiliriz.

④ `n_clusters` → Disordan belirlenmesi gerekir. Örneğin parametre

$$SSE = SSD = SSR$$

4) Kmeans nesnesi verisinden neler alabiliriz.

5) `kmeans.n_clusters` → küme sayısını verir

6) `kmeans.cluster_centers_` → Cluster'ların merkezini verir.

7) `kmeans.labels_` → kümeler belirledi ve etiketler verir.

En yakın sınıflara olan uzaklıklarını, SSD hesaplamak için

8) `kmeans.inertia_` → SSD, SSE, SSR değeri

Optimum Küme Sayısı Belirleme

Öyle bir işlem yapmalıyız ki farklı k-değerlerine göre bir şeyleri inceleyip o şeylere göre karar vermeliyiz. Bunun için SSE'yi incelemeliyiz

```

8) kmeans = KMeans()
   ssd = []
   k = range(1, 30)
   for k in k:
       kmeans = KMeans(n_clusters=k).fit(df)
       ssd.append(kmeans.inertia_)

```

Burada "ssd" değerlerini inceleriz. Farklı değerler görürüz. Bu değerleri incelemek için grafik düzteriz.

```

9) plt.plot(k, ssd, "bx-")
   plt.xlabel("Farklı k değerlerine karşılık Uzaklık Artık Etkileri")
   plt.title("Optimum Küme Sayısı için Elbow Yöntemi")
   plt.show()

```


Göstern birimü kadar sınıf olduğıunda tüm göstern birimleri sınıf olur, her bir göstern birimü bir merkez olur. Dolayısıyla ssd sıfır olur. Bunun dolay küme sayısı arttıktan, hatanın düşmesini bekleriz.

K-means yöntemi, hiperansit kümelere yöntemi gibi kümelere yöntemleri kullanılarak algoritmanın bize verdiği matematiksel referanslara göre (SSE'ye göre) olan küme sayılarına direkt bakılarak ile yapılır.

Çalışma başında ile bilgisi ile birim zaten aşağı yukarı ne kadar küme olur bileceği ile ilgili bir ön filtrasyon olması gerekir.

Gözetimsiz öğrenme için sadece algoritmanın güvenmemeli, ile bilgisini de göz önünde bulundurmalıyız.

Körar vermek için eğilim en ziddetli olduğu noktalar seçilir. Grafikleri görmek zor olabilir fakat alternatif yolu var.

```
(10) kmeans = kmeans()  
elbow = ElbowVisualizer(kmeans, k=(2,20))  
elbow.fit(dt)  
elbow.show()
```

↳ Bu veri setini kümelere ayırmak istiyorsanız ayırmak gereken optimum nokta burasıdır demniç. Bunu görmek için:

```
(11) elbow.elbow_value -
```

↳ Optimum küme sayısını verir. Hiperparametre optimizasyonu yapmış olur.

Final Cluster'ların Oluşturulması

Önce parametreyi güncelleyelim.

(12) `kmeans = kmeans(n_clusters = elbow.elbow_value).fit(df)`

(13) `kmeans.n_clusters`
`kmeans.cluster_centers_` → Değerleri Görmek İçin
`kmeans.labels_`

Fakat hangi etiket hangi cluster'da bilmiyoruz. Bunu çok basit şekilde düzeltebiliriz. Önce labelleri bir nesneye atayalım.

(14) `clusters = kmeans.labels_`

Ardından veri setini tekrar tanımlayalım.

(15) `df = pd.read_csv("datasets/USArrests.csv", index_col=0)`

Ardından yeni bir sütun ekleyelim ve hangi satırın hangi sütuna ait olduğunu bakalım

(16) `df["cluster"] = clusters`
`df.head()`

Böylece bazı değerler belli kümelede, aynı kümelede toplanmış olur. Sıfırıncı küme olarak görebiliriz. Bunu düzeltebiliriz.

(17) `df["cluster"] = df["cluster"] + 1`

Sorguyu da şu şekilde yaparız.

(18) `df[df["cluster"] == 5]` → 5. sınıf / cluster'in elemanlarını / değerlerini gösterir.

Değişkenleri standartlaştırdığımız için bu sınıflandırmaya güvenebiliriz.

(19) Ardından her sınıftaki bilgilerin ortasını görebiliriz.

`df.groupby("cluster").agg(["count", "mean", "median"])`

Bu bize kaç sınıf var, ortalaması ve median değeri ne diye söyler.

Birbirine yakın değerde veriler farklı kümelere bulunuyor olabilir. Sadece algoritmanın verdiğiye bakmıyoruz. Bunun yanında kendimizi de kontrol ediyoruz. Dosyayı kaydedebiliyoruz.

(20) `ldf.to.csv("clusters.csv")`

Hiyerarşik Kümeleme Yöntemi (Hierarchical Cluster Analysis)

Gözetimleri birbirine olan benzerliklerine göre alt kümelere ayırmaktadır.

Burada kümelere ayırma işlemi hiyerarşik bir şekilde gerçekleşir. İki yalı vardır.

1) # Birleştirici Yöntem (Agglomerative)

Örneğin, başında tüm gözlem birimleri tek başına bir küme gibi düşünür. Zamanla doğru iki gözlem birimi bir araya gelip bir küme oluşturur. Daha sonra bunlar da birleşip yeni bir küme oluşturur.

2) # Bölünleştirici Kümeleme Yöntemi (Divisive)

Tüm gözlemlerden iki küme, o iki kümeden dört küme gibi devam eder. Bölünerek azacı doğru tüm veriseti her bir gözlem tek başına kalacak şekilde azacı doğru dağıtılır.

K-Means'tan Farkı

K-means'ta küme oluşturma sürecine dışarıdan müdahale edemiyorsunuz. Başlangıçla bir gözlemlene imkanı, sayısı yoktur.

Hiyerarşik kümede belirli noktalardan girilerek görsel teknik çerinde gezilebilir. Kümeleme seviyelerinde yeni kümelere tanımlayabilirsiniz.

Kodlama kısmını geçtikçimizde veri setini standartlaştırmamız gerekmektedir.

```
21) sc = MinMaxScaler(1,1)
df = sc.fit_transform(df)
```

↳ Her seferinde standartlaştırmamızın sebebi UZAKLIK TEMELLİ YÖNTEMLER KULLANDIĞIMIZ İÇİNDİR.

Buradan sonra bütünleyici ya da birleştirici yöntemlerden bir tanesini kullanabiliriz.

```
22) hc = average = linkage(df, "average")
```

↳ Birleştirici bir clustering yöntemi
↳ Öklid uzaklığına göre gösterilen kümeler (Hepsini bireysel sayıp benzerliklerine göre kümeliyor)

Hiyerarşik kümelene yöntemlerinin temel yapı taşı "dendrogram" adını verdiğimiz kümelene yapısını gösteren şemadır.

```
23) plt.figure(figsize=(10,5))
plt.title("Hiyerarşik Kümelene Dendrogram")
plt.xlabel("Gösterim Birimleri")
plt.ylabel("Uzaklıklar")
dendrogram(hc, leaf_font_size=10)
plt.show()
```

↳ Hiyerarşik kümelene dendrogram/ağacı haritasını bize verir.

Küme sayısını manuel olarak ayarlamak isterseniz dendrogram genişliğini değiştiririz.

(24) dendrogram (hc-average,
truncate-mode="lastp",
p=12,
show_contracted=True,
leaf-font-size=12)

Hiyerarşik kümellemenin amacı, avantajı
birte götlen birimlerin gelenden bakma zansı
tanınmasıdır. Böylece belirli karar noktalarına
dokunabilme zansı tanınmasıdır.

Küme Sayısı Belirleme

Dendrogramın theme programatik olarak
çizme yapabiliriz.

(25) plt.figure(figsize=(7,5))
plt.title("Dendograms")
dend = dendrogram(hc-average)
plt.axhline(y=0.6, color="r", linestyle="--")
plt.show()

• $y=0.6$ 'dan kesikli çizik çıkar. farklı sayıda
farklı renklerde çizilebiliriz.

Final Modeli Oluşturma

(26) from sklearn.cluster import AgglomerativeClustering
cluster = AgglomerativeClustering(n_clusters=5)

"Ön tanımlı uzaklık
metriği"

"linkage='average'"

Cluster yapısını
ortaya çıkaracak olan
ilgili metodu kullandık.

Seneca(.fit(df)) yazabiliriz fakat farklı olsun diye aşağıdaki gibi yapalım.

(27) ~~cluster~~ clusters = cluster.fit_predict(df) → ^{kümeleme işlemi yapıldı.}

~~dataframe'leri~~
Ardından dataframe'ye ekleyeceğiz ve cluster'ları buraya ekliyoruz olacaktır

(28) `df = pd.read_csv("datasets/USArrests.csv", index_col=0)
df["hi-cluster-no"] = clusters`

Burada sıfırdan başlayınca holumuz gitmiyordu. Bunu da

(29) `df["hi-cluster-no"] = df["hi-cluster-no"] + 1`
↳ Hiyerarşik kümeleme böylece düzenleriz.

Ardından "kmeans-cluster-no" ile "hi-cluster-no" kıyası yapabiliriz. Yan yana sütunlara koyabiliriz.

Temel Bileşen Analizi (Principal Component Analysis) (PCA)

- Temel fikri, çok değişkenli verinin ana özelliklerini daha az sayıda değişken / bileşen ile temsil etmektir.
- Diğer Madesi ile; küçük miktarda bir bilgi kaybını göze alıp değişken boyutunu azaltmaktır.

Eğer mülakatla Principal Component Analizi ne diye soru gelirse!

→ Temel bileşen analizi bir boyut indirgeme yaklaşımıdır. Verisetinin boyutunu küçük miktarda bilgi kaybını göze alarak indirgeme işlemidir.

PCA (Principal Component Analysis), bir boyut indirgene yaklaşımdır. Bir amaç. Bir çok farklı amaç için kullanılabılır.

PCA, veri setinin bağımsız değişkenlerin doğrusal kombinasyonları ile ifade edilen bileşenlere indirger. Bu bileşenler arasında korelasyon yoktur.

Varyans ve özdeğerlere dayalı gruplandırma yapılır.

Hitters verisetindeki asıl amaç maç tahmini gerçekleştirmektir.

df = pd.read_csv("datasets/Hitters.csv")
num_cols = [col for col in df.columns if df[col].dtype != "O" and "Salary" not in col]
↳ Verisetinde gez. Tipi object olmayan ve Salary değilse ilgili sütun bunları getir des.

df[num_cols].head()

↳ (32)'den sonra tüm sayısal olmayanları atan Sayısal verileri ekrana yazarır. Ardından bir nesneye atarız ve etset verileri uyarırız.

df = df[num_cols]
df.dropna(inplace=True)
df.shape

Burada değişken sayısını düşürmeye çalışacağız. Bu yöntem standartlaştırma ihtiyacı duyan bir yöntemdir.

df = StandardScaler().fit_transform(df)

Ardından nesnemizi kuralım.

(34) $\begin{cases} \text{pca} = \text{PCA}() \\ \text{pca.fit} = \text{pca.fit_transform}(df) \end{cases}$

PCA temel amacı verisetindeki bilgimin daha az sayıda bileşen ile açıklanmasıdır. Bunu yaparken belirli miktarda bilgi kaybı göze alınır.

BİLGİ = VARYANS'tır. Dolayısıyla bu bileşenlerin başarısını değerlendirebileceğimiz metriği açıkladıkları varyans oranıdır.

Bileşenlerin açıkladıkları varyans oranı R²m

(35) $\text{pca.explained_variance_ratio}$

↳ Açıkladıkları bilgi oranı demektir.

Pespeşe ikinci bileşenin açıklıyarak olduğu varyans R²m.

(36) $\text{np.cumsum}(\text{pca.explained_variance_ratio})$

↳ Verilen çıktıdaki ^{değer} birinci bileşenin tek başına açıkladığı varyans oranı,

→ 2. değer → 1. bileşen + 2. bileşenin toplam açıkladıkları varyans

→ 3. değer → 1. bileşen + 2. bileşen + 3. bileşen toplamı.

Bir noktadan sonra 0.99... olacaktır.

📌 cumsum (kümülatif toplam) kullanıyorum ki bu bileşenler bir araya geldiğinde toplam ne kadar açıklama oranına sahip olur göreyim.

Bu kümülatif toplam sonucu elde edilen sayısal değişkenlerden yola çıkarak veriyi nasıl küçületeceğiz. Bu bileşen sayısına nasıl karar vereceğimize bakalım.

Optimum Bileşen Analizi

Elbow yöntemi ile en keskin geçişin en doğru döğü değişiminde nerede olduğunu verir. (incelenir.) Grafik için

3) `pca = PCA().fit(df)`
`plt.plot(np.cumsum(pca.explained_variance_ratio_))`
`plt.xlabel("Bileşen Sayısı")`
`plt.ylabel("Kümülatif Varyans Oranı")`
`plt.show()`

↳ Grafikten kaç bileşen ile veriyi ne kadar açıklayabileceğimize yüzdeyi görüyoruz. Örneğin grafikten 3 noktayı seçelim. Yani ilk üç sütun veriyi %80 bandında açıklıyor. Karar kabul edelim. Böylece final modeli oluşturmaya geçebiliriz.

Final PCA'ın Oluşturulması

3) `pca = PCA(n_components=3)`
`pca_fit = pca.fit_transform(df)`

(Bileşen Sayısı = 3)

Açık olan varyans oranlarını tekrar gözlemleyelim. Hangi bilgi TEK BAŞINA veriyi ne kadar açıklıyor.

VERİ SETİNDEN ELDE EDİLEN BİLEŞENLERİN VERİ SETİNİN VARYANSINI AÇIKLAMA ORANI

3) `pca.explained_variance_ratio_`

↳ İlk üç bileşeni seçmiştik. İlk üç bileşen tek başına verinin yüzde kaçını açıkladığını verir.

(42)

np.cumsum(pca.explained-variance-ratio-)

↳ Bu da 4 bileşen toplam ne kadar varyans açıkladığını, ne kadar bilgi açıkladığını verir.

❏ Bölüm başındaki "bir miktar bilgi kaybı"

❏ kısmı burasıdır.

MÜLAKAT SORUSU

Elimde bir veri seti var ama verisetinde label yok. Ama sınıflandırma problemi çözmem gerekiyor. Ne yapabilirim?

1) Önce Unsupervised şekilde çeşitli cluster'lar çıkarırım.

2) Bu çıkardığım cluster'lar = sınıflar olur. Yani 4 cluster varsa 4 sınıf var diye düşünürüm.

3) Onları etiketleriz. ve veri setine ekleriz.

4) Ardından onları bir sınıflandırıcıya sokarız.

5) Yeni bir müşteri geldiğinde ve müşteri özellikleri olduğunda onu ait olduğu sınıfa ekleyebiliriz.

YANI?

1) Önce UNSUPERVISED yöntem kullanırım.

2) Önce çıkan CLUSTERLERE - LABEL muamelesi yaparız.

3) Daha sonra bunu sınıflandırıcıya sokup yeni bir gözlem birimi geldiğinde bunu artık sınıflandırılabilir hale getiririm.

K-Means algoritması ile sınıflandırma yaptığımız bir veri setinde optimum küme sayısını bulmak için inertia- kullanılır.

Makine Öğrenmesi Pipeline

Bu bölümün sonunda gelmeyi hedeflediğimiz yer belirli işlemleri otomatikleştirip bir şekilde yapabilmeli ve script level yani işletim sistemi düzeyinde çeşitli python makine öğrenmesi görevlerimizi kullanabilmeyi öğrenmiş olacağız.

1) # Kesitici Veri Analizi (EDA)

Burada bazı fonksiyonlar tanımlayıp kullanacağız.

`def check - df` : shape, dtypes, head, ~~isnull()~~ sum(), tail, quantile'ye yazar.

`def cat - summary` : Kategorik değişkeni ve bir dataframe verdiğimizde o kategorik değişkenin sınıf frekansını ve sınıfların oranını ifade eder. Kategorik değişkeni gösterir.

`def num - summary` : Sayısal değişkeni göstermek için kullanacağımız fonksiyon.

`def target - summary - with - num` : Bağımlı değişkeni sayısal bir değişken ile göstermek için kullanılır. Bağımlı değişkene göre groupby alınır ve sayısal değişkene göre ortalaması alınır.

`def target - summary - with - cat` : Bağımlı değişkeni bir kategorik değişken ile göstermek istediğimizde kullanılır. Kategorik değişkene göre groupby'la alıp bağımlı değişkenin ortalamasını alır.

`def correlation - matrix` : Verisetinin içerisindeki sayısal değişkenlerin birbirleri arasındaki korelasyonu hesaplayarak ve bunu görselleştirir. Bir harita yardımıyla

def grab_col_names: Kategorik değişkenleri, sayısal değişkenleri ve kategorik gibi görünen kardinal değişkenleri verir. Kategorik değişken içinde sayısal gibi görünen ama örneğin a kadar sınıftan az sayıda sınıfa sahip olduğundan dolayı bizim kategorik değişken muamelesi' yaptığımız değişkenler vardı. Sınıf sayısı 10'dan küçük ise bunlar sayısal değil kategoriktir diye düşünüp kategorik listeye attığımız değişkenler vardı.

İkinci şu şekildedir. $\Rightarrow$ def grab_col_names(dataframe, cat_th=10, car_th=20):

Sınıf sayısı
ondan küçük
ise bunlar sayısal
değil kategoriktir.

Kategorik
gibi görünen
ama kardinal
olan değişkenler

41) Öncelikle datayı tanımlayalım.

df = pd.read_csv(---)
check_df(df) $\rightarrow$ özetini verir.

42) Ardından değişken türlerini ayırtıralım.

cat_cols, num_cols, cat_but_car = grab_col_names(df, cat_th=5, car_th=20)

cat_th=5: Yani bir sayısal değişkende 5'ten az sınıf varsa bunu kategorik olarak görer demektir. Kategorik değişkenlerin incelenmesi

43) for col in cat_cols:
cat_summary(df, col)

Sayısal değişkenlerin incelenmesi

44) df[num_cols].describe().T

Sayısal değişkenlerin grafiği

45) for col in num_cols:
num_summary(df, col, plot=True)

46) Sayısal değişkenin birbirini ile korelasyonu

correlation_matrix(df, num_cols)

1: pozitif yönlü korelasyon
-1: negatif yönlü korelasyon

Target ile sayısal değişkenlerin incelenmesi

(47) `for col in num_cols:`
`target_summary_with_num(df, "Outcome", col)`

(2) # Veri Ön İşleme (Data Pre-Processing)

Bu bölümde bu verisetindeki olası problemleri
çözüp yeni değişkenler türetilip bu işlemleri bir
fonksiyona tanımlamaya çalışacağız.

`def outlier_thresholds` : Kendisine girilen bir değişkenin
out ve ist eşik değerlerini
hesaplamaktır. Bunu `boxplot` ile yapar.

`def replace_with_thresholds` : Aykırı değer var mı
yok mu? Bunu sorgular ve gönderir.
Bunun yerine eşik değeri koyulabilir.

`def check_outlier`

`def one-hot-encoder` : Yeni değişkenler türeteceğiz
ve türetilen bu yeni değişkenlerde
kategorik değişkenler var. Bundan dolayı
`one-hot-encoder` kullanılır. Aynı
zamanda `label-encoder` olarakta kullanmak
istersek `drop_first=True` yaparız.

Yapılacak olan işlemlerde değişkenler hem büyük
hem de küçük harflerden oluşur. Bu bize sıkıntı
yaratabilir.

Değişken isimlerini büyütme

(48) `df.columns = [col.upper() for col in df.columns]`

Yeni değişkenler de oluşturulabilir. Mesela
Günlük rain alınacak olursa :

(49)

0 dan başlatması için çok önemli

```
df["NEW-GLUCOSE-CAT"] = pd.cut(x=df["GLUCOSE"], bins=[-1, 139, 200], labels=["normal", "prediabetes"])
```

0-139 arası normal

139-200 arası ön diyabet diye yeni değişken atadık. Bunu var olan bir değişken içerisinde ürettik.

Glikoz'dan yeni bir değişken ürettik. Peki şimdi Glikoz sütununu sileriz mi? Hayır!

Bunun için modelleme aşamasını bekleyelim.

Var değişkeni içerisinde de sınıf oluşturabiliriz.

(50)

```
df.loc[(df["AGE"] < 35), "NEW-AGE-CAT"] = "young"
df.loc[(df["AGE"] >= 35) & (df["AGE"] <= 55), "NEW-AGE-CAT"] = "middleage"
df.loc[(df["AGE"] > 55), "NEW-AGE-CAT"] = "old"
```

NEW-BMI-RANGE adlı da oluşturabiliriz.

NEW-BLOODPRESSURE ~ ~ ~

Burada yapılan ayrıştırmaların hepsi var olan değişkenlerin içerisinde çok büyük umutlarla oluşturulan değişkenler değildir. Sadece burada machine öğrenme sürecinin iki taraflı olduğunu göstermek için bu değişkenleri ekledik. Yani veri ön işleme kapsamında ve keşifçi veri analizi kapsamında yaptığımız işler research adı altında birbirini iki taraflı etkilemektedir. Bunca şeyin ardından tekrar kontrol etmeliyiz.

(51) check - df(df)

Önemli **ÖZELİ** Bir kod akışı esnasında belirli bir veri geçince her şey ydunda mı? diye sormaktır. Her şey ydunda gibi görüyorsa grab-col-names'i bir daha çağıracaktır. Çünkü artık yeni değişkenler vardır.

cat-cols, num-cols, cat-but-cor = grab-col-names(
 df, cat-th=5, cor-th=20)

Yeni değişkenler var. Değişken mühendisliği bölümünde yeni feature'lar eklemenin en büyük sebebi ileri tarafta sürekli birbirini etkileyen ve etkilenenlerin farkında olmamızı sağlamayı amaçlamaktır. Ardından devam ederiz.

for col in cat-cols
 cat-summary(df, col)

Bu sefer cat-cols'larda daha fazla değişken olduğundan dolayı buradaki kategorik değişkenleri gösteriyor olacak. Yeni oluşturulan değişkenler ile bağımlı değişkenlerin ilişkisi ne şekildedir? Aşağıdaki kod ile göreceğiz.

for col in cat-cols
 target-summary-with-cat(df, "outcome", col)

TARGET-MEAN, TARGET = outcome'ydı. Bunun 1. sınıfı diyabeti 0. sınıfı diyabet olmamayı ifade ediyor. Sağlıklı sınıf 1'den uzaktır. Obez sınıf 1'e yakındır.

Outcome bizim bağımlı değişkenimiz olduğundan artık onu kategorik değişken listesinden çıkarabiliriz.

cat-cols = [col for col in cat-cols if "outcome" not in col]

Elimizde sadece kategorik değişkenler kaldı.

Makine Öğrenmesi modelleri binlerce bit standart bekle. Tüm değişkenleri ya

- Binary Encode etmeliyiz ya da
- Sayısal Bir Formda ifade etmemiz gerekir

Bu noktada yeni kategorik değişkenlerle ilgili yapabileceğimiz bazı şeyler var. Bu noktada yine yorumu açık noktadır. Mesela düştürden kategorik değişkenler 1-2-3-4 gibi değerlerle temsil edilip olduğu gibi bırakılabilir. Hangi sınıfın büyük hangi sınıfın küçük olduğunu bilerek bu dönüşümü kullanabiliriz.

Bir diğer tercih bunları one-hot-encoder'den geçirmek olabilir. Bu yolda bir değişkendeki sınıf sayısı kadar yeni kategorik değişken oluşuyor olacak. Böylece bunlar Binary Encode edilmiş olacak.

54) `df = one-hot-encoder(df, cat_cols, drop_first=True)`
`check-df(df)`

Önce one-hot-encoder'e sahip sonra veriyi kontrol edersin.

Bunu True yapıyorum ki ilk sınıflı olan kategorik değişkenlerin ilk sınıfı dışarıda kalmış olmasıyla Dummy Değişken Tutacağından kurtulalım.

Değişken isimleri yeniden boyutlanmak istenebilir.

55) `df.columns = [col.upper() for col in df.columns]`
Yapılan analizleri tekrar tekrar güncelliyorum.

56) `cat_cols, num_cols, cat_but_col = grab_col_names(df, cat_th=5, car_th=20)`

En son güncel değişken türlerini tutuyorum.



MADEN MAKİNALARI
İMAKAT MÜHENDİSLİK
PAZ. SAN. ve TİC. LTD. ŞTİ.

True : Aykırı Değer Var.
False : Aykırı Değer Yok

Güçtürüm
enir.

Aykırı değerleri göstereceğiz.

(59) `for col in num_cols:
 print(col, check_outlier(df, col, 0.05, 0.95))`

Değiştirilebilir

Böyle yapmanın sebebi sadece Aykırı
Aykırı olmalarıdır.

Bunu yapmayada bilirdik diğer yöntemlerde.

Bulunan aykırı değerler threshold değeri ile
değiştirilelim.

Aykırı Değer
Bulundu

(60) `replace_with_thresholds(df, "INSULIN")`

Ardından standartlaştırma işlemi yapalım.
Mümkün olan en güzel eğitimi gerçekleştirile-
mek için standartlaştırma işlemi yapmayı
tercih ederiz.

(61) `X_scaled = StandardScaler().fit_transform(df[num_cols])`

Standartlaştırılan veri tekrar Dataframe'e geri
atılır. Burada numpy array'e dönüştü, isimlendirme gitti.

(62) `df[num_cols] = pd.DataFrame(X_scaled, columns=df[num_cols].columns)`

Böylece isimlendirilmesini sağlarız.
Ardından bağımlı ve bağımsız değişkenleri seçmeliyiz.

(63) `y = df["Outcome"]
X = df.drop(["Outcome"], axis=1)`

TEKRAR TEKRAR
Teker check_df etmeliyiz

(64) `check_df(X)` Son adımlarda ila ki
problemler çıkabilir.

Bütün veri önleme adımlarını

(65) `def diabetes_data_prep(dataframe):`

ile fonksiyonlaştırırız.

↳ Bunu çağırmak için:

(66) `X, y = diabetes_data_prep(df)` kullanırız.

(67) `check_df(X)` ile kontrol de ederiz.

Pyplaine nedir?

Pypline bir boru hattı demektir. Ulu uca boruları oklayarak belirli bir aktarma işlemi temsil eder. Bu anda borumuzun ilk halkasını döşemiş olduk.

[3] # Base Models

Genelde pypline'de yer alması gerekir. Normalde tek model kullanılır fakat biz 2 model arasında da gideceğiz. Olayı durumda nasıl ilerleneceği gösterilir. Bir fonksiyon oluştururuz.

(68) `def base_models(X, y, scoring = "roc_auc")`

Bu fonksiyonun altında classifiers var. Bir skor algoritması da var. Şimdi bu modeli değiştirince diğer tüm modellere fikirleri sormasını isteyeceğiz. Temel hatalarını görmek istiyoruz.

Böylece öyle bir işlem yapmalıyız ki bu liste (classifiers) içerisinde birinci eleman model ismi ikinci eleman sınıflandırıcıları (classifiers) temsil etsin. Bu işlem için döngüye ihtiyacımız var.

69

Her modelin cross-validation:

```
for name, classifier in classifiers:
    cv_results = cross_validate(classifier, X, y,
                                cv=3,
                                scoring=scoring)
    print("# {scoring} : {round(cv_results['test_score'],
                                mean(1,4))} ({name})")
```

Ardından;

70 base_models(X, y) yazmam yeterli olacaktır.

En başarılı modeli görebiliriz. Tek satırlık bu kod ile tüm modellere baktık.

14 # Otomatik Hiperparametre Optimizasyonu

Birde fazla görev birde fazla evane, görevin yine bir fonksiyon tanımlayarak yapacağız.

Sonuçta bir bir pipeline oluşturacağız. Bunun amacı bir makine öğrenmesi sürecini fonksiyonel bir formda belirli noktalarla müdahale edilebilir bir şekilde tanımlamaktır. Bu yüzden bunu hatırlamaktadırlar. Belirli yerler üzerinde oynatabiliriz. Genelde bir proje için oluşturulan pipeline o proje içinde dir, genellenemezliği değildir.

knn, cart, rf, xgboost, lightgbm için ayrı ayrı parametre seti veriyor. Arcana yapması ile diğer hiperparametre setleri verilir.

Şimdi şöyle bir döngü yazmalıyım ki isimleri alsın, raporlamak için, modelleri alsın, parametreleri alsın. Daha sonra gördüğümüz hiperparametre optimizasyonunu otomatik bir şekilde uygulansın.

(71) `def hyperparameter_optimization(X, y, cv=3, scoring='roc_auc')`
Bu fonksiyonu çağırtmak için

(72) `best_models = hyperparameter_optimization(X, y)`

15 # Stacking & Ensemble Learning

Temeli birden fazla modeli bir arada kullanmaya dayanmaktadır. Birden fazla ağa, bir araya gelip bir sonuç gibi hareket ediyorlar. Bunun için de fonksiyon yazacağız.

(73) `def voting_classifier(best_models, X, y)`

~~bu~~ Bu fonksiyon içerisinde VotingClassifier kullanılır.

VotingClassifier'da ki ; Bana kullanacak olduğun modelleri söyle.

Ardından bunların cross-validate hatasını bakarız. Bu fonksiyonu kullanmak için

(74) `voting_clf = voting_classifier(best_models, X, y)`

76) # New Observation

Voting - classifier kullanarak tahmin etme işlemini yapalım ve tekrar genel resimimizde dönelim. Veriseti içerisinde rastgele bir gözlem birimi seçelim. Bu gözlem birimi ram bu verisetini kullanarak bir tahmin işlemi gerçekleştirebiliriz.

77) `X.columns` → Değişkenlerimizi verir

Veri in işleme sonrası son halimizi

78) `random_user = X.sample(1, random_state=45)`

Fonksiyonu çağırarak ve predict diyerek bu kullanıcı ram diyabet tahminini gerçekleştirebiliriz.

79) `voting_clf.predict(random_user)`

↳ Diyabet tahmini gerçekleştirilir.

Eğer bu öğrenmenin kaybolmasını istemiyorsak kaydedebiliriz.

80) `joblib.dump(voting_clf, "voting_clf2.pkl")`

Voting - clf2.pkl adında dosyaya kaydedtiğimiz. Bunu tekrar yüklemek istersek

81) `new_model = joblib.load("voting_clf2.pkl")`

↳ Böyle yüklenir. Bunu da şu şekilde kullanırız

82) `new_model.predict(random_user)`

17) # Pipeline main Function

Buraya kadar olan aşamada nihai olarak projeyi son haline getireceğimizi yapıyı belirledik. Şimdi öyle bir işlem yapmalıyız ki örneğin bu python script'ini bir proje dosyası gibi bir sunucu ortamında ya da kişisel bir bilgisayarda işletim sistemi seviyesinde çalıştırabilmeliyiz.

81) Pipeline aşamaları tek bir fonksiyonda

```
def main():  
    df = pd.read_csv("datasets/diabetes.csv")  
    ↳ Veritabanından veriyi çek  
    X, y = diabetes_data_prep(df)  
    ↳ Çekilen veriyi veri ön işleme scriptinden geçir.  
    base_models(X, y)  
    ↳ Genel modellere bak  
    best_models = hyperparameter_optimization(X, y)  
    ↳ Hyperparameter optimizasyonu. En iyi modelleri al.  
    voting_clf = voting_classifier(best_models, X, y)  
    ↳ Bununla voting_classifier oluştur.  
    joblib.dump(voting_clf, "voting_clf.pkl")  
    ↳ Daha sonra bu modeli kaydet  
    return voting_clf
```

82) fonksiyonları bir arada tek bir dosyada "helpers.py" adında bir dosyada tuttuk şart edelim.

82) `from helpers import *` → Tüm fonksiyonları yükle

83) `from helpers import a, b, c` → İlgili fonksiyonları yükle

Orta ~~şekilde~~ ilgili girbette

→ utils.py } ilgili kütüphanelerin olduğu
→ helpers.py } adında dosyalar görebiliriz.

Bunlar içerisinde program ana scriptinde yer kaplamaması için yorak, başka scriptler tarafından da kullanılabilircek fonksiyonları da barındırıyorlar. Bu sayede dışarıdan çağırılabilir.

→ config.py : ilgili program dışarıdan ayarlamaya müsait olan bazı ayarların barındırıyor olabilir.

(81) kodun devamı:

(84) `if __name__ == '__main__':` → Dandir name
`main()` → Dandir main diye oturur.

→ Eğer çalıştırılabilir bir nesne varsa
→ zaman bu işi yapar.

→ TEMEL AMACI: Bir Python dosyasını tetikleyici bölümüdür. Bir Python dosyasını çalıştıracak olan nihai bölümdür.

→ Kodun içine girip çalıştırma görevini görmesi için böyle bir blok eklenir.

Bu noktada `main()` kısmı başlamadan önce çeşitli dokunuşlar yapmamızı sağlar.

(86) `if __name__ == '__main__':`
`main()`
→ Bu noktada dokunuşlar yapabiliriz. İlk çalışacak kısım burası.

→ print atanabilir.
→ çeşitli görevlerin süreleri hesaplanabilir.

Yani kısaca bir Python script'ini işletim sistemi seviyesinden, komut satırı seviyesinden çalıştırmak istediğimizde böyle bir blok ekleyerek bunu gerçekleştirebiliriz.

Prediction (BONUS) (SCORING)

Daha önce kaydettiğimiz dosyayı tekrar başka bir bilgisayarla çalışalım.

```
86 import joblib  
import pandas as pd
```

Problem: Hastanemizde yapay zeka ile diyabet tahmini yapıyorca model kurduk ve sakladık. Şimdi test başları.

Yeni kayıtlar geldi. Problem burada başlıyor. Bu kayıtları hastalarla ilgili bir tahmin sonucu üretilecek diyelim. Örnek olsun isim rastgele hasta çekelim

```
87 random-user = df.sample(1, random_state=45)  
Şimdi modeli çağıralım.
```

```
88 new-model = joblib.load("voting-clf.pkl")
```

Daha önce kurulan modele soralım.

```
89 new-model.predict(random-user)
```

! HAHA BURADA BÖZLİYORUZ !

Bir boyut hatası alabiliriz. Boyutların uyumsuz olduğunu söyleyebiliriz.

GÖRÜLEBİLİR BİR PARADOKSAL NOKTASI

Şimdi modeli kurarken modelle sürekli oynadık. Modeli değiştirdik. Dışarıya feature'ler artık farklı featurelerdir. Yenileri türedik. Yeni hasta geldi fakat bu hastanın bilgileri bizim yapımız ile uyumlu değildir. Bu verisetinin bizim verisetimiz ile aynı olması, dönüştürülmesi lazım.

Sonradan etkilenen verisetini ~~diğer~~

diabetes_data_prep'ten geçirmek lazım.

91) from diabetes_pipeline import diabetes_data_prep

'ten geçirmemiz lazım. Burada hata alıyor olursak aşağıdaki fonksiyonları yüklememiz gerekir.

```
def grab_col_names
def outlier_thresholds
def replace_with_thresholds
def one_hot_encode
def diabetes_data_prep
```

Bu yüklemeleri yaptıktan sonra hata almayıp önüne geçmiş olacağız.

92)'i güncelledik diyelim. Şimdi ise

93) X,y = diabetes_data_prep(df)

Yeni gelen veriyi Prediction işlemi için

aynı scriptten geçirmek zorundayım.

Artık bu işlemi yaptığımızda ve datadaki benzerlikleri yakaladığımızda göre işleme tekrardan geçirebiliriz.

(93)

```

random-user = X-sample(1, random-state=50)
new-model = joblib.load("voting-clf.pkl")
new-model.predict(random-user)

```

↳ Sağlıklı bir şekilde analize devam ederiz.

Anlaşılacağı üzere bu modeli tekrar kullanmak istediğimizde modelin anladığı ve bildiği bir şey var. O şeyi uygun bir şekilde değişken seti göndermemiz lazım. Bunun dolayı buradaki veri işleme işlemini tekrar ete aldık.

❑ Bir makine öğrenmesi pipeline'ı veri oluşturma işlemini içermez.

❑ Bir sınıflandırma problemi için base modelin kontrol noktası referans modelidir.

❑ Bir pipeline'da kullanılacak işlemler için dışarıdan bir script getirmek kod okunurluğunu sağlar.

❑ Makine öğrenme aşamalarından olan Model training and fitting "modelin yapısını öğrenmek" anlamına gelir.

$$\text{PRECISION} = \frac{TP}{TP + FP}$$

❑ Karar ağaçlarında en tepede görülen değişken HATA EN GÜÇ AKAHAN DEĞİSKENDİR.

❑ CART modelinde en iyi bölünme işlemini (best split) bulurken Gini KATSAYISI dikkate alınır.

■ Kmean algoritması kullanılarak sınıflandırılacak istenen veri setinin kümelerinin merkezi;

```
from sklearn.cluster import KMeans
k_fit = KMeans().fit(df)
k_fit.cluster_centers_
```

↳ Küme merkezi bu kod ile bulunur

■ Hedef değişken yer almadığı bir çalışmada TEMEL BİLEŞEN ANALİZİ (Unsupervised Learning) kullanılır

■ Doğrusal regresyonda bağımsız değişkenlerin (x) katsayılarını (w) tahmin etmek machine öğrenmesinde VERİNİN ÖRÜNTÜSÜNÜ BULMAYI ifade eder.

$$\hat{y} = \beta + w_1 \cdot x_1 + w_2 \cdot x_2 + \dots$$

■ Gradient Descent, fonksiyonun minimizeasyonu için kullanılan bir optimizasyon yöntemidir

■ Doğrusal Regresyonda hataların ortalamasını veren başarı değerlendirme metriği MEAN ABSOLUTE ERROR'dur.

■ Gradient Boosting machines; AdaBoost'un sınıflandırma ve regresyon problemlerine kolayca uyarlanabilen geliştirilmiş versiyonudur

■ Birden fazla karar ağacının ürettiği tahminlerin bir araya getirilerek değerlendirilmesine BOOTSTRAP AGGREGATING denir.

■ Benzer şeyler bir aradadır $\Rightarrow$ KNN

☑ Bir makine öğrenmesi modeli oluşturmak için gerekli iş akışını kodlamanın ve bu süreci otomatikleştirmenin adı MACHINE LEARNING PIPELINE' dir

Doğukan mehmet
KILINÇ