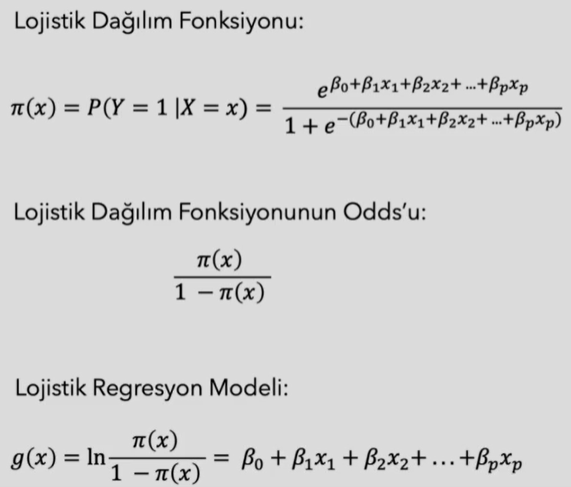
**Sınıflandırma Modelleri**

* Lojistik Regresyon
* Naive Bayes
* K-En Yakın Komşu (KNN)
* Destek Vektör Sınıflayıcı (SVC)
* Yapay Sinir Ağları (YSA)
* Sınıflandırma Ağaçları (CART)
* Random Forests Sınıflandırma (RF)
* Gragient Boosting Machines (GBM)
* Extreme Gradient Boosting (XGBoost)
* LightGBM
* CatBoost

1.Lojistik Regresyon (Teori):

* Amaç sınıflandırma problemleri için bağımlı ve bağımsız değişkenler arasındaki ilişkiyi tanımlayan doğrusal bir model kurmaktır.
* Bağımlı değişkenimiz (y) artık kategorik değişkendir.
* Kategorik yanıt değişkeninin tahmin edilmesine sınıflandırma denir.
* Amaç henüz gözlemlenmemiş bir x değer seti geldiğinde bunun sonucunda oluşacak sınıfı ortaya çıkarmaktır.
* Genellikle x in her bir kategoriye ait olma olasılıklarıyla ilgilenilir.
* Sınıf tahminlerinin iyiliği yanlış-doğru sınıflandırma oranıyla ölçülür.
* Şimdi yanıt değişkeni 1-0 olduğunda doğrusal regresyon kurulabilir fakat ikiden fazla değer alınacağı zaman kullanılamaz.
* Lojistik regresyonla modelinde kullanılan fonksiyondan dolayı tahmin edilen değerlerin 1 ile 0 arasında olması durumu garanti altına alınmış olur ve ilgilendiğimiz sınıfın 1 olması olasılığı ilgili olasılık değerleri hesaplamış olur. Değerlendirir.



* Dağılım fonksiyonunda x bağımsız değişkenlerin değerleri alındığında y bağımlı değişkenin 1. Sınıfa ait olma olasılığı sağdaki formül ile hesaplanır.
* Model kısmında doğrusal modelleme karşımıza çıkıyor.
* Lojistik Regresyon Modeli sınıflandırmanın atasıdır.

**Özellikler**

* Bağımlı değişken kategoriktir.
* Adını Bağımlı Değişkene uygulanan **Logit Dönüşümden** alır.
* Doğrusal regresyonda arana varsayımlar burada aranmadığı için daha esnek kullanılabilirliği vardır.
* Bağımlı değişkenin 1 olarak tanımlanan değerinin gerçekleşme olasılığı hesaplanır. Dolayısıyla bağımlı değişkenin alacağı değer ile ilgilenmez.
* Lojistik fonksiyonu sayesinde üretilen değerler 1-0 arasında olur.

1.Lojistik Regresyon (Model):

import numpy as np

import pandas as pd

import statsmodels.api as sm

import statsmodels.formula.api as smf

import seaborn as sns

from sklearn.preprocessing import scale

from sklearn.model\_selection import train\_test\_split, GridSearchCV, cross\_val\_score

from sklearn.metrics import confusion\_matrix, accuracy\_score, classification\_report

from sklearn.metrics import roc\_auc\_score,roc\_curve

import statsmodels.formula.api as smf

import matplotlib.pyplot as plt

from sklearn.neighbors import KNeighborsClassifier

from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis

from sklearn.linear\_model import LogisticRegression

from sklearn.svm import SVC

from sklearn.naive\_bayes import GaussianNB

from sklearn import tree

from sklearn.ensemble import RandomForestClassifier

from sklearn.ensemble import GradientBoostingClassifier

from xgboost import XGBClassifier

from lightgbm import LGBMClassifier

from catboost import CatBoostClassifier

from warnings import filterwarnings

filterwarnings('ignore')

* Genel import işlemleri.

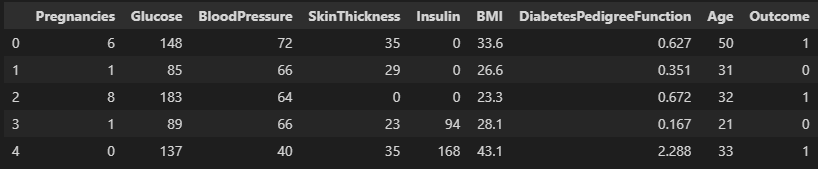
diabetes = pd.read\_csv("diabetes.csv")

df = diabetes.copy()

df = df.dropna()

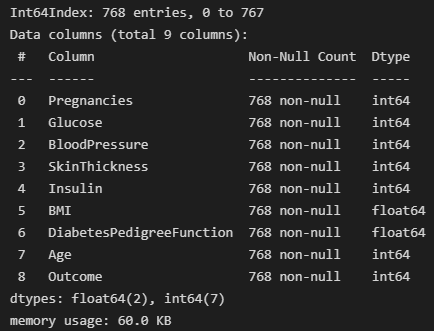
df.head()

* Veri setinin kopyası oluşturuldu, eksik gözlemler uçtu.



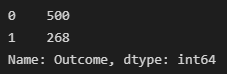
* Veri seti bir hasta geldiğinde onun şeker hastası olup olmadığını tahmin edebilecek makine öğrenmesi olur mu?
* Pregnancies: Daha önceki hamilelik sayısı
* BloodPressure: Kan basıncı.
* BMI: Vücut kitle indeksi.
* Outcome: Çıktı değişkenimiz. (Bağımlı değişken)

df.info()

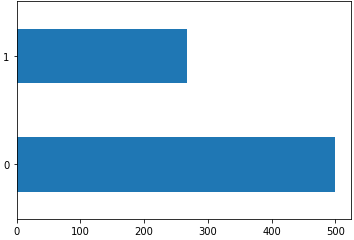


* 768 gözlem var.
* 2 kesirli, 7 tane tam sayılı değişken var. (Hepsi sürekli değişken)

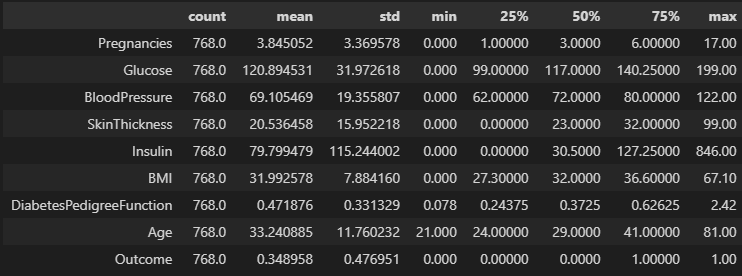
df["Outcome"].value\_counts()

* 1 sınıfında 268 kişi varken 0 sınıfında 500 kişi varmış.

df["Outcome"].value\_counts().plot.barh();



* Görselleştirme işlemi yaptık.

df.describe().T

* Betimsel istatistik incelendi.

y = df["Outcome"]

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

* Y sadece outcome alacak şekilde. (Bağımlı Değişken)
* X outcome olmayacak şekilde alındı. (Bağımsız Değişken)

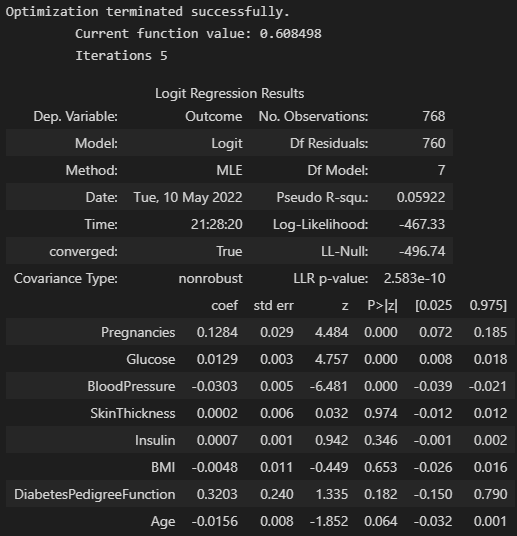
**Stats Model**

loj = sm.Logit(y, X)

loj\_model= loj.fit()

loj\_model.summary()

* Model kurup çıktısına bakalım.



**scikit-learn**

from sklearn.linear\_model import LogisticRegression

loj = LogisticRegression(solver = "liblinear")

loj\_model = loj.fit(X,y)

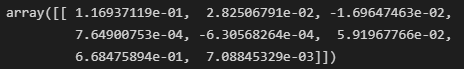
loj\_model

* Model kurup, fit işlemi gerçekleştirdik.

loj\_model.intercept\_

* Sabit değeri: -5.84073324

loj\_model.coef\_



* Bütün bağımsız değişkenlerin baş kat sayısını elde ettik.

1.Lojistik Regresyon (Tahmin & Tuning):

- Tuning edilebilecek bir şey yok doğrulama işlemi yapılabilir.

y\_pred = loj\_model.predict(X)

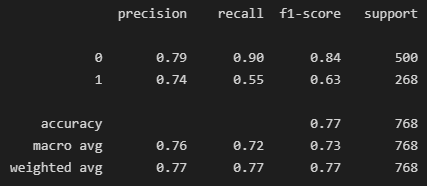
confusion\_matrix(y, y\_pred)

* Sol tarafta karmaşıklık matrisini oluşturduk. Gerçekte 1 iken 0 olan gerçekte 0 iken 0 olan vs.

accuracy\_score(y, y\_pred)

* Doğru sınıflandırma oranı: 0.7747395833333334
* % 77 denebilir.

print(classification\_report(y, y\_pred))



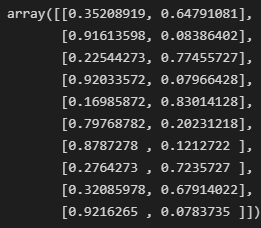
* Yukarıdaki tabloya bakarak değerlendirme işlemleri yapılabilir.

loj\_model.predict(X)[0:10]

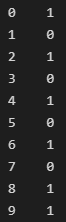


* Şimdi modelimiz bize tahmin ettiği değerleri veriyor. Mesela 0,7 olan değeri 1’e yuvarlıyor. 0,4 değerini 0’a yuvarlıyor. Şimdi doğru olduğunu kontrol edelim.

loj\_model.predict\_proba(X)[0:10][:,0:2]



* Şimdi sol taraftaki 0. olma olasılık değerleri, sağdaki 1. Sınıfın olasılık değerleri gösterir. Bizim ilgileneceğimiz sağ taraftaki tablo.
* Aşağıya bide gerçek değerleri getirelim.

y[0:10]

* Sağ taraftaki gerçek değerlerle yukarıdaki sağ taraftaki değerler karşılaştırdığımızda örnek 0. İndeks deki 0,64 değerini bir üsteki tahmin değerleri ile 1’e yuvarlamış. Sol taraftaki tabloya bakılınca 0. İndexde 1 olduğu için doğru tahmin gerçekleştirmiş. Mesela 1. İndexde 0,083 değerini 0’a diye tahmin etmiş ve sol taraftaki gerçek değerlerde 1. İndexde 0 olduğu görülür.
* Şimdi bu tahmin değerlerimizin 0’a veya 1’e yuvarlamak için bir eşik değer koyalım.

y\_probs = loj\_model.predict\_proba(X)

y\_probs = y\_probs[:,1]

y\_probs[0:10]

* y\_probs değişkenine tahmin edilen olasılıkların sadece 2. Satırını yani 1. İndexini aldık.

y\_pred = [1 if i > 0.5 else 0 for i in y\_probs]

y\_pred[0:10]

* y\_pred değişkenine içerisindeki değerlerden 0,5 den büyük olanları 1’e küçük olanları 0 a yuvarlama işlemi gerçekleştirdik.
* Şimdi ROC eğrisi oluşturalım.

confusion\_matrix(y, y\_pred)

accuracy\_score(y, y\_pred)

print(classification\_report(y, y\_pred))

loj\_model.predict\_proba(X)[:,1][0:5]

* Karmaşıklık matrisi , doğruluk skoru vs yukarıda bu kodlar anlatımı gerçekleşmiştir. Şimdi grafiğe dökelim.

logit\_roc\_auc = roc\_auc\_score(y, loj\_model.predict(X))

fpr, tpr, thresholds = roc\_curve(y, loj\_model.predict\_proba(X)[:,1])

plt.figure()

plt.plot(fpr, tpr, label='AUC (area = %0.2f)' % logit\_roc\_auc)

plt.plot([0, 1], [0, 1],'r--')

plt.xlim([0.0, 1.0])

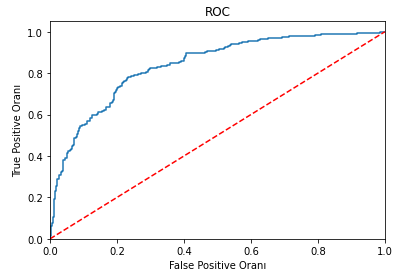
plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Oranı')

plt.ylabel('True Positive Oranı')

plt.title('ROC')

plt.show()



* Mavi çizgi modelin başarısını hesaplamak, değerledirmek için kullanacak olduğumuz eğri.
* Kırmızı çizgi hiçbir modelleme yapmasaydık zaten elde edeceğimiz başarıyı ifade etmektedir.
* Mavi çizgi ile kırmızı çizgi arasındaki alan başarı oranımızdır.
* Şimdi model doğrulama işlemlerine geçelim. Diğer bölümlerle alışık olduğumuz sistemle hata bulacağız, diğer modeller ile karşılaştırmak için.

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y, test\_size = 0.30, random\_state = 42)

* Test- Train ayırma işlemi gerçekleştirdik.

loj = LogisticRegression(solver = "liblinear")

loj\_model = loj.fit(X\_train,y\_train)

loj\_model

* x\_train ve y\_train üzerinden model kurma işlemi gerçekleşti.

accuracy\_score(y\_test, loj\_model.predict(X\_test))

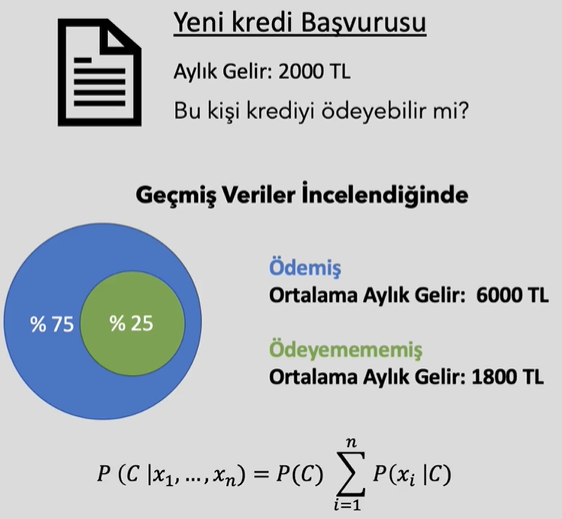
* Test hatamız: 0.7532467532467533

cross\_val\_score(loj\_model, X\_test, y\_test, cv = 10).mean()

* Cross validation yapılmış test hatası: 0.7748188405797102

2.Naive Bayes (Teori):

* Olasılık temelli bir modelleme tekniğidir. Amaç belirli bir örneğin her bir sınıfa ait olma olasılığının koşullu olasılık temelli hesaplanmasıdır.



* Mesela yukarıda 2000 lira maaş alan adam var. Kredi istiyor. Ortalama geliri 6000 lira olan kişiler %75, ortalama geliri 1800 lira olan kişiler %25. Bunlara kıyasla bu kişinin krediyi ödeyip ödeyemediği hesaplanır.

2.Naive Bayes (Model & Tahmin):

df = diabetes.copy()

df = df.dropna()

y = df["Outcome"]

X = df.drop(['Outcome'], axis=1)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.30, random\_state=42)

* Test-Train işlemleri yapıldık. Yalnız burada %70’e %30’luk orana göre ayrılmıştır.
* Test-Train ayırma işlemleri yapılırken kaç tane 0 var kaç tane 1 var vs. onları göz önünde bulundurarak ayrılmalı.

from sklearn.naive\_bayes import GaussianNB

nb = GaussianNB()

nb\_model = nb.fit(X\_train, y\_train)

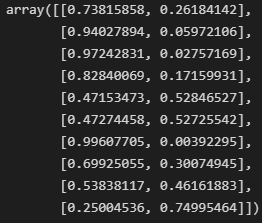
nb\_model

* Model kurma işlemleri gerçekleştirildi.

nb\_model.predict(X\_test)[0:10]

* Tahmin işlemi gerçekleştirildi.

nb\_model.predict\_proba(X\_test)[0:10]



* Olasılık değerleri yan tarafta verilmiştir.

y\_pred = nb\_model.predict(X\_test)

accuracy\_score(y\_test, y\_pred)

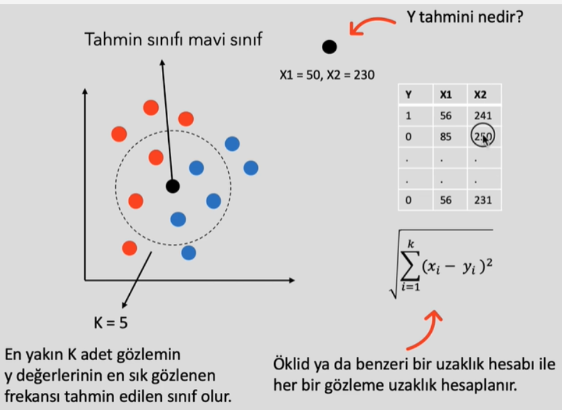
* Test hatamız: 0.7445887445887446

cross\_val\_score(nb\_model, X\_test, y\_test, cv = 10).mean()

* Doğrulanmış test hatamız: 0.775

3.K-En Yakın Komşu (KNN) (Teori):

* Tahminler gözlem benzerliğine göre yapılır. Bana arkadaşını söyle sana kim olduğunu söyleyeyim. :)



* En yakın komşularına uzaklıklarına bakarak tahmin yapar.

**KNN Basamakları**

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

3.K-En Yakın Komşu (KNN) (Model & Tahmin):

diabetes = pd.read\_csv("diabetes.csv")

df = diabetes.copy()

df = df.dropna()

y = df["Outcome"]

X = df.drop(['Outcome'], axis=1)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y,

test\_size=0.30, random\_state=42)

* Test- Train işlemleri gerçekleştirildi.

knn = KNeighborsClassifier()

knn\_model = knn.fit(X\_train, y\_train)

knn\_model

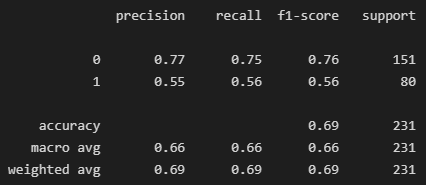
* Model kurulup fit edildi.

y\_pred = knn\_model.predict(X\_test)

accuracy\_score(y\_test, y\_pred)

* İlkel Hata değerimiz: 0.6883116883116883
* Çok fazla sürekli değişken olduğu için şaşırtmadı.

print(classification\_report(y\_test, y\_pred))



* Yan tarafta detaylı çıktımız verildi.

3.K-En Yakın Komşu (KNN) (Model Tuning):

knn\_params = {"n\_neighbors": np.arange(1,50)}

* Komşu sayımız 1 den 50 ye kadar olan değerleri alsın.

knn = KNeighborsClassifier()

knn\_cv = GridSearchCV(knn, knn\_params, cv=10)

knn\_cv.fit(X\_train, y\_train)

* Komşu sayısı parametresini değiştirip deneyecek.

print("En iyi skor:" + str(knn\_cv.best\_score\_))

print("En iyi parametreler: " + str(knn\_cv.best\_params\_))

* En iyi skor: 0.748637316561845
* En iyi parametreler: 11

knn = KNeighborsClassifier(11)

knn\_tuned = knn.fit(X\_train, y\_train)

knn\_tuned.score(X\_test, y\_test)

* Final modeli kurup hata değerimiz: 0.7316017316017316

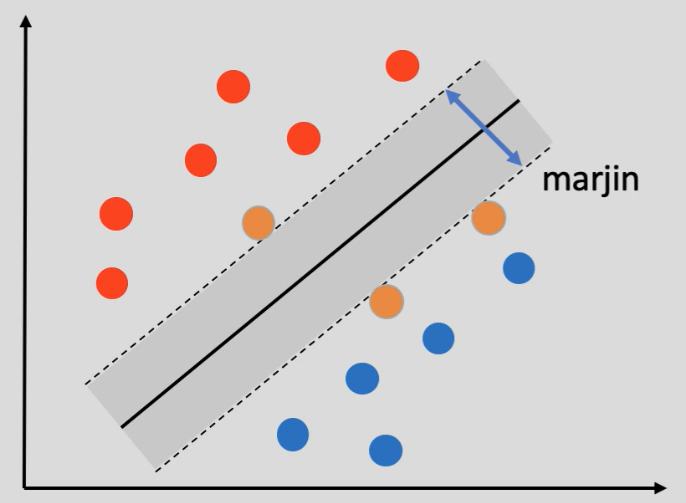
y\_pred = knn\_tuned.predict(X\_test)

accuracy\_score(y\_test, y\_pred)

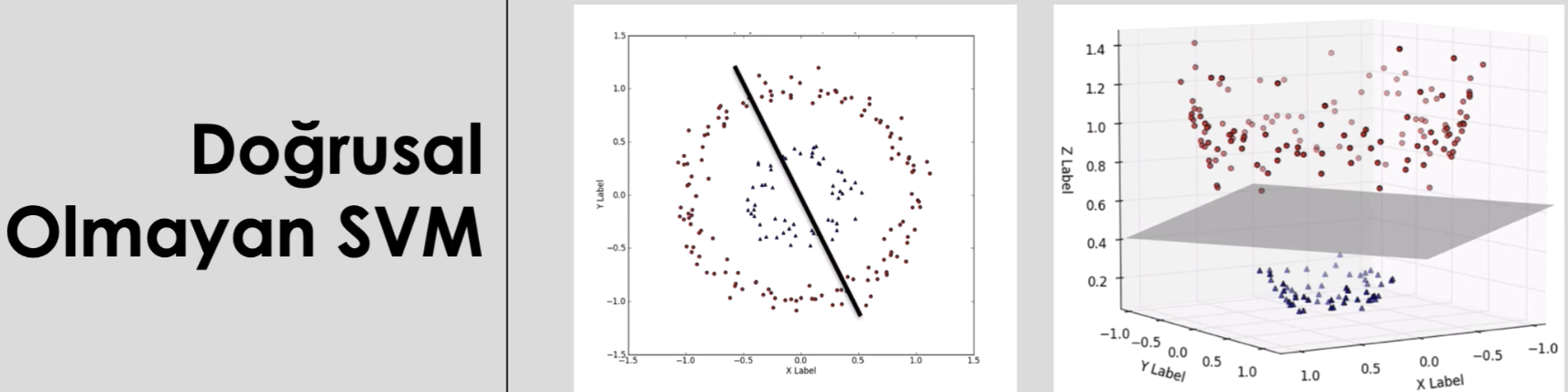
* Bu şekilde de final hatamıza ulaşılabilir. Sonuç aynı.

4.Destek Vektör Makineleri (SVM) (Teori):

* Amaç iki sınıf arasındaki ayrımın optimum olmasını sağlayacak hiper – düzlemi bulmaktır.



* Marjin: Hiper düzlem ile en yakın veri noktası arasındaki alandır.
* Hiperdüzlem : İki doğru arasında nokta olmayacak şekilde aralarındaki mesafenin maksimum tutulması şekliyle sağlanır.



* Doğrusal olmayan sınıflarda farklı yöntemler ile bu ayrıştırma işlemi gerçekleştirilir.

4.Destek Vektör Makineleri (SVM) (Model & Tahmin):

df = diabetes.copy()

df = df.dropna()

y = df["Outcome"]

X = df.drop(['Outcome'], axis=1)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.30, random\_state=42)

* Test-Train işlemleri yapıldık. Yalnız burada %70’e %30’luk orana göre ayrılmıştır.

svm\_model = SVC(kernel = "linear").fit(X\_train, y\_train)

* Model kurma işlemi ve fit etme işlemi gerçekleştirildi.
* Lineer SVC isimli farklı bir fonksiyonda var.

y\_pred = svm\_model.predict(X\_test)

accuracy\_score(y\_test, y\_pred)

* Test Hatamız: 0.7445887445887446

4.Destek Vektör Makineleri (SVM) (Model Tuning):

svc\_params = {"C": np.arange(1,10)}

svc = SVC(kernel = "linear")

svc\_cv\_model = GridSearchCV(svc,svc\_params,

                            cv = 10,

                            n\_jobs = -1,

                            verbose = 2 )

svc\_cv\_model.fit(X\_train, y\_train)

* C parametrelerini model üzerinde deneyip en iyi sonucu verecektir.
* C değeri 0 ile 10 arasında olmaz , 1 ile başlatmanız gerekir.

print("En iyi parametreler: " + str(svc\_cv\_model.best\_params\_))

* En iyi parametre değeri: C=5

svc\_tuned = SVC(kernel = "linear", C = 5).fit(X\_train, y\_train)

* Final modeli oluşturduk.

y\_pred = svc\_tuned.predict(X\_test)

accuracy\_score(y\_test, y\_pred)

* Final modeli Test Hatası : 0.7445887445887446

4.Doğrusal Olmayan Destek Vektör Makineleri (RBF) (SVM) (Model & Tahmin):

svc\_model = SVC(kernel = "rbf").fit(X\_train, y\_train)

* Normalde kernel’in ön tanımlı değeri rbf dir. Biz önceki bölümde linear girerek faklı ön tanımla denedik.
* Model kurulup fit edildi.

dir(svc\_model)

* Dir () ile alınabilecek argümanları gösterebiliriz.

y\_pred = svc\_model.predict(X\_test)

accuracy\_score(y\_test, y\_pred)

* Test hatamız: 0.7359307359307359

4.Doğrusal Olmayan Destek Vektör Makineleri (RBF) (SVM) (Model Tuning):

svc\_params = {"C": [0.0001, 0.001, 0.1, 1, 5, 10 ,50 ,100],

"gamma": [0.0001, 0.001, 0.1, 1, 5, 10 ,50 ,100]}

* C: Aykırı gözlemlerin göreceli olarak etkilerini kontrol altında tutarak oluşacak olan düzlemin bize oluşmasını kontrol imkânı sağlıyor.
* Gamma: Oluşacak model üzerinde etkisi fazladır.

svc = SVC()

svc\_cv\_model = GridSearchCV(svc, svc\_params,

                         cv = 10,

                         n\_jobs = -1,

                         verbose = 2)

svc\_cv\_model.fit(X\_train, y\_train)

* Parametrelerin deneme işlemleri.
* Ön tanımlı değer rbf olduğu için girmemize gerek yok.

print("En iyi parametreler: " + str(svc\_cv\_model.best\_params\_))

* En iyi parametreler: C=10 gamma = 0.0001

svc\_tuned = SVC(C = 10, gamma = 0.0001).fit(X\_train, y\_train)

* Final Modelimizi kurduk.

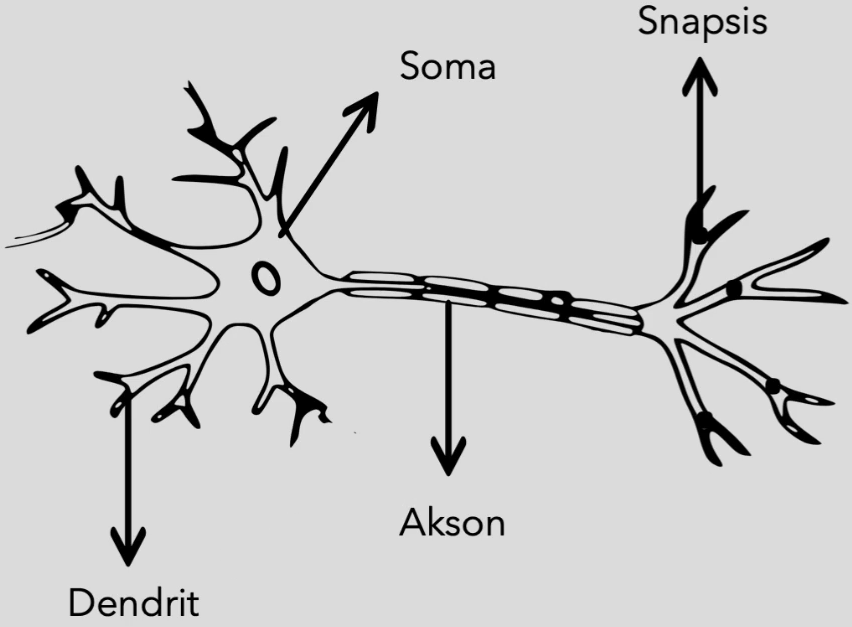
y\_pred = svc\_tuned.predict(X\_test)

accuracy\_score(y\_test, y\_pred)

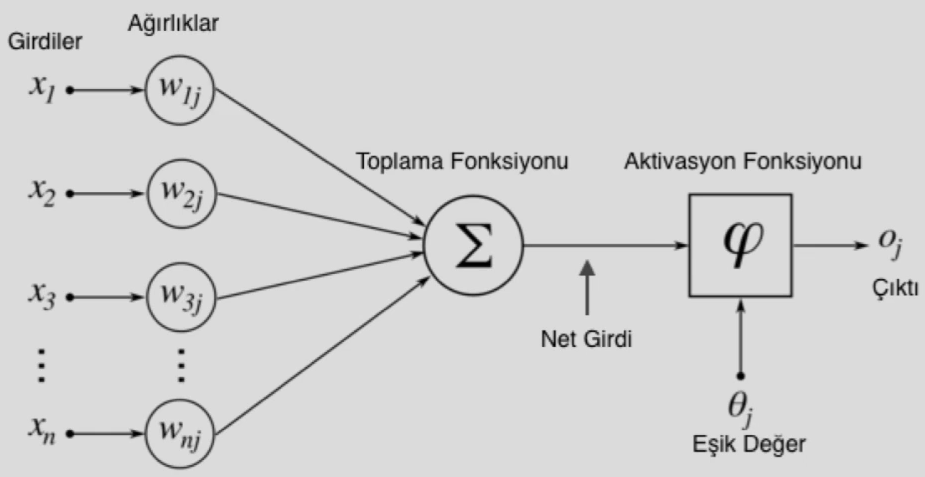
* Final modeli Test hatası: 0.7359307359307359

5.Yapay Sinir Ağları (YSA) (Teori):

* İnsan beyninin bilgi işleme şeklini referans alan sınıflandırma ve regresyon problemleri için kullanılabilen kuvvetli bir makine öğrenmesi algoritmalarından birisidir.

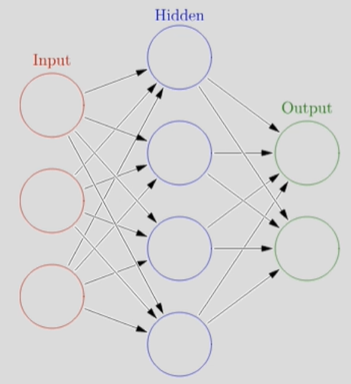


* Gerçek bir sinir hücresi yukarıda verilmiştir.
* Dendritler hücrelere gelen sinirleri çekirdeğe iletir, soma iletilen sinirleri orta kısımda toplar. Akson somadan gelen sinirleri diğer hücrelere iletir. Sinyaller diğer hücrelere aktarılmadan önce snapsislerde bir önişlemeye tabi tutuluyor. Sinapsislerin görevi gelen sinyalleri belirli bir eşik değere denk getirecek şekilde değiştirip işleyip sonrasında diğer hücreye aktarır.

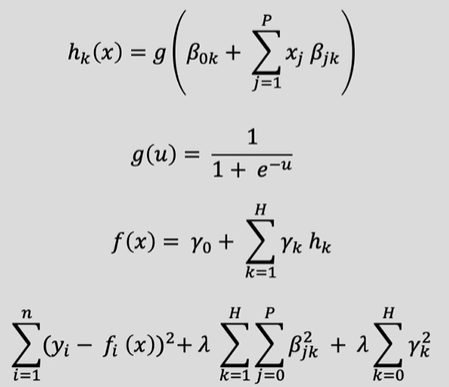


* Yapay sinir hücresi yukarıda verilmiştir.
* Girdiler (Dentritler) : Bağımsız değişkenlerin değerleridir.
* Ağırlıklar: Başkatsayıları olarak düşünülebilir.
* Toplama Fonksiyonu (Soma): Hesaplamalar.
* Bilgi net girdi (Aksonlar) kısmıyla Aktivasyon Fonksiyonu (Sinapsis) kısmına geliyor.

Amaç her zamanki gibi aynı hataları minimum yapacak şekilde bir matematiksel form veya kural seti çıkarmak. Katsayılar değiştiriliyor ve sürekli geri yayılıp kat sayıları değiştirip minimum hata elde edene kadar devam eder. Ağırlıklar(katsayılar) değişmesi esnasında aslında öğrenme işlemi gerçekleşmiş olur. Bu yapay sinir hücreleri bir araya gelerek **yapay sinir ağlarını** oluşturuyor.



* Yukarıda yapay sinir ağları gösterilmiştir.
* Bu ağı oluşturan birimlere nöron adı verilir.
* İnput (Girdi Katmanı) , Hidden (Gizli Katmanı) Output (Çıktı kavramı)
* Çok Katmanlı algılayıcılar olarak da geçer.
* En çok kullanılan geriye yayılımlı ağlardır. Çalışma prensibi kabul edilebilir bir hata miktarı ağ çıktısıyla gerçek çıktısı arasındaki farkları minimize etmek için girdilerin ağırlıklarını değiştirmektir.



* Genel matematiksel formülü bu şekildedir.

5.Yapay Sinir Ağları (YSA) (Model & Tahmin):

diabetes = pd.read\_csv("diabetes.csv")

df = diabetes.copy()

df = df.dropna()

y = df["Outcome"]

X = df.drop(['Outcome'], axis=1)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.30, random\_state=42)

* Test-Train işlemleri yapıldık. Yalnız burada %70’e %30’luk orana göre ayrılmıştır.

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

* Değişkenlerin ölçeklerine ve bunların birbirleri aralarındaki durumlardan etkilendiği için standartlaştırma işlemi yaptık.

scaler.fit(X\_train)

X\_train\_scaled = scaler.transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

* Dönüşüm işlemleri gerçekleştirildi.

from sklearn.neural\_network import MLPClassifier

mlpc = MLPClassifier().fit(X\_train\_scaled, y\_train)

* Model kurma ve fit işlemi gerçekleştirildi.

y\_pred = mlpc.predict(X\_test\_scaled)

accuracy\_score(y\_test, y\_pred)

* Test hatamız : 0.7229437229437229

5.Yapay Sinir Ağları (YSA) (Model Tuning):

mlpc\_params = {"alpha": [0.1, 0.01, 0.02, 0.005, 0.0001,0.00001],

              "hidden\_layer\_sizes": [(10,10,10),

                                     (100,100,100),

                                     (100,100),

                                     (3,5),

                                     (5, 3)],

              "solver" : ["lbfgs","adam","sgd"],

              "activation": ["relu","logistic"]}

* **Hidden\_layer\_sizes:** Bir değer varsa bir katmanlı olacağı ve o değer kadar sinir hücresi olacağını ifade eder. 2 değer varsa 2 katmanlı ve 2. Değer kadar hücresi olacağını ifade eder.
* **Activation:** Yapıyı doğrusal olmayan yapay sinir ağı regresyon modeli gibi değerleri oynar.
* **Solver:** Ağırlık (Katsayı) optimum değerlerini bulmasını sağlar.
* **Alpha:** Eskilerden hatırlanacak olursa cezalandırma parametresi.

mlpc = MLPClassifier()

mlpc\_cv\_model = GridSearchCV(mlpc, mlpc\_params,

                         cv = 10,

                         n\_jobs = -1,

                         verbose = 2)

mlpc\_cv\_model.fit(X\_train\_scaled, y\_train)

* Deneme işlemleri gerçekleştirildi.

print("En iyi parametreler: " + str(mlpc\_cv\_model.best\_params\_))

* En iyi parametreler aşağıda verilmiştir.
* {'activation': 'relu', 'alpha': 0.01, 'hidden\_layer\_sizes': (100, 100, 100), 'solver': 'sgd'}

mlpc\_tuned = MLPClassifier(activation = "relu",

              alpha = 0.01,

              hidden\_layer\_sizes = (100, 100, 100),

              solver = "sgd")

mlpc\_tuned.fit(X\_train\_scaled, y\_train)

* Final modelimizi kurduk.

y\_pred = mlpc\_tuned.predict(X\_test\_scaled)

accuracy\_score(y\_test, y\_pred)

* Final test hatamız: 0.7489177489177489

6.Sınıflandırma Ağaçları (CART) (Teori):

* Amaç veri seti içerisindeki karmaşık yapıları basit karar yapılarına dönüştürmektir.
* Heterojen veri setleri belirlenmiş bir hedef değişkenine göre homojen alt gruplara ayrılır.



* Yukarıda basit olarak çalıştırma sistemi gösterilmiştir. Bu şekilde sınıflandırma işlemi gerçekleştirilir.

6.Sınıflandırma Ağaçları (CART) (Model & Tahmin):

diabetes = pd.read\_csv("diabetes.csv")

df = diabetes.copy()

df = df.dropna()

y = df["Outcome"]

X = df.drop(['Outcome'], axis=1)

#X = df["Pregnancies"]

X = pd.DataFrame(X)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y,

test\_size=0.30, random\_state=42)

* Test- train işlemleri gerçekleştirdik. İstenirse tek değişken içinde tahmin yapılabilir. Detaylar 354. Video

from sklearn.tree import DecisionTreeClassifier

cart = DecisionTreeClassifier()

cart\_model = cart.fit(X\_train, y\_train)

* Model kurma işlemi gerçekleştirdik.

from skompiler import skompile

print(skompile(cart\_model.predict).to("python/code"))

* İstenirse tahmin kodlarına yukarıdaki kodla ulaşılabilir.

y\_pred = cart\_model.predict(X\_test)

accuracy\_score(y\_test, y\_pred)

* Test hatamız: 0.70995670995671

6.Sınıflandırma Ağaçları (CART) (Model Tuning):

cart\_grid = {"max\_depth": range(1,10),

"min\_samples\_split" : list(range(2,50)) }

* Max\_depth : Maximum derinlik. Karmaşıklık kontrolü için parametredir.
* Min\_samples\_split : Budama sistemi. Bölünme için minimum işlem sayısı. Belli bölünme işleminden sonra bölünüp bölünmeyeceğine kadar veren argüman.

cart = tree.DecisionTreeClassifier()

cart\_cv = GridSearchCV(cart, cart\_grid, cv = 10, n\_jobs = -1, verbose = 2)

cart\_cv\_model = cart\_cv.fit(X\_train, y\_train)

* Deneme işlemleri gerçekleştirildi.

print("En iyi parametreler: " + str(cart\_cv\_model.best\_params\_))

* En iyi parametreler aşağıda verilmiştir.
* {'max\_depth': 5, 'min\_samples\_split': 19}

cart = tree.DecisionTreeClassifier(max\_depth = 5, min\_samples\_split = 19)

cart\_tuned = cart.fit(X\_train, y\_train)

* Final modeli oluşturuldu , fit edildi.

y\_pred = cart\_tuned.predict(X\_test)

accuracy\_score(y\_test, y\_pred)

* Test hatamız: 0.7532467532467533

7.Random Forests Sınıflandırma (RF) (Teori):

* Temeli birden çok karar ağacının ürettiği tahminlerin bir araya getirilerek değerlendirilmesine dayanır.

**Özellikler**

* Bagging (Breiman, 1996) ile Random Subspace (Ho, 1998) yöntemlerinin birleşimi ile oluşturulmuştur.
* Ağaçlar için gözlemler bootstrap rasgele örnek seçim yöntemi ile değişkenler **random subspace yöntemi** ile seçilir.
* Karar ağacının her bir düğümünde en iyi dallara ayırıcı (bilgi kazancı) değişken tüm değişkenler arasında rastgele seçilen daha az sayıdaki değişken arasından seçilir.
* Ağaç oluşturmada veri setinin 2/3’ü kullanılır. Dışarıda kalan veri ağaçların performans değerlendirmesi ve değişken öneminin belirlenmesi için kullanılır.
* Her düğüm noktasında rastgele değişken seçimi yapılır.

(Regresyonda p/3, sınıflandırmada karekök p)

* Nihai tahmin için ağaçlardan tahmin değerleri talep edilirken her bir ağacın daha önce hesaplanan hata oranları göz önüne alınarak ağaçlara ağırlık verilir.

7.Random Forests Sınıflandırma (RF) (Model & Tahmin):

diabetes = pd.read\_csv("diabetes.csv")

df = diabetes.copy()

df = df.dropna()

y = df["Outcome"]

X = df.drop(['Outcome'], axis=1)

X = pd.DataFrame(X)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.30, random\_state=42)

* Test-Train işlemleri yapıldık.

from sklearn.ensemble import RandomForestClassifier

rf\_model = RandomForestClassifier().fit(X\_train, y\_train)

* Model kurulup fit edildi.

y\_pred = rf\_model.predict(X\_test)

accuracy\_score(y\_test, y\_pred)

* Test hatamız: 0.7445887445887446

7.Random Forests Sınıflandırma (RF) (Model Tuning):

rf\_params = {"max\_depth": [2,5,8,10],

            "max\_features": [2,5,8],

            "n\_estimators": [10,500,1000],

            "min\_samples\_split": [2,5,10]}

* Denenecek olan parametreleri girdik.

rf\_model = RandomForestClassifier()

rf\_cv\_model = GridSearchCV(rf\_model,

                           rf\_params,

                           cv = 10,

                           n\_jobs = -1,

                           verbose = 2)

rf\_cv\_model.fit(X\_train, y\_train)

* Deneme işlemi gerçekleştirildi.

print("En iyi parametreler: " + str(rf\_cv\_model.best\_params\_))

* **En iyi parametreler:** {'max\_depth': 8, 'max\_features': 8, 'min\_samples\_split': 2, 'n\_estimators': 1000}

rf\_tuned = RandomForestClassifier(max\_depth = 8,

                                  max\_features = 8,

                                  min\_samples\_split = 2,

                                  n\_estimators = 1000)

rf\_tuned.fit(X\_train, y\_train)

* Final modelimizi kurduk.

y\_pred = rf\_tuned.predict(X\_test)

accuracy\_score(y\_test, y\_pred)

* Final test hatamız: 0.7489177489177489

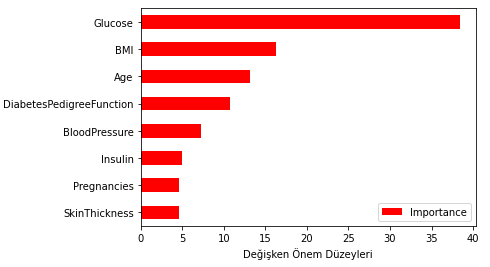
Importance = pd.DataFrame({"Importance": rf\_tuned.feature\_importances\_\*100},index = X\_train.columns)

Importance.sort\_values(by = "Importance", axis = 0,

ascending = True).plot(kind ="barh", color = "r")

plt.xlabel("Değişken Önem Düzeyleri")

* Değişkenlerin önem düzeyini grafiğe döktük.



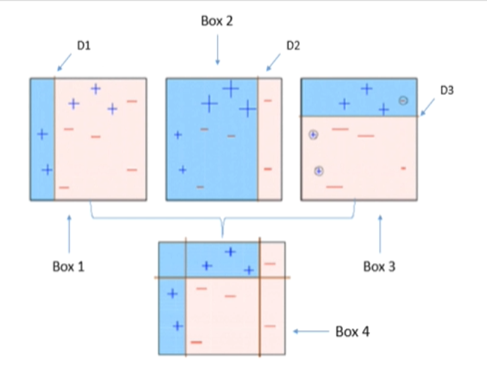
* Değişkenlerin önemlerine baktığımızda Glikoz değeri , Yaş önemli faktör olarak dururken hamilelik geçirmiş olması biraz daha az önem arz ediyormuş.

8.Gradient Boosting Machines (GBM) (Teori):

* Zayıf öğrencileri bir araya getirip güçlü bir öğrenci ortaya çıkarma fikrine dayanır.
* AdaBoost’un sınıflandırma ve regresyon problemlerine kolayca uyarlanabilen genelleştirilmiş versiyonudur.
* Atıklar üzerinde tek bir tahminsel model formunda olan modeller serisi kurulur.
* Zayıf öğrencileri bir araya getirip güçlü bir öğrenci ortaya çıkarmak fikrine dayanır.

**AdaBoost**

* Zayıf sınıflandırıcıların bir araya gelerek güçlü bir sınıflandırıcı oluşturulması fikrini hayata geçiren algoritmadır.



* Yukarıdaki grafikte 1-2-3-4 sistem gerçekleştirilmiş. 1. De + lar ayrılmış olsada (- +) karışık şekilde. 4.de Evrimleşerek sınıflandırılıp – ve + lar birbirinden tamamen ayrılmıştır.

**Özellikler (GBF)**

* 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) oluşturur.
* GBM diferansiyellenebilen herhangi bir kayıp fonksiyonunu optimize edebilen Gradient descent algoritmasını kullanmakta.
* GB birçok temel öğrenici tipi (base learner type) kullanabilir. (Trees, linear terms, splines, …)
* Cost fonksiyonları ve link fonksiyonları modifiye edilebilirdir.
* Boosting + Gradient Descent

8.Gradient Boosting Machines (GBM) (Model & Tahmin):

diabetes = pd.read\_csv("diabetes.csv")

df = diabetes.copy()

df = df.dropna()

y = df["Outcome"]

X = df.drop(['Outcome'], axis=1)

X = pd.DataFrame(X)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.30, random\_state=42)

* Test-Train işlemleri yapıldık.

from sklearn.ensemble import GradientBoostingClassifier

gbm\_model = GradientBoostingClassifier().fit(X\_train, y\_train)

* Model kurulup fit edildi.

y\_pred = gbm\_model.predict(X\_test)

accuracy\_score(y\_test, y\_pred)

* İlkel test hatası: 0.7445887445887446

8.Gradient Boosting Machines (GBM) (Model Tuning):

gbm\_params = {"learning\_rate" : [0.001, 0.01, 0.1, 0.05],

             "n\_estimators": [100,500,100],

             "max\_depth": [3,5,10],

             "min\_samples\_split": [2,5,10]}

* Denenecek Parametre değerlerimizi girdik.

gbm = GradientBoostingClassifier()

gbm\_cv = GridSearchCV(gbm, gbm\_params, cv = 10, n\_jobs = -1, verbose = 2)

gbm\_cv.fit(X\_train, y\_train)

* Deneme işlemleri gerçekleştirildi.

print("En iyi parametreler: " + str(gbm\_cv.best\_params\_))

**En iyi parametreler:** {'learning\_rate': 0.01, 'max\_depth': 3, 'min\_samples\_split': 5, 'n\_estimators': 500}

gbm = GradientBoostingClassifier(learning\_rate = 0.01,

                                 max\_depth = 3,

                                min\_samples\_split = 5,

                                n\_estimators = 500)

gbm\_tuned =  gbm.fit(X\_train,y\_train)

* Final modelimizi kurduk.

y\_pred = gbm\_tuned.predict(X\_test)

accuracy\_score(y\_test, y\_pred)

* Final Test Hatamız: 0.7489177489177489

9.eXtreme Gradient Boosting (XGBoost) (Teori):

* XGBoost, GBM’in hız ve tahmin performansını arttırmak üzere optimize edilmiş; ölçeklenebilir ve farklı platformlara entegre edilebilir halidir.

**Özellikler (XGBoost)**

* R, Python, Hadoop, Scala, Julia ile kullanılabilir.
* Ölçeklenebilir.
* Hızlıdır.
* Tahmin başarısı yüksektir.
* Birçok kaggle yarışmasında başarısını kanıtlamıştır.

9.eXtreme Gradient Boosting (XGBoost) (Model & Tahmin):

diabetes = pd.read\_csv("diabetes.csv")

df = diabetes.copy()

df = df.dropna()

y = df["Outcome"]

X = df.drop(['Outcome'], axis=1)

X = pd.DataFrame(X)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.30, random\_state=42)

* Test-Train işlemleri yapıldık.

from xgboost import XGBClassifier

xgb\_model = XGBClassifier().fit(X\_train, y\_train)

* Model kuruldu.

y\_pred = xgb\_model.predict(X\_test)

accuracy\_score(y\_test, y\_pred)

* İlkel test hatası: 0.7359307359307359

9.eXtreme Gradient Boosting (XGBoost) (Model Tuning):

xgb\_params = {

        'n\_estimators': [100, 500, 1000, 2000],

        'subsample': [0.6, 0.8, 1.0],

        'max\_depth': [3, 4, 5,6],

        'learning\_rate': [0.1,0.01,0.02,0.05],

        "min\_samples\_split": [2,5,10]}

* Değerlendirilecek olan parametre değerlerimizi girdik.

xgb = XGBClassifier()

xgb\_cv\_model = GridSearchCV(xgb, xgb\_params, cv = 10, n\_jobs = -1, verbose = 2)

xgb\_cv\_model.fit(X\_train, y\_train)

* Değerlendirme işlemleri gerçekleştirildi.

xgb\_cv\_model.best\_params\_

**En iyi parametreler:** {'learning\_rate': 0.02, 'max\_depth': 3, 'min\_samples\_split': 2,'n\_estimators': 100, 'subsample': 0.6}

xgb = XGBClassifier(learning\_rate = 0.02,

max\_depth = 3, min\_samples\_split = 2,

n\_estimators = 100, subsample = 0.6)

xgb\_tuned =  xgb.fit(X\_train,y\_train)

* Final modelimizi kurduk.

y\_pred = xgb\_tuned.predict(X\_test)

accuracy\_score(y\_test, y\_pred)

* Final test hatamız: 0.7575757575757576

10. Light GBM (Teori):

* Light GBM , XGBoost un eğitim süresi performansını arttırmaya yönelik geliştirilen bir diğer GBM türüdür.
* Microsoft tarafından 2017’de kurulmuştur.

**Özellikler (Light GBM)**

* Daha performanslı.
* Level-wise büyüme stratejisi yerine Leaf-wise büyüme stratejisi kullanır.
* Level-wise (XGBoost): Değişkenleri ayırma, farklı bölme noktalarını araştırmak, keşfetmek için karar ağaçlarını kullanır. (Ağaç Büyüme Stratejisi)
* Leaf-wise (Light GBM): Daha iyi fit edebilmek için script(senaryo) odaklanıyor.
* Genel olarak bakıldığında XGBoost daha kapsamlı ilk arama işlemi (Breadth-First Search (BFS)) yapar. Light GBM ise derinlemesine ilk arama (Depth-First Search (DFS)) yapar.

10. Light GBM (Model & Tahmin):

diabetes = pd.read\_csv("diabetes.csv")

df = diabetes.copy()

df = df.dropna()

y = df["Outcome"]

X = df.drop(['Outcome'], axis=1)

X = pd.DataFrame(X)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.30, random\_state=42)

* Test – Train işlemleri gerçekleştirildi.

from lightgbm import LGBMClassifier

lgbm\_model = LGBMClassifier().fit(X\_train, y\_train)

* Modeli kurduk.

y\_pred = lgbm\_model.predict(X\_test)

accuracy\_score(y\_test, y\_pred)

* İlkel Test Hatamız: 0.7229437229437229

10. Light GBM (Model Tuning):

lgbm\_params = {

        'n\_estimators': [100, 500, 1000, 2000],

        'subsample': [0.6, 0.8, 1.0],

        'max\_depth': [3, 4, 5,6],

        'learning\_rate': [0.1,0.01,0.02,0.05],

        "min\_child\_samples": [5,10,20]}

* Parametre değerlerimiz.

A

lgbm = LGBMClassifier()

lgbm\_cv\_model = GridSearchCV(lgbm, lgbm\_params, cv = 10, n\_jobs = -1,verbose = 2)

lgbm\_cv\_model.fit(X\_train, y\_train)

* Deneme işlemleri gerçekleştirildi. XGBoost a göre aynı parametre sayısını yaklaşık yarısı kadar sürede tahmin etti.

lgbm\_cv\_model.best\_params\_

**En iyi parametrelerimiz:** learning\_rate: 0.05, 'max\_depth': 3, 'min\_child\_samples': 20, 'n\_estimators': 100, 'subsample': 0.6}

lgbm = LGBMClassifier(learning\_rate = 0.05,

                       max\_depth = 3,

                       subsample = 0.6,

                       n\_estimators = 100,

                       min\_child\_samples = 20)

lgbm\_tuned = lgbm.fit(X\_train,y\_train)

* Final modelimizi kurduk.

y\_pred = lgbm\_tuned.predict(X\_test)

accuracy\_score(y\_test, y\_pred)

* Test hatamız: 0.7489177489177489

11. Category Boosting (CatBoost) (Teori):

* Kategorik değişkenler ile otomatik olarak mücadele edebilen, hızlı, başarılı bir diğer GBM türevi.
* Yandex tarafından 2017’de kurulmuştur.

**Özellikler (Light GBM)**

* Kategorik değişken desteği
* Hızlı ve ölçeklenebilir GPU desteği
* Daha başarılı tahminler
* Rusya’nın ilk açık kodlu, başarılı Makine Öğrenmesi çalışması.

11. Category Boosting (CatBoost) (Model & Tahmin):

diabetes = pd.read\_csv("diabetes.csv")

df = diabetes.copy()

df = df.dropna()

y = df["Outcome"]

X = df.drop(['Outcome'], axis=1)

X = pd.DataFrame(X)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.30, random\_state=42)

* Test – Train işlemleri gerçekleştirildi.

from catboost import CatBoostClassifier

cat\_model = CatBoostClassifier().fit(X\_train, y\_train)

* Model kurulup fit edildi.

y\_pred = cat\_model.predict(X\_test)

accuracy\_score(y\_test, y\_pred)

* İlkel test hatası: 0.7402597402597403

11. Category Boosting (CatBoost) (Model Tuning):

catb\_params = {

    'iterations': [200,500],

    'learning\_rate': [0.01,0.05, 0.1],

    'depth': [3,5,8] }

* Denenecek parametreler girildi.

catb = CatBoostClassifier()

catb\_cv\_model = GridSearchCV(catb, catb\_params, cv=5, n\_jobs = -1, verbose = 2)

catb\_cv\_model.fit(X\_train, y\_train)

* Deneme işlemi gerçekleştiriliyor.

catb\_cv\_model.best\_params\_

* **En iyi Parametreler:** {'depth': 8, 'iterations': 200, 'learning\_rate': 0.01}

catb = CatBoostClassifier(iterations = 200,

learning\_rate = 0.05,depth = 5)

catb\_tuned = catb.fit(X\_train, y\_train)

y\_pred = catb\_tuned.predict(X\_test)

* Final modelimiz kuruldu.

y\_pred = catb\_tuned.predict(X\_test)

accuracy\_score(y\_test, y\_pred)

* Final Test Hatamız: 0.7532467532467533

Tüm Modellerin Karşılaştırılması

modeller = [

    knn\_tuned,

    loj\_model,

    svc\_tuned,

    nb\_model,

    mlpc\_tuned,

    cart\_tuned,

    rf\_tuned,

    gbm\_tuned,

    catb\_tuned,

    lgbm\_tuned,

    xgb\_tuned

]

for model in modeller:

    isimler = model.\_\_class\_\_.\_\_name\_\_

    y\_pred = model.predict(X\_test)

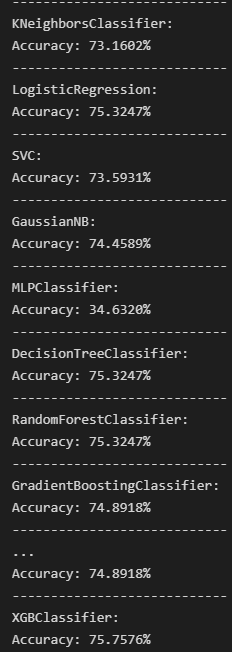
    dogruluk = accuracy\_score(y\_test, y\_pred)

    print("-"\*28)

    print(isimler + ":" )

    print("Accuracy: {:.4%}".format(dogruluk))

* Modellerin doğruluk oranları tahminlerini ele alacağız.



* Genel olarak bakıldığında tahmin oranlarında XGB değerinin en iyisi olduğu gözüküyor.
* Yapay sinir ağaçlarının %34 olmasına takılmayın gerçek değeri değildir. Genel bir işlem yaptığımız için bu şekilde geldi.

sonuc = []

sonuclar = pd.DataFrame(columns= ["Modeller","Accuracy"])

for model in modeller:

    isimler = model.\_\_class\_\_.\_\_name\_\_

    y\_pred = model.predict(X\_test)

    dogruluk = accuracy\_score(y\_test, y\_pred)

    sonuc = pd.DataFrame([[isimler, dogruluk\*100]], columns= ["Modeller","Accuracy"])

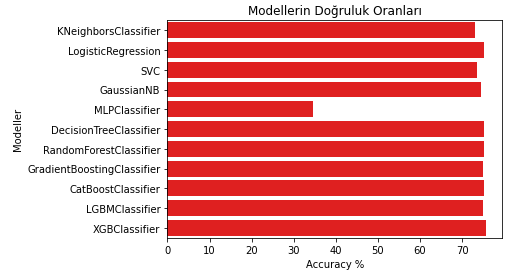
    sonuclar = sonuclar.append(sonuc)

sns.barplot(x= 'Accuracy', y = 'Modeller', data=sonuclar, color="r")

plt.xlabel('Accuracy %')

plt.title('Modellerin Doğruluk Oranları');

* Değerlerimizi Grafiğe dökelim.



* Modellerimiz büyük bir kısmının %70 in üzerinde değerler çıktığı yukarıdaki grafikle gözlemlenmektedir.