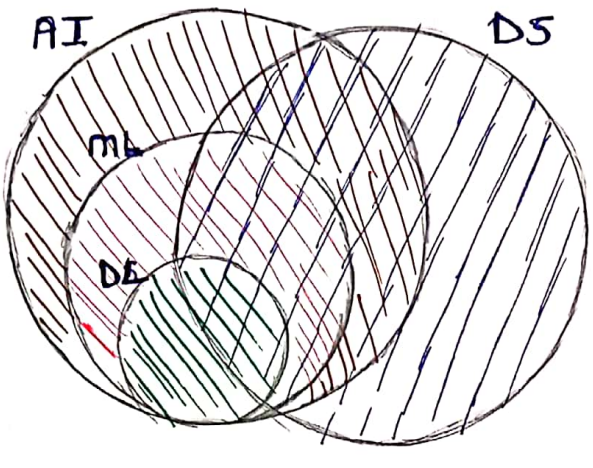
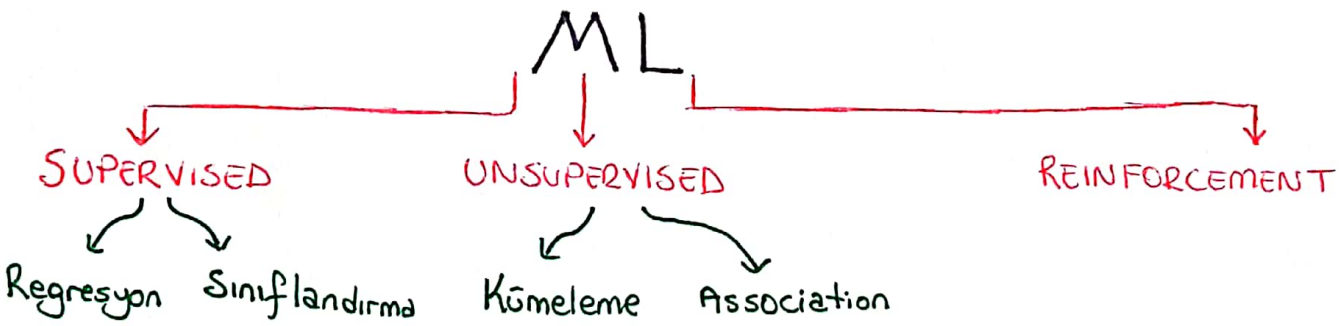
MAKİNE ÖĞRENMESİ



Görüleceği üzere AI ve DS kesişen bir kümededir.

MAKİNE ÖĞRENMESİ SINIFLANDIRMA

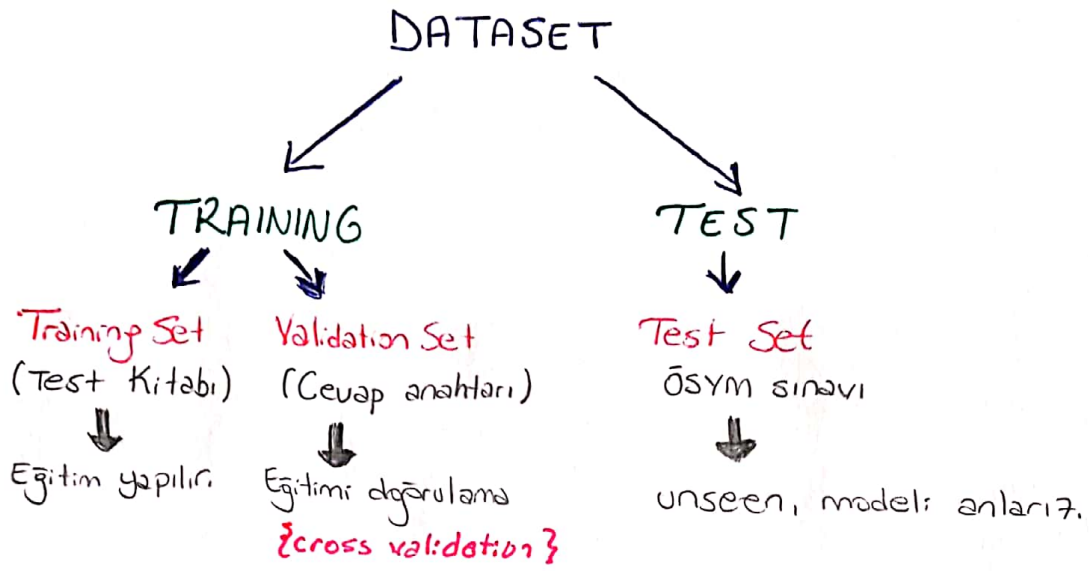


- Yukarıdaki kavramlar üzerinde duralım.
  - Supervised Learning = veri + label ⇒ Denetimli Öğrenme
  - Unsupervised Learning = veri + no label ⇒ Denetimsiz Öğrenme
  - Reinforcement Learning = Deneme - yanılma ⇒ Pekiştirmeli Öğrenme

  - Regresyon = Sürekli, sayısal sonuç verir. Ev fiyatı tahmini
  - Sınıflandırma = Sonuç kategoriktir. Spam - spam değil.

- Şimdi de Scikit-learn haritasını inceleyelim.

Bu işlemler yapılırken verimizi iyi şekilde bölmemiz gerekir.

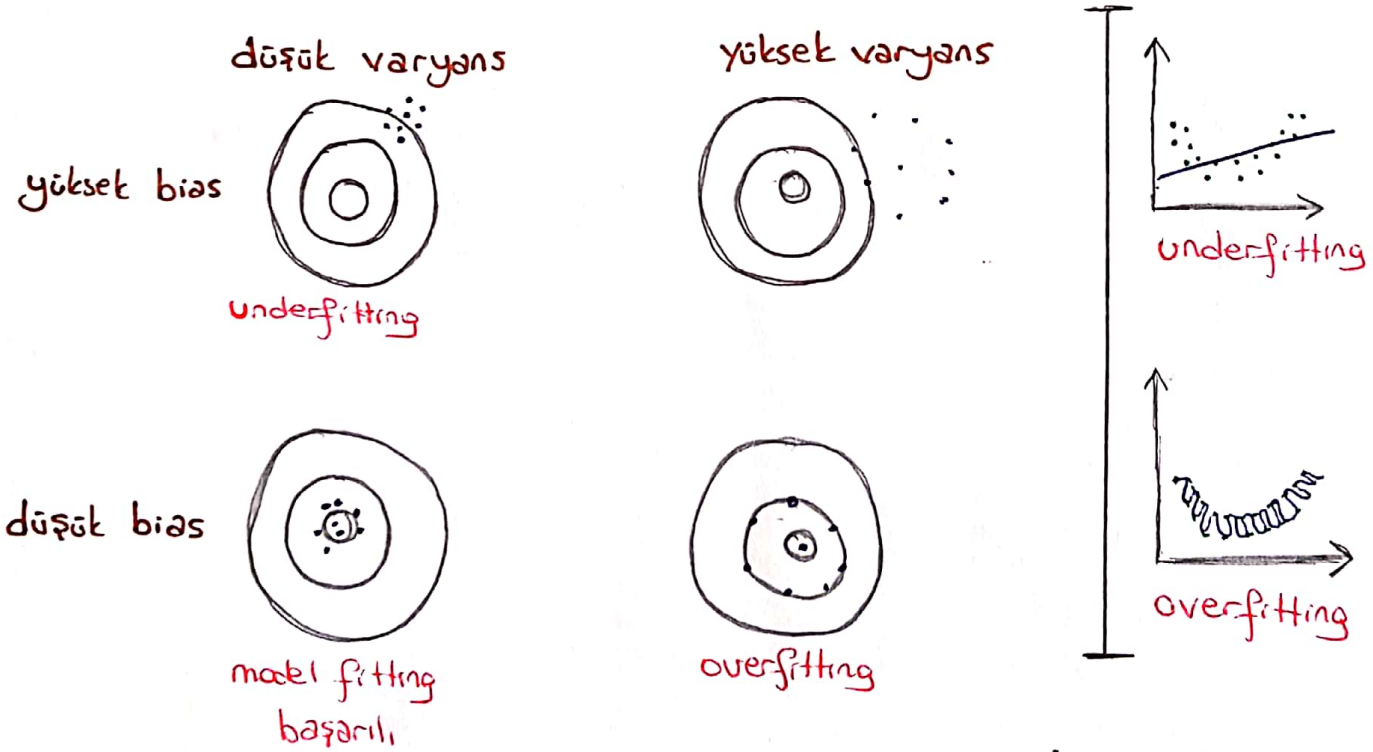


BIAS - VARYANS

bias = merkeze, doğruluğa uzaklık.

varyans = yayılımı verir.

- underfitting = yüksek bias düşük varyans = Hiç öğrenmeme = doğruluğa çok uzak.
- overfitting = yüksek varyans düşük bias = ezberleme olmuş, çok iyi dağılmış.



↓ bias ↓ varyans = başarı

REGRESYON

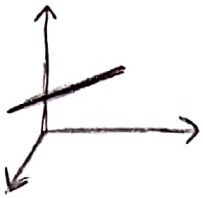
Basit Lineer Reg.

- 1 tane bağımsız değişken

$y = wx + b$

Coklu Lineer Reg.

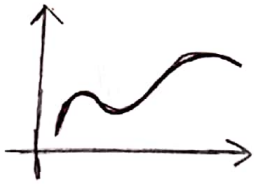
- Birden fazla bağımsız değişken



$y = b_0 + b_1x_1 + b_2x_2 \dots$

Polinomial Regression

- bağımsız derece n'inci dereceden.



$y = x^n \dots \quad n \leq 2$

① KOD GRAMERİ ⇒ BASIT VE COKLU

```
from sklearn.linear_model import LinearRegression → kütüphane ekledik
LR = LinearRegression()  ⇒ LR objesi oluştu
LR.fit(X_train, Y_train)  ⇒ eğitim
y_pred = LR.predict(X_test)  ⇒ tahmin
```

② KOD GRAMERİ ⇒ Polinomial

```
from sklearn.preprocessing import PolynomialFeatures
poly_reg = PolynomialFeatures(degree = n)
X_poly = poly_reg.fit_transform(X)
LR.fit(X_poly, y)
```

$y = wx + b$

- score = R2 score
- coef_ = w = ağırlık
- intercept_ = b

} $y = coef\_ * x + intercept\_$

REGRESYON HATA AZALTMA

① Early Stopping

- Gradient Descent gibi yinelemeli algoritmalarda düzenlilik gerekir. Doğrulama hatası **min** ulaşınca eğitim durur.
- callback

② L1 - L2 Regülarizasyon ⇒ overfitting önlenir.

- **L1 LASSO**

  Gradient Descent Denklemi + mutlak değer

  Mutlak değer ile cezalandırılır.

  0 olabilir.

$$\frac{\lambda}{m}|\theta_i|$$

```
from sklearn.linear_model import Lasso
model = Lasso(alpha=n)  ⇒ fit ve predict ⇒ metric
```

- **L2 RIDGE**

  Gradient Descent Denklemi + kareli değer

  Karesi ile cezalandırılır.

  0 asla olamaz.

$$\frac{\lambda}{m}\theta_i^2$$

```
from sklearn.linear_model import Ridge
model = Ridge(alpha=n)  ⇒ fit ve predict ⇒ metric
```

KAVRAMLAR :

α = Learning Rate = Gradient Descent çalışırken yakınsama hızı.

θ = theta = feature ağırlıdır.

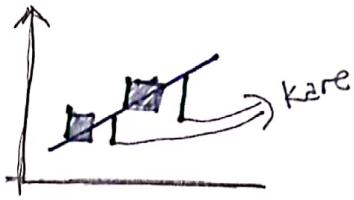
λ = Regülarizasyon term = Ağırlığın ne kadar cezalandıracağımızı belirler.

REGRESYON PERFORMANS ÖLÇÜMÜ {metrics}

Amaç: y ile y-pred arasındaki hatayı ölçmektir.

① R Squared

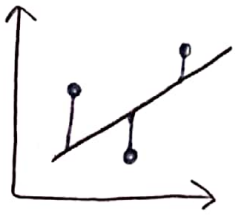
```
from sklearn.metrics import r2_score
r2_score(y, y-pred)
```



$$R^2 = 1 - \frac{SS_{reg}}{SS_{total}}$$

② Mean Absolute Error = MAE

```
from sklearn.metrics import mean_absolute_error
mean_absdute_error(y, y-pred)
```



- Aykırı değerden etkilenmez. ⇒ Avantaj
- Minimumda türevlenebilir değil ⇒ Dezavantaj = Gradient Descent olmaz.

$$MAE = \frac{1}{n} \cdot \Sigma |y - \hat{y}|$$ ⇒ farkların mutlak değerinin ortalaması

③ Mean Squared Error = MSE

```
from sklearn.metrics import mean_squared_error
mean_squared_error(y, y-pred)
```

- En sık kullanılan regresyon hata fonksiyonudur.
- Türevlenebilir. ⇒ Çünkü üssü var.

$$MSE = \frac{1}{n} \sum (y - \hat{y})^2$$

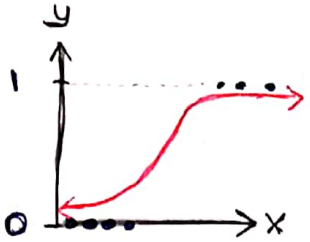
❗ y = y_test

CLASSIFICATION

- 2'li veya 2'den fazla siniflandirma olabilir.

① LOGISTIC REGRESYON

- Siniflandirma işlemi yapan regresyon modelidir.



spam / not spam
yes / no
} 0 ile 1 arasında olasılık değeri
threshold = eşik değeri

Lineer Regression ⇒ Logistic Regression

Activation Function = *sigmoid , ~~tanh~~ , ~~RELU~~ ....
↳ SOFT MAX = 2'den fazla sınıf.

```
from sklearn.linear_model import LogisticRegression
LR = LogisticRegression(*)
LR.fit(X_train, y_train)
LR.predict(X_test)
```

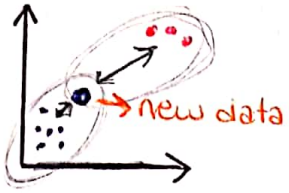
(*) → penalty
random_state
⋮

Kısaca özet geçecek olursak:

- Lineer fonksiyon + Sigmoid Function = Logistic Regresyon
- Eşik değeri önemlidir.
- S-shape

! sklearn.metrics → classification_report

② KNN = K-Nearest Neighbors



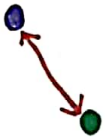
- Çalışma mantığı : en yakın komşuya yeni datanın eklenmesidir.

  k = bakılması istenen komşu sayısıdır.

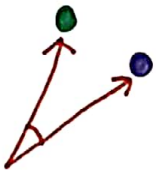
  k=5 olsun. ⇒ new datanın en yakın 5 komşusuna bakılır.

  3 mor, 2 kırmızı varsa ⇒ new data = mor olur.

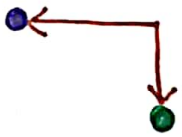
- Mesafe Ölçümleri



euclidean



cosine



manhattan . . . . .

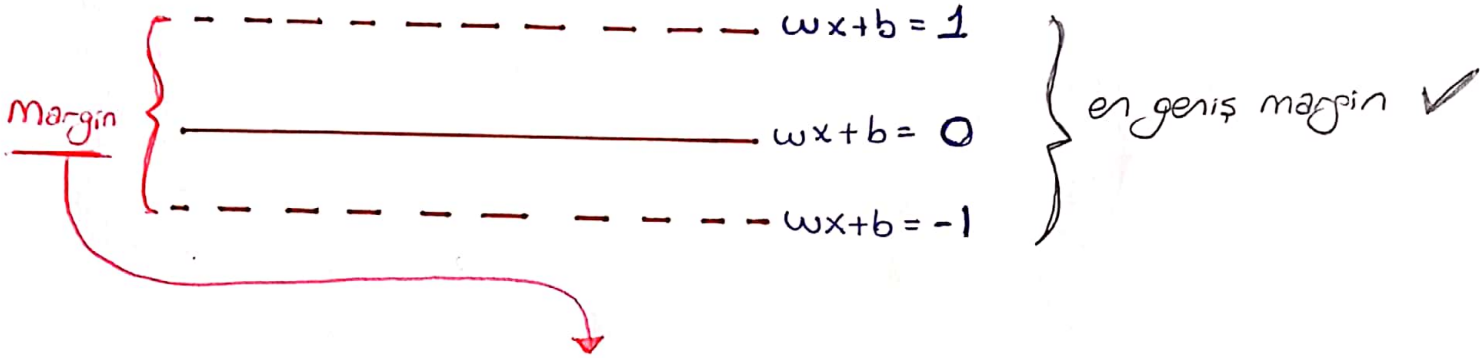
- En iyi k değeri = Decision Boundary


```
from sklearn.neighbors import KNeighborsClassifier
KNN = KNeighborsClassifier ( n_neighbors = n )
KNN.fit ( X_train , y_train )
KNN.predict ( X_test )
```

③ Support Vector Machine = SVM

SVM ; sınıflandırma , regression , aykırı değer sonucu üretir.

- Küçük ve orta ölçekli veriler için çok uygundur.

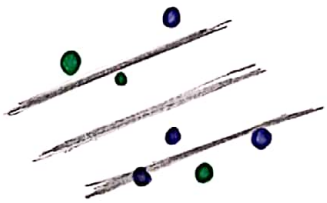


MARGIN

HARD MARGIN

- margin içi boş

SOFT MARGIN



- margininde olabilir. Yanlış değerler olabilir. → overfitting engellenir.

```
from sklearn.svm import SVC
svm_model = SVC(kernel=...)        → rbf
svm_model.fit(X_train, y_train)
svm_model.predict(X_test)
```

Kernel Library

- Gausian
- Polynomial
- RBF

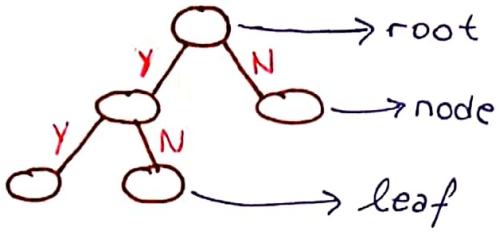
↓

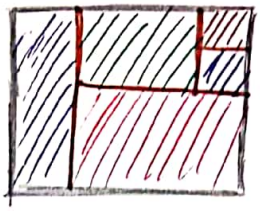
- Kernel Trick
- Boyut artırmak

- sklearn.kernel_approximation
  - ⇓
  - Nystroem
  - RBFsampler

boyut ↑

④ Decision Tree





⇒ overfitting
↳ yaygın

ID3

- pure = → tüm veriler aynı sınıfa aittir.
- impure = → tüm veriler aynı sınıfa ait değildir.

- gini impurity = verinin ne sıklıkla yanlış tanımlandığını anlarız.
  ↳ GINI INDEX

! Gini Index ↓ ⇒ Tercih Edilme ↑

ENTROPY = Düzensizlik

Entropi:	low	medium	high
Bilgi :	high	medium	low

! graphviz

! pruning ⇒ optimize

INFORMATION GAIN

- 0 kötü - 1 iyi
- IG = entropy(parent) − n [entropy(c1) + ... entropy(cn)] (n → dal sayısı)

! Gini = [0, 0.5]
Entropy = [0, 1]

```
from sklearn.tree import DecisionTreeClassifier
DTC = DecisionTreeClassifier( )
DTC.fit (X_train, y_train) ⇒ DTC.predict (X_test)
```

} aynı

ENSEMBLE LEARNING

- Birden fazla algoritma çalışması ile sonuç verir.
  - Maximum oylama
  - Ortalama
  - Ağırlıklı ortalama
  - Yığma {Stacking}
  - Blending

BAGGING

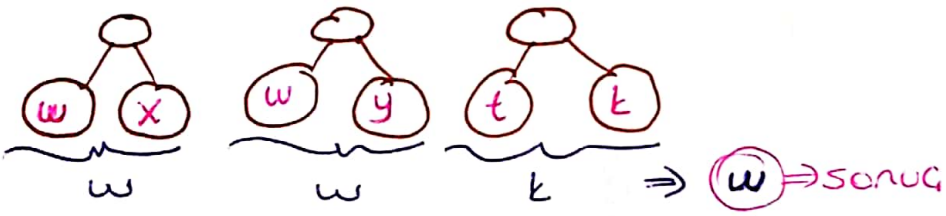
- Random Forest

BOOSTING

- Ada Boost
- Gradient Boosting
- XG Boost

⑤ Random Forest

- Decision tree'deki overfitting sorunu için alternatiftir.
- Kolonlar rastgele birleşir ve birden çok ağaç oluşur.
  - ↳ Sonucunda en iyisi seçilir.

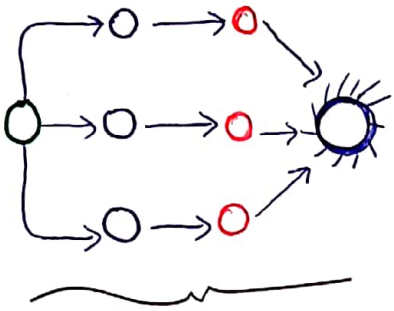



```
from sklearn.ensemble import RandomForestClassifier
RF = RandomForestClassifier ( )
RF.fit (X_train, y_train )
RF. predict (X_test )
```

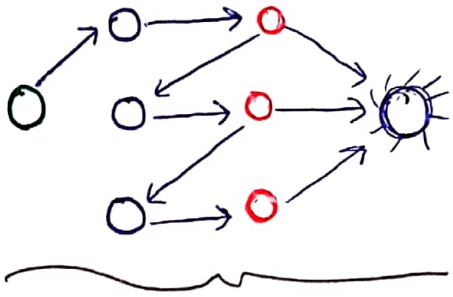
BAGGING - BOOSTING

O ⇒ weak learner

⇒ strong learner



paralel



sequential

Her algoritma ayrı ayrı çalışır ve sonunda birleştirilir.

Her algoritmanın en iyi olduğu kısım göz önüne alınır. Sıralı olarak birleştirilir.

⑥ Adaboost

doğruluk: (1) (0) (-1) } modelin ağırlığına göre birleştirme yapılır.

weight: LARGE (+) medium Large (−)

(+) + (−) = (+) olur.

(+) ↙ weight↑ (−) ↘ weight↓

base_estimator = weak learner ⇒ DecisionTree ... vs.

n_estimator = kaç tane learner olacak?

Kod grameri aynıdır. DecisionTree vs. import etmeyi unutma.

⑦ Gradient Boosting

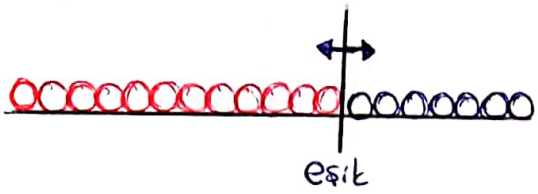
⑧* XGBoost

⑨ Naive Bayes

SINIFLANDIRMA PERFORMANS ÖLÇÜMÜ {metrics}

- Bu bölümde ölçüm ve hata ataltma üzerine duracağız.

KAVRAM **Thresholding** = Eşik değerimiz değiştirilirse performans değişir.



KAVRAM **TRUE - FALSE & Positive - Negative**

- Ölçüm yapabilmek için TN, TP, FN, FP gereklidir.

Positive = YES
Negative = NO
} olsun. ⇒

NO tahmin ettik.
- ↙ DOĞRU ise → TN
- ↘ Yanlış ise → FN

YES tahmin ettik.
- ↙ Doğru ise → TP
- ↘ Yanlış ise → FP

① CONFUSION MATRIX

$$\begin{bmatrix} TP & FP \\ FN & TN \end{bmatrix}$$

```
from sklearn.metrics import confusion_matrix
confusion_matrix ( y_test , y_pred )
```

② Accuracy

- Doğru tahmin edilenlerin tüm tahminlere bölümü ile bulunur.

```
from sklearn.metrics import accuracy_score
accuracy_score ( y_test , y_pred )
```

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

↑ = yanlışlar yakalanmıyor. Yani kötü

③ Precision & Recall

Precision = "Pozitif tahminlerin ne kadarı doğru?" cevabını verir.

Recall = "Gerçekte pozitif olanların kaçı doğru?" cevabını verir.

④ F1 SCORE = F_β Score

- Precision ve Recall arasında seçim yapmak zordur.

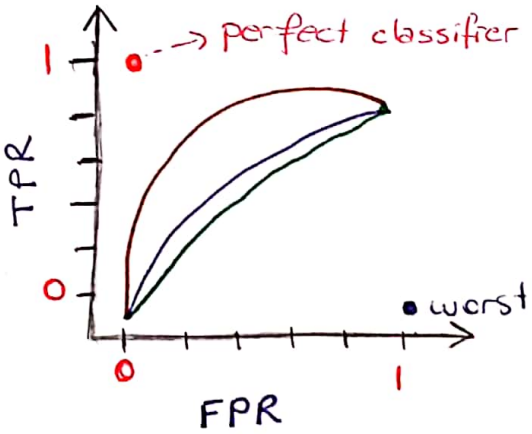
F1 Score = Precision ve Recall harmonik ortalamasıdır.

↳ Çünkü eşit katkı ile doğru sonuç.

⑤ ROC - AUC

ROC = Her threshold adımı için metrikleri gösterir.

$$TPR = \frac{TP}{TP + FN}$$

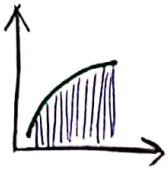


kahverengi > mor > yeşil

↓

daha iyi sınıflandırma ● 'a yakın.

AUC = ROC eğrisinin integrali {altında kalan alan}



- sklearn.metrics'den kod kısmını yazabilirsin.
- Aşağıda formüllere göz gezdirebilirsin.

PYTHON LISTS [ ]

İçerisinde farklı veri tipleri barındırır

fam = [1.73 , "doğukan" , True , False , "mom" , 32 , "sevval"]

fam2 = [["doğu" , 174] , ["arif" , 172]] } liste içinde liste de olabilir

type (fam) type (fam2) → list

∝ Erişim

fam [0] = 1.73 fam [-1] = ~~True~~ "sevval"

genişlettim.

∝ Slicing

fam [x : y]

x: başlangıç dahil

y: bitiş index dahil değil

} fam [:2] = 2. indexe kadar olan
fam [1 :] = 1 dahil ve sonrası

∝ Manipulating Lists

• Changing list elements

- fam [4] = 1.86 ⇒ 4. indexteki değer "mom" → 1.86 olur.
- fam [0:2] = ["lisa , 1.74] ⇒ 0 ve 1. index değeri değişir.

• Add list elements

fam = fam + ["me" , 1.79] ⇒ fam'in sonuna iki değer ekledik.

• Remove list elements

del (fam [2]) ⇒ True değeri silinir.

max (fam)

fam. index ("mom")

fam. count (1.73)

fam. append (x)

list → NUMPY

np_array = np. array (list)

DICTIONARIES { }

- Listeleri incelemiştik. Eğer listeler arasında bağlantı kuruyorsak dictionaries bizim için bir çözümdür.

pop = [30 , 40 , 50]
country = ['TR' , 'FR' , 'BE'] } 2 bağıntılı liste

world = { 30 : 'TR' , 40 : 'FR' , 50 : 'BE' } - dictionary
key ↙ ↘ value

işlemler için = [ ]

[world [key] = value]

∝ YENİ DEĞER EKLEME

world [60] = 'PL' ⇒ sözlüğe yeni bir değer ekledik.

∝ DEĞER SİLME

del (world [key])

world.keys() — world.values()

pd.DataFrame (dict)

MATPLOTLIB

```
import matplotlib.pyplot as plt

year = [1950 , 1970 , 1990 , 2010]
pop = [2 ,   3   ,  5  ,  7 ]

plt.plot (year , pop) = line plot
plt.scatter (year , pop) = scatter plot
plt.hist (year , bins=3) = hist plot

plt.xlabel ('Year')
plt.ylabel ('Pop')
plt.title ('Title')
plt.yticks ([0,2,4,6,8,10])
plt.yticks ([0,2,4,6,8,10] , ['0B','2B','4B','6B','8B','10B'])
plt.xscale ('log')
plt.grid (True) #
```

import NUMPY as np

- Listeleri daha iyi ve matematiksel olarak manipole etmemize yarar.

```
list1 = [1.73 , 1.86 , 1.55 , 1.90 , 2.00]
list2 = [60 , 70 , 80.2 , 95 , 105.4]
```

list1 / list2**2 ⇒ ERROR ✗

SOLUTION

```
np_list1 = np.array(list1)
np.list2 = np.array(list2)
```

} listeleri → numpy array yaptık.

np_list1 / np_list2**2 ⇒ SONUÇ ✓

- list1 + list2 ⇒ [1.73 , 1.86 ---- 2.00 , 60 , 70 --- 105.4] ⇒ birleştirir
- np_list1 + np_list2 ⇒ ([61.73 , 71,86 .... 107.4]) ⇒ toplar

numpy subsetting

np_list > 1 ⇒ [True , False, True, True..]

np_list[np_list>1] ⇒ True ise değeri verir ⇒ filtreleme

2D numpy array

```
np.array ( [[1.73, 1.68 , 1.71 , 1.89 , 1.79],
            [65 , 59 , 63 , 88 , 68 ]] )
```

} liste

x[row][col] = x[row, col] ⇒ x[1 , 2] = 63

x[: , :] ⇒ tüm satır ve kolonu getir ⇒ filtreleme ⇒ ayrıntılı var. ↓

---

np.mean (np_city[:,0]) , median , std , sum , sort , round

np.corrcoef (np1, np2) x.shape logical_and() - or - not

PANDAS

```
import pandas as pd
brics = pd.DataFrame(dict)
brics = pd.read_csv("...")
```
} DataFrame oluşturduk.

Filtreleme - ACCESS

column:
- `brics["column"]` = pandas.Series
- `brics[["column"]]` = pandas.DataFrame
- `brics[["col1","col2"]]` ⇒ 2 kolonlu df olabilir.

row:
- `brics[1:4]` ⇒ 1,2,3. satırı verir.

- **loc** ⇒ Kategorik verilerde işlem yapılır.
  - `brics.loc[["RU"]]` = RU satırını verir.
  - `brics.loc[["RU","AL"]]` = RU ve AL satırını verir.
  - `brics.loc[["RU","AL"],["col1","col2"]]` = row-col loc
  - `brics.loc[["RU","AL"],:]` = tüm kolonlar, ru-al satırı

- **iloc** ⇒ integer {index} ile erişim yapılır.

- **karşılaştırmalı filtreleme**
  - `brics[brics["col"] > 8]` ⇒ !np.logical kullan.!

loop ⇒ datacamp / Intermediate Python / Loops
random ⇒ datacamp / Intermediate Python / Case Study