WEKA

Ganancia - Select Attributes / InfoGain

PYTHON

Import Comunes ----------------------------------------------------------------

+ import pandas as pd

+ import numpy as np

Regresión ---------------------------------------------------------------------

Comun

+ from sklearn.preprocessing import MinMaxScaler

+ from sklearn.preprocessing import LabelBinarizer

- Read\_csv(X) # Leer el csv

- X = MinMaxScaler().fit\_transform(X) # Normalizar los atributos (0,1)

- YB = LabelBinarizer().fit\_transform(Y) # Binariza la clase (o clases)

- ### Hold Out o KFold ###

Hold Out

+ from sklearn.model\_selection import train\_test\_split

- X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, YB, test\_size=0.33, stratify=YB) # Hold Out 2/3

- ### Regresion ###

KFold

+ from sklearn.model\_selection import StratifiedKFold

- cv = StratifiedKFold(n\_split=10) # KFold

- accuracy = 0

- for train\_index, test\_index in cv.split(X,Y):

- X\_train, X\_test = X[train\_index], X[test\_index] # Sacar las X para el train y para el test

- Y\_train, Y\_test = Y[train\_index], Y[test\_index] # Sacar las Y para el train y para el test

- ### Regresion ###

- accuracy += np.sum(np.argmax(Y\_predict, axis=1) == np.argmax(Y\_test, axis=1)) / Y\_test.shape[0] # Tasa de Error en cada parte

- accuracy = accuracy / 10 # Tasa de Error total

Regresión Lineal

+ from sklearn.linear\_model import LinearRegression

- model = LinearRegression() # Regresion Linear

- model.fit(X\_train, Y\_train) # Entrenar

- Y\_predict = model.predict(X\_test) # Prediccion

Regresión Múltiple

+ from sklearn.linear\_model import LinearRegression

- Y\_predict = np.zeros(Y\_test.shape) # Matriz de ceros

- for i in range (Y\_train.shape[1]):

- model = LinearRegression() # Regresion Linear (Multiple)

- model.fit(X\_train, Y\_train[:,i]) # Entrenar

- Y\_predict[:,i] = model.predict(X\_test) # Prediccion

- np.sum(np.argmax(Y\_predict, axis=1) == np.argmax(Y\_test, axis=1)) / Y\_test.shape[0] # Tasa de Error

Regresión Logística

+ from sklearn.linear\_model import LogisticRegression

- clf = LogisticRegression(n\_jobs=-1) # Regresion Logistica

- clf.fit(X\_train, Y\_train) # Entrenar

- clf.score(X\_test, Y\_test) # Tasa de Error

- Y\_predict = clf.predict(X\_test) # Prediccion

Perceptron --------------------------------------------------------------------

Crear Puntos

+ from sklearn.datasets import make\_blobs

- X, y = make\_blobs(n\_samples=[1000,1000], n\_features=2) # Crear puntos

Mostrar Todo

+ from sklearn.datasets import make\_blobs

- plt.scatter(X[:,0], X[:,1], c=y) # Algo raro

- plt.show() # Mostrar

Comun

+ import matplot.pyplot as plt

+ from sklearn.preprocessing import MinMaxScaler

- ### Crear Puntos ###

- X = MinMaxScaler().fit\_transform(X) # Normalizar los atributos (0,1)

- YY = np.where(YB==0, -1, YB) # Cambiar 0 por -1

- ### Perceptron ###

- ### Mostrar ###

Perceptron

+ from sklearn.linear\_model import Perceptron

- ps = Perceptron(max\_iter=10000) # Perceptron

- ps.fit(X, YY) # Entrenar

- ps.predict(X) # Prediccion

Mostrar

- xt = np.linspace(0,1) # No lo se

- w1 = ps.coef\_[0][1] # Peso #1

- w2 = ps.coef\_[0][1] # Peso #2

- plt.plot(xt, (w1\*xt\*ps.intercept\_)/(-w2), 'g', linewidth=2) # Linea

- ### Mostrar Todo ###

Perceptron Multiclase ---------------------------------------------------------

Crear Puntos

+ from sklearn.datasets import make\_blobs

- X, y = make\_blobs(n\_samples=40, n\_features=2, centers=4) # Crear puntos

Mostrar Todo

+ from sklearn.datasets import make\_blobs

- plt.scatter(X[:,0], X[:,1], c=y) # Algo raro

- plt.show() # Mostrar

Comun

+ import matplotlib.pyplot as plt

+ from sklearn.preprocessing import MinMaxScaler

+ from sklearn.preprocessing import LabelBinarizer

- ### Crear Puntos ###

- X = MinMaxScaler().fit\_transform(X) # Normalizar los atributos (0,1)

- YB = LabelBinarizer().fit\_transform(y) # Binarizar Y

- YY = np.where(YB==0, -1, YB) # Cambiar 0 por -1

- ### Multiclase ###

Multiclase

+ from sklearn.linear\_model import Perceptron

- ps = Perceptron(max\_iter=10000) # Perceptron

- for i in range(4):

- ps.fit(X, YY[;,i]) # Entrenar

- ps.predict(X) # Prediccion

- ### Mostrar ###

- ### Mostrar Todo ###

Mostrar

- xt = np.linspace(0,1) # No lo se

- w1 = ps.coef\_[0][1] # Peso #1

- w2 = ps.coef\_[0][1] # Peso #2

- plt.plot(xt, (w1\*xt+ps.intercept\_)/(-w2), 'g', linewidth=2) # Linea

Adaline -----------------------------------------------------------------------

Cargar Datos

+ from sklearn.datasets import load\_algo

- Algo = load\_algo() # Cargar datos

- X = Algo.data # Cargar las X

- Y = Algo.target # Cargar la Y

Comun

+ form sklearn.preprocessing import StandardScaler

- ### Cargar Datos ###

- X = StandardScaler().fit\_transform(X) # Estandarizar

- ### KFold ###

- ### Media y Desviacion ###

KFold

+ from sklearn.model\_selection import KFold

- cv = KFold(n\_split=10) # KFold

- Mat = ps.DataFrame(columns=['Aciertos','R2']) # Matriz para guardar los resultados

- i = 0

- for train\_index, test\_index in cv.split(X,Y):

- X\_train, X\_test = X[train\_index], X[test\_index] # Sacar las X para el train y para el test

- Y\_train, Y\_test = Y[train\_index], Y[test\_index] # Sacar las Y para el train y para el test

- ### SGDRegressor ###

- acierto = np.sum(np.abs(np.divide(Y\_predict, Y\_test) -1) < 0.15) / Y\_test.shape[0] # Como de parecidos son

- Mat.loc[i,:] = [acierto, r2]

- i = i+1

SGDRegressor

+ from sklearn.linear\_model import SGDRegressor

- model = SGDRegressor(tol=ie-6, max\_iter=5000) # SGDRegressor

- model.fit(X\_train, Y\_train) # Entrenar

- Y\_predict = model.predict(X\_test) # Prediccion

- r2 = model.score(X\_test, Y\_test) # Tasa de Error

Media y Desviacion

- Mat.mean() # Media

- Mat.std() # Desviacion estandar

Perceptron Multicapa ----------------------------------------------------------

Cargar Datos

+ from sklearn.datasets import fetch\_openml as fetch\_mldata

- db\_name = 'Algo'

- data\_set = fetch\_mldata(db\_name) # Cargar los datos

- X = data\_set.data # Cargar las X

- Y = data\_set.target # Cargar la Y

Comun

+ form sklearn.preprocessing import StandardScaler

- ### Cargar Datos ###

- X = StandardScaler().fit\_transform(X) # Estandarizar

- ### KFold ###

KFold

+ from sklearn.model\_selection import StratifiedKFold

- cv = StratifiedKFold(n\_split=10) # KFold

- for train\_index, test\_index in cv.split(X,Y):

- X\_train, X\_test = X[train\_index], X[test\_index] # Sacar las X para el train y para el test

- Y\_train, Y\_test = Y[train\_index], Y[test\_index] # Sacar las Y para el train y para el test

- ### MLPClassifier ###

- accuracy = np.sum(Y\_predict == Y\_test) / Y\_test.shape[0] # Tasa de Error cada parte

MLPClassifier

+ from sklearn.neural\_network import MLPClassifier

- model = MLPClassifier(hidden\_layer\_sizes=(val), max\_iter=2000) # MLPClassifier

- model.fit(X\_train, Y\_train) # Entrenar

- Y\_predict = model.predict(X\_test) # Prediccion