**1**

# import libraries

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.metrics import r2\_score, mean\_squared\_error

from math import sqrt

# this allows plots to appear directly in the notebook

%matplotlib inline

# read data into a DataFrame

data = pd.read\_csv('Advertising.csv', index\_col=0)

data.head()

data.columns = ['TV','Sales']

# print the shape of the DataFrame

data.shape

# visualize the relationship between the features and the response using scatterplots #data.plot(kind='scatter', x='TV', y='Sales')

plt.scatter(data['TV'], data['Sales'])

# create X and y

#taking only one variable for now

X = data[['TV']]

X

y = data.Sales

y

# follow the usual sklearn pattern: import, instantiate, fit

from sklearn.linear\_model import LinearRegression

lm = LinearRegression()

lm.fit(X, y)

# print intercept and coefficients

print(lm.intercept\_)

print(lm.coef\_)

# manually calculate the prediction using above slope and intercept in b0+b1\*x 7.032594 + 0.047537\*50

# you have to create a DataFrame since the Statsmodels formula interface expects it X\_new = pd.DataFrame({'TV': [230.1]})

X\_new.head()

# use the model to make predictions on a new value

lm.predict(X\_new)

data['TV'].min()

# create a DataFrame with the minimum and maximum values of TV X\_new = pd.DataFrame({'TV': [data['TV'].min(), data['TV'].max()]}) X\_new.head()

# make predictions for those x values and store them

preds = lm.predict(X\_new)

preds

# first, plot the observed data

data.plot(kind='scatter', x='TV', y='Sales')

# then, plot the least squares line

plt.plot(X\_new, preds, c='red', linewidth=2)

predictions = lm.predict(X)

print(sqrt(mean\_squared\_error(y, predictions)))

r2 = r2\_score(y, predictions)

r2

**2**

import numpy

from sklearn import linear\_model

X = numpy.array([3.78, 2.44, 2.09, 0.14, 1.72, 1.65, 4.92, 4.37, 4.96, 4.52, 3.69, 5.88]).reshape(-1,1)

y = numpy.array([0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1])

logr = linear\_model.LogisticRegression()

logr.fit(X,y)

def logit2prob(logr, X):

log\_odds = logr.coef\_ \* X + logr.intercept\_

odds = numpy.exp(log\_odds)

probability = odds / (1 + odds)

return(probability)

print(logit2prob(logr, X))

**3**

import numpy as np

import matplotlib.pyplot as plot

%matplotlib inline

import sklearn

from sklearn.datasets import load\_digits

digits = load\_digits()

X = digits.data

y = digits.target

print("Shape of X is {}".format(X.shape))

print("Shape of y is {}".format(y.shape))

X[0]

def plot\_digit(x,index):

plot.imshow(x.reshape(8,8))

print(index)

plot\_digit(X[104],y[104])

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=1) print(X\_train.shape)

from sklearn.linear\_model import LogisticRegression

lr = LogisticRegression()

lr.fit(X\_train,y\_train)

y\_predict1 = lr.predict(X\_train)

from sklearn.metrics import accuracy\_score

accuracy = accuracy\_score(y\_predict1,y\_train)

print(accuracy)

y\_predict = lr.predict(X\_test)

accuracy = accuracy\_score(y\_predict,y\_test)

print(accuracy)

lr.predict([X[100], X[152]])

**4**

import pandas

from sklearn import linear\_model

df = pandas.read\_csv("cars.csv")

X = df[['Weight', 'Volume']].values

y = df['CO2'].values

regr = linear\_model.LinearRegression()

regr.fit(X, y)

#predict the CO2 emission of a car where the weight is 1150kg, and the volume is 1600cm3: predictedCO2=regr.predict([[1150, 1600]])

print(predictedCO2)

#predict the CO2 emission of a car where the weight is 2300kg, and the volume is 1300cm3: predictedCO2 = regr.predict([[2300, 1300]])

print(predictedCO2)

X

print(regr.intercept\_)

print(regr.coef\_)

**5**

# Basic packages

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

# Sklearn modules & classes

from sklearn.linear\_model import Perceptron, LogisticRegression

from sklearn.svm import SVC

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

from sklearn.preprocessing import StandardScaler

from sklearn import datasets

from sklearn import metrics

# Load the data set; In this example, the breast cancer dataset is loaded. bc = datasets.load\_breast\_cancer()

X = bc.data

y = bc.target

print(X.shape)

print(y.shape)

# Create training and test split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=1, stratify=y)

sc = StandardScaler()

sc.fit(X\_train)

X\_train\_std = sc.transform(X\_train)

X\_test\_std = sc.transform(X\_test)

# Instantiate the Support Vector Classifier (SVC)

svc = SVC(C=1.0, random\_state=1, kernel='linear')

# Fit the model

svc.fit(X\_train\_std, y\_train)

# Make the predictions

y\_predict = svc.predict(X\_test\_std)

# Measure the performance

print("Accuracy score %.3f" %metrics.accuracy\_score(y\_test, y\_predict))

**6**

# -\*- coding: utf-8 -\*-

"""

Created on Thu Mar 9 16:46:38 2023

@author: tiver

"""

def hebbian\_learning(samples):

print(f'{"INPUT":^8} {"TARGET":^16}{"WEIGHT CHANGES":^15}{"WEIGHTS":^25}') w1, w2, b = 1, 1, 1

print(' ' \* 45, f'({w1:2}, {w2:2}, {b:2})')

for x1, x2, y in samples:

w1 = w1 + x1 \* y

w2 = w2 + x2 \* y

b = b + y

print(f'({x1:2}, {x2:2}) {y:2} ({x1\*y:2}, {x2\*y:2}, {y:2}) ({w1:2}, {w2:2}, {b:2})') AND\_samples = {

'binary\_input\_binary\_output': [ [1, 1, 1],

[1, 0, 0],

[0, 1, 0],

[0, 0, 0]

],

'binary\_input\_bipolar\_output': [ [1, 1, 1],

[1, 0, -1],

[0, 1, -1],

[0, 0, -1]

],

'bipolar\_input\_bipolar\_output': [ [ 1, 1, 1],

[ 1, -1, -1],

[-1, 1, -1],

[-1, -1, -1]

]

}

OR\_samples = {

'binary\_input\_binary\_output': [ [1, 1, 1],

[1, 0, 1],

[0, 1, 1],

[0, 0, 0]

],

'binary\_input\_bipolar\_output': [ [1, 1, 1],

[1, 0, 1],

[0, 1, 1],

[0, 0, -1]

],

'bipolar\_input\_bipolar\_output': [ [ 1, 1, 1],

[ 1, -1, 1],

[-1, 1, 1],

[-1, -1, -1]

]

}

XOR\_samples = {

'binary\_input\_binary\_output': [ [1, 1, 0],

[1, 0, 1],

[0, 1, 1],

[0, 0, 0]

],

'binary\_input\_bipolar\_output': [

[1, 1, -1],

[1, 0, 1],

[0, 1, 1],

[0, 0, -1]

],

'bipolar\_input\_bipolar\_output': [

[ 1, 1, -1],

[ 1, -1, 1],

[-1, 1, 1],

[-1, -1, -1]

]

}

#For AND gate

print('-'\*20, 'HEBBIAN LEARNING', '-'\*20)

print('AND with Binary Input and Binary Output')

hebbian\_learning(AND\_samples['binary\_input\_binary\_output']) print('AND with Binary Input and Bipolar Output')

hebbian\_learning(AND\_samples['binary\_input\_bipolar\_output']) print('AND with Bipolar Input and Bipolar Output')

hebbian\_learning(AND\_samples['bipolar\_input\_bipolar\_output'])

# #OR Gate

# print('-'\*20, 'HEBBIAN LEARNING', '-'\*20)

# print('OR with binary input and binary output')

# hebbian\_learning(OR\_samples['binary\_input\_binary\_output']) # print('OR with binary input and bipolar output')

# hebbian\_learning(OR\_samples['binary\_input\_bipolar\_output']) # print('OR with bipolar input and bipolar output')

# hebbian\_learning(OR\_samples['bipolar\_input\_bipolar\_output'])

# #XOR Gate

# print('-'\*20, 'HEBBIAN LEARNING', '-'\*20)

# print('XOR with binary input and binary output')

# hebbian\_learning(XOR\_samples['binary\_input\_binary\_output']) # print('XOR with binary input and bipolar output') # hebbian\_learning(XOR\_samples['binary\_input\_bipolar\_output']) # print('XOR with bipolar input and bipolar output') # hebbian\_learning(XOR\_samples['bipolar\_input\_bipolar\_output'])

**7**

# -\*- coding: utf-8 -\*-

"""

Created on Thu Mar 16 13:18:59 2023

@author: tiver

"""

import numpy as np

# np.random.seed(seed=2)

I = np.random.choice([0,1], 3)# generate random vector I, sampling from {0,1} # W = np.random.choice([-1,1], 3) # generate random vector W, sampling from {-1,1} W = np.array([1,1,1])

print(f'Input vector:{I}, Weight vector:{W}')

dot = I @ W

print(f'Dot product: {dot}')

def linear\_threshold\_gate(dot: int, T: float) -> int:

'''Returns the binary threshold output'''

if dot >= T:

return 1

else:

return 0

T = 3

activation = linear\_threshold\_gate(dot, T)

print(f'When Threshold =3, Activation: {activation}')

**8**

# -\*- coding: utf-8 -\*-

"""

Created on Thu Mar 23 12:29:55 2023

@author: tiver

"""

import numpy as np

class Perceptron:

def \_\_init\_\_(self, input\_size, lr=1, epochs=100): self.W = np.zeros(input\_size+1)

self.epochs = epochs

self.lr = lr

def activation\_fn(self, x):

return 1 if x >= 0 else 0

def predict(self, x):

z = self.W.T.dot(x)

a = self.activation\_fn(z)

return a

def fit(self, X, d):

for epoch in range(self.epochs):

for i in range(d.shape[0]):

x = np.insert(X[i], 2, 1)

y = self.predict(x)

e = d[i] - y

self.W = self.W + self.lr \* e \* x

#For AND Gate

X1 = np.array([[0, 0], [0, 1], [1, 0], [1, 1]]) d1 = np.array([0, 0, 0, 1])

#For OR Gate

X2 = np.array([[0, 0], [0, 1], [1, 0], [1, 1]]) d2 = np.array([0, 1, 1, 1])

perceptron1 = Perceptron(input\_size=2) perceptron1.fit(X1, d1)

perceptron2 = Perceptron(input\_size=2) perceptron2.fit(X2, d2)

print(perceptron1.W)

# Output: [ 2. 1. -3.]

print(perceptron2.W)

# Output: [ 1. 1. -1.]

test\_in=np.array([0, 1, 1])

AND\_prediction=perceptron1.predict(test\_in) print(AND\_prediction)

OR\_prediction=perceptron2.predict(test\_in)

print(OR\_prediction)

**9**

# -\*- coding: utf-8 -\*-

"""

Created on Wed Apr 5 14:24:40 2023

@author: tiver

"""

# the code for importing and splitting the dataset

import sklearn

from sklearn.datasets import load\_digits

digits = load\_digits()

X = digits.data

y = digits.target

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=1)

print(X\_train.shape)

# Transforming the train and test sets such that they explain 95% of variance from sklearn.decomposition import PCA

sklearn\_pca = PCA(n\_components=0.95)

sklearn\_pca.fit(X\_train)

X\_train\_transformed = sklearn\_pca.transform(X\_train)

print(X\_train\_transformed.shape)

print(X\_test.shape)

X\_test\_transformed =sklearn\_pca.transform(X\_test)

print(X\_test\_transformed.shape)