**Practical 1: Implement Linear Regression (Diabetes Dataset)**

**Code:**

import pandas as pd #for data manipulation.

import numpy as np #for scientific computing.

import sklearn #for machine learning

import seaborn as sns #visualization package

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

import matplotlib.pyplot as plt #plotting & visualization

#importing the inbuilt dataset in sklearn package for performing regression analysis.

from sklearn import datasets

diabetes = datasets.load\_diabetes()

#print(diabetes.DESCR)

diabetes.feature\_names #checking the feature names

diabetes.data.shape #checking the shape of data

diabetes.target.shape

db\_df = pd.DataFrame(diabetes.data,columns=diabetes.feature\_names)

print(“Sample:\n”,db\_df.sample(5)) #checking a sample of the dataframe

db\_df['Progression'] = diabetes.target #new column name 'Progression'

print(“Sample with Target :\n”,db\_df.sample(2)) #checking the dataset once again.

print(“Null Value Check:\n”,db\_df.isna().sum())#check null value

print(“Summary Statistics:\n”,db\_df.describe()) #the below is the summary statistics of the dat

print(“Information:\n”,db\_df.info()) #Getting the information about the dataframe, the datatypes e

corr = db\_df.corr()

print(“correlation”,corr)

plt.subplots(figsize=(8,8))

sns.heatmap(corr,cmap= 'RdYlGn',annot=True)

plt.show()

x = db\_df.drop(labels='Progression', axis=1) #axis=1 means we

y = db\_df['Progression']

#splitting the dataset into 75%-25% train-test split

train\_x, test\_x, train\_y, test\_y = train\_test\_split(x,y,test\_size=0.25,random\_state=999)

print(“train\_x:”,train\_x.shape)

print(“test\_x:”,test\_x.shape)

print(“train\_y:”,train\_y.shape)

print(“test\_y:”,test\_y.shape)

#let us import the linear regression from sklearn & create instance of the model.

from sklearn.linear\_model import LinearRegression

lm = LinearRegression()

print(“Model:\n”,lm)

print(type(lm))

lm.fit(train\_x, train\_y)

predicted\_y = lm.predict(test\_x)

from sklearn import metrics as mt

print("1) The model explains,", np.round(mt.explained\_variance\_score(test\_y,predicted\_y)\*100,2),"% variance of the target w.r.t features is")

print("2) The Mean Absolute Error of model is:", np.round(mt.mean\_absolute\_error(test\_y,predicted\_y ),2))

print("3) The R-Square score of the model is " , np.round(mt.r2\_score(test\_y,predicted\_y),2))

coeff = pd.Series(lm.coef\_, index = train\_x.columns)

intercept = lm.intercept\_

print("Coefficients:\n")

print(coeff)

print("\n")

print("Intercept:\n")

print(intercept)

print("\n")

**Output**:

Sample:

age sex bmi ... s4 s5 s6

374 -0.107226 -0.044642 -0.034229 ... -0.039493 -0.000609 -0.079778

421 0.038076 0.050680 0.016428 ... 0.071210 0.049769 0.015491

340 -0.016412 -0.044642 -0.013751 ... -0.039493 -0.035817 -0.030072

278 0.067136 0.050680 -0.036385 ... 0.034309 0.001144 0.032059

354 -0.023677 0.050680 0.045529 ... 0.034309 0.074193 0.061054

[5 rows x 10 columns]

Sample with Target :

age sex bmi ... s5 s6 Progression

6 -0.045472 0.05068 -0.047163 ... -0.062913 -0.038357 138.0

155 -0.027310 0.05068 0.060618 ... 0.037814 0.048628 186.0

[2 rows x 11 columns]

Null Value Check:

age 0

sex 0

bmi 0

bp 0

s1 0

s2 0

s3 0

s4 0

s5 0

s6 0

Progression 0

dtype: int64

Summary Statistics:

age sex ... s6 Progression

count 4.420000e+02 4.420000e+02 ... 4.420000e+02 442.000000

mean -3.639623e-16 1.309912e-16 ... -3.398488e-16 152.133484

std 4.761905e-02 4.761905e-02 ... 4.761905e-02 77.093005

min -1.072256e-01 -4.464164e-02 ... -1.377672e-01 25.000000

25% -3.729927e-02 -4.464164e-02 ... -3.317903e-02 87.000000

50% 5.383060e-03 -4.464164e-02 ... -1.077698e-03 140.500000

75% 3.807591e-02 5.068012e-02 ... 2.791705e-02 211.500000

max 1.107267e-01 5.068012e-02 ... 1.356118e-01 346.000000

[8 rows x 11 columns]

Correlation

age sex bmi ... s5 s6 Progression

age 1.000000 0.173737 0.185085 ... 0.270777 0.301731 0.187889

sex 0.173737 1.000000 0.088161 ... 0.149918 0.208133 0.043062

bmi 0.185085 0.088161 1.000000 ... 0.446159 0.388680 0.586450

bp 0.335427 0.241013 0.395415 ... 0.393478 0.390429 0.441484

s1 0.260061 0.035277 0.249777 ... 0.515501 0.325717 0.212022

s2 0.219243 0.142637 0.261170 ... 0.318353 0.290600 0.174054

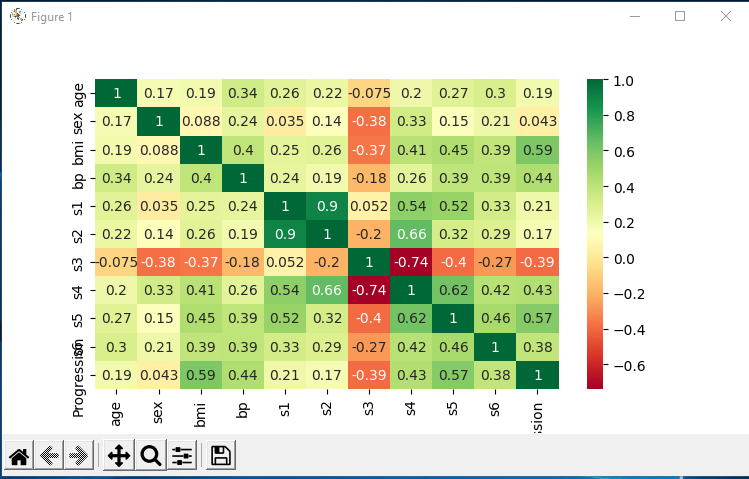
s3 -0.075181 -0.379090 -0.366811 ... -0.398577 -0.273697 -0.394789

s4 0.203841 0.332115 0.413807 ... 0.617857 0.417212 0.430453

s5 0.270777 0.149918 0.446159 ... 1.000000 0.464670 0.565883

s6 0.301731 0.208133 0.388680 ... 0.464670 1.000000 0.382483

Progression 0.187889 0.043062 0.586450 ... 0.565883 0.382483 1.000000

[11 rows x 11 columns] 

train\_x: (331, 10)

test\_x: (111, 10)

train\_y: (331,)

test\_y: (111,)

Model:

LinearRegression()

<class 'sklearn.linear\_model.\_base.LinearRegression'>

1) The model explains, 57.57 % variance of the target w.r.t features is

2) The Mean Absolute Error of model is: 38.08

3) The R-Square score of the model is 0.56

Coefficients:

age 54.820535

sex -260.930304

bmi 458.001802

bp 303.502332

s1 -995.584889

s2 698.811401

s3 183.095229

s4 185.698494

s5 838.503887

s6 96.441048

dtype: float64

Intercept:

154.42752615353518

**Practical No. 2: Implement Logistic Regression (Iris Dataset)**

**Code:**

#https://medium.com/@kgpvijaybg/logistic-regression-on-iris-dataset-48b2ecdfb6d3

import matplotlib.pyplot as plt

import pandas as pd

import sklearn

# Importing the dataset

from sklearn import datasets

from sklearn import preprocessing

iris = datasets.load\_iris()

##dataset = pd.read\_csv(‘iris.csv’)

print("described:\n ",iris.DESCR)

##print(iris.sample(5))

iris.feature\_names #checking the feature names

iris.data.shape #checking the shape of data

iris.target.shape

db\_df = pd.DataFrame(iris.data,columns=iris.feature\_names)

print("sample:\n",db\_df.sample(5))

db\_df['Species'] = iris.target #new column name 'Progression'

print('Sample with target:\n',db\_df.sample(5)) #checking the dataset once again.

# Splitting the dataset into the Training set and Test set

#X = iris.iloc[:, [0,1,2, 3]].values

#y = iris.iloc[:, 4].values

x = db\_df.drop(labels='Species', axis=1) #axis=1 means we

y = db\_df['Species']

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

train\_x, test\_x, train\_y, test\_y = train\_test\_split(x, y, test\_size = 0.25, random\_state = 0)

print("Train And Test sets for x and y: ")

print(train\_x.shape)

print(test\_x.shape)

print(train\_y.shape)

print(test\_y.shape)

# Fitting Logistic Regression to the Training set

from sklearn.linear\_model import LogisticRegression

classifier = LogisticRegression(random\_state = 0, solver='lbfgs', multi\_class='auto')

classifier.fit(train\_x, train\_y)

# Predicting the Test set results

y\_pred = classifier.predict(test\_x)

# Predict probabilities

probs\_y=classifier.predict\_proba(test\_x)

print(probs\_y)

probs\_y = np.round(probs\_y, 2)

res = "{:<10} | {:<10} | {:<10} | {:<13} | {:<5}".format("test\_y", "y\_pred", "Setosa(%)", "versicolor(%)", "virginica(%)\n")

res += "-"\*65+"\n"

res += "\n".join("{:<10} | {:<10} | {:<10} | {:<13} | {:<10}".format(x, y, a, b, c) for x, y, a, b, c in zip(test\_y, y\_pred, probs\_y[:,0], probs\_y[:,1], probs\_y[:,2]))

res += "\n"+"-"\*65+"\n"

print("Result:\n",res)

from sklearn.metrics import confusion\_matrix

cm = confusion\_matrix(test\_y, y\_pred)

print("confusion\_matrix:\n",cm)

# Plot confusion matrix

import seaborn as sns

import pandas as pd

# confusion matrix sns heatmap

ax = plt.axes()

df\_cm = cm

sns.heatmap(df\_cm, annot=True, annot\_kws={"size": 30}, fmt='d',cmap="Blues", ax = ax )

ax.set\_title('Confusion Matrix')

plt.show()

**Output:**

described:

.. \_iris\_dataset:

Iris plants dataset

--------------------

\*\*Data Set Characteristics:\*\*

:Number of Instances: 150 (50 in each of three classes)

:Number of Attributes: 4 numeric, predictive attributes and the class

:Attribute Information:

- sepal length in cm

- sepal width in cm

- petal length in cm

- petal width in cm

- class:

- Iris-Setosa

- Iris-Versicolour

- Iris-Virginica

:Summary Statistics:

============== ==== ==== ======= ===== ====================

Min Max Mean SD Class Correlation

============== ==== ==== ======= ===== ====================

sepal length: 4.3 7.9 5.84 0.83 0.7826

sepal width: 2.0 4.4 3.05 0.43 -0.4194

petal length: 1.0 6.9 3.76 1.76 0.9490 (high!)

petal width: 0.1 2.5 1.20 0.76 0.9565 (high!)

============== ==== ==== ======= ===== ====================

:Missing Attribute Values: None

:Class Distribution: 33.3% for each of 3 classes.

:Creator: R.A. Fisher

:Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)

:Date: July, 1988

The famous Iris database, first used by Sir R.A. Fisher. The dataset is taken

from Fisher's paper. Note that it's the same as in R, but not as in the UCI

Machine Learning Repository, which has two wrong data points.

This is perhaps the best known database to be found in the

pattern recognition literature. Fisher's paper is a classic in the field and

is referenced frequently to this day. (See Duda & Hart, for example.) The

data set contains 3 classes of 50 instances each, where each class refers to a

type of iris plant. One class is linearly separable from the other 2; the

latter are NOT linearly separable from each other.

.. topic:: References

- Fisher, R.A. "The use of multiple measurements in taxonomic problems"

Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to

Mathematical Statistics" (John Wiley, NY, 1950).

- Duda, R.O., & Hart, P.E. (1973) Pattern Classification and Scene Analysis.

(Q327.D83) John Wiley & Sons. ISBN 0-471-22361-1. See page 218.

- Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New System

Structure and Classification Rule for Recognition in Partially Exposed

Environments". IEEE Transactions on Pattern Analysis and Machine

Intelligence, Vol. PAMI-2, No. 1, 67-71.

- Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule". IEEE Transactions

on Information Theory, May 1972, 431-433.

- See also: 1988 MLC Proceedings, 54-64. Cheeseman et al"s AUTOCLASS II

conceptual clustering system finds 3 classes in the data.

- Many, many more ...

sample:

sepal length (cm) sepal width (cm) petal length (cm) petal width (cm)

130 7.4 2.8 6.1 1.9

52 6.9 3.1 4.9 1.5

111 6.4 2.7 5.3 1.9

69 5.6 2.5 3.9 1.1

15 5.7 4.4 1.5 0.4

Sample with target:

sepal length (cm) sepal width (cm) ... petal width (cm) Species

109 7.2 3.6 ... 2.5 2

114 5.8 2.8 ... 2.4 2

81 5.5 2.4 ... 1.0 1

21 5.1 3.7 ... 0.4 0

38 4.4 3.0 ... 0.2 0

[5 rows x 5 columns]

Train And Test sets for x and y:

(112, 4)

(38, 4)

(112,)

(38,)

[[1.17923448e-04 5.61475773e-02 9.43734499e-01]

[1.26289624e-02 9.60454389e-01 2.69166483e-02]

[9.84397637e-01 1.56023240e-02 3.85609887e-08]

[1.25176637e-06 2.31523221e-02 9.76846426e-01]

[9.70234853e-01 2.97649846e-02 1.62596819e-07]

[2.01667077e-06 5.94450398e-03 9.94053479e-01]

[9.81899524e-01 1.81004053e-02 7.04417688e-08]

[2.84241362e-03 7.47090851e-01 2.50066735e-01]

[1.50915507e-03 7.38522498e-01 2.59968347e-01]

[2.05288365e-02 9.35891456e-01 4.35797074e-02]

[9.22416596e-05 1.59472152e-01 8.40435606e-01]

[6.98628082e-03 8.09991333e-01 1.83022387e-01]

[4.08220648e-03 7.93602098e-01 2.02315696e-01]

[3.05681833e-03 7.60910063e-01 2.36033118e-01]

[3.87699816e-03 7.10277107e-01 2.85845895e-01]

[9.82815608e-01 1.71843351e-02 5.65441274e-08]

[6.72901788e-03 7.56466095e-01 2.36804887e-01]

[1.14291989e-02 8.45110532e-01 1.43460269e-01]

[9.67582258e-01 3.24175277e-02 2.14232285e-07]

[9.82872104e-01 1.71278366e-02 5.96858685e-08]

[8.34494291e-04 1.93259363e-01 8.05906142e-01]

[1.03255983e-02 7.11148969e-01 2.78525433e-01]

[9.44128927e-01 5.58700247e-02 1.04835588e-06]

[9.75498625e-01 2.45012077e-02 1.67517340e-07]

[1.36907121e-03 4.26370766e-01 5.72260163e-01]

[9.94203375e-01 5.79661570e-03 9.65257810e-09]

[9.50240496e-01 4.97583716e-02 1.13237159e-06]

[1.07122732e-02 9.00995361e-01 8.82923653e-02]

[1.40885575e-01 8.52873507e-01 6.24091831e-03]

[9.61492038e-01 3.85075120e-02 4.49502590e-07]

[9.90725711e-05 1.15643973e-01 8.84256954e-01]

[1.19870371e-02 6.84361361e-01 3.03651602e-01]

[9.68058508e-01 3.19413418e-02 1.50143020e-07]

[1.28526238e-03 3.57780774e-01 6.40933964e-01]

[1.48833267e-05 3.38267926e-02 9.66158324e-01]

[4.81305998e-02 8.80739889e-01 7.11295109e-02]

[9.44629258e-01 5.53703512e-02 3.91111701e-07]

[6.02621023e-04 3.11030102e-01 6.88367277e-01]]

Result:

test\_y | y\_pred | Setosa(%) | versicolor(%) | virginica(%)

-----------------------------------------------------------------

2 | 2 | 0.0 | 0.06 | 0.94

1 | 1 | 0.01 | 0.96 | 0.03

0 | 0 | 0.98 | 0.02 | 0.0

2 | 2 | 0.0 | 0.02 | 0.98

0 | 0 | 0.97 | 0.03 | 0.0

2 | 2 | 0.0 | 0.01 | 0.99

0 | 0 | 0.98 | 0.02 | 0.0

1 | 1 | 0.0 | 0.75 | 0.25

1 | 1 | 0.0 | 0.74 | 0.26

1 | 1 | 0.02 | 0.94 | 0.04

2 | 2 | 0.0 | 0.16 | 0.84

1 | 1 | 0.01 | 0.81 | 0.18

1 | 1 | 0.0 | 0.79 | 0.2

1 | 1 | 0.0 | 0.76 | 0.24

1 | 1 | 0.0 | 0.71 | 0.29

0 | 0 | 0.98 | 0.02 | 0.0

1 | 1 | 0.01 | 0.76 | 0.24

1 | 1 | 0.01 | 0.85 | 0.14

0 | 0 | 0.97 | 0.03 | 0.0

0 | 0 | 0.98 | 0.02 | 0.0

2 | 2 | 0.0 | 0.19 | 0.81

1 | 1 | 0.01 | 0.71 | 0.28

0 | 0 | 0.94 | 0.06 | 0.0

0 | 0 | 0.98 | 0.02 | 0.0

2 | 2 | 0.0 | 0.43 | 0.57

0 | 0 | 0.99 | 0.01 | 0.0

0 | 0 | 0.95 | 0.05 | 0.0

1 | 1 | 0.01 | 0.9 | 0.09

1 | 1 | 0.14 | 0.85 | 0.01

0 | 0 | 0.96 | 0.04 | 0.0

2 | 2 | 0.0 | 0.12 | 0.88

1 | 1 | 0.01 | 0.68 | 0.3

0 | 0 | 0.97 | 0.03 | 0.0

2 | 2 | 0.0 | 0.36 | 0.64

2 | 2 | 0.0 | 0.03 | 0.97

1 | 1 | 0.05 | 0.88 | 0.07

0 | 0 | 0.94 | 0.06 | 0.0

1 | 2 | 0.0 | 0.31 | 0.69

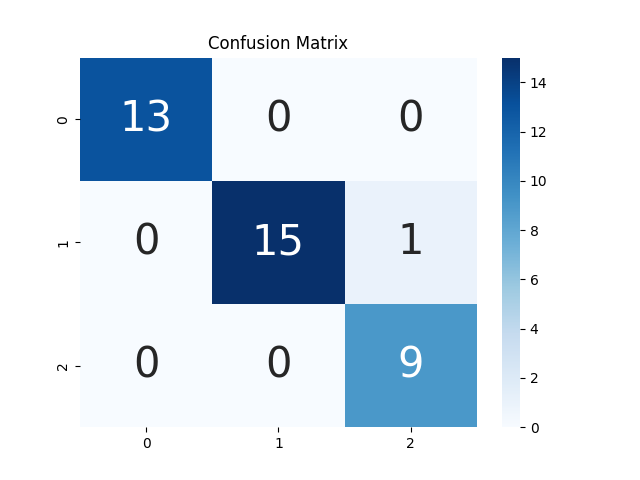
-----------------------------------------------------------------

confusion\_matrix:

[[13 0 0]

[ 0 15 1]

[ 0 0 9]]



**Practical No. 3: Implements Multinomial Logistic Regression (Iris Dataset)**

**Code:**

#Importing the libraries

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

import sklearn

# Importing the dataset

from sklearn import datasets

from sklearn import preprocessing

iris = datasets.load\_iris()

#print(iris.DESCR)

#print("Feature names:\n",iris.feature\_names) #checking the feature names

#print("data shape: ",iris.data.shape) #checking the shape of data

#print("target shape: ",iris.target.shape)

db\_df = pd.DataFrame(iris.data,columns=iris.feature\_names)

print("sample:\n",db\_df.sample(5))

db\_df['Species'] = iris.target #new column name 'Species'

print('Sample with target:\n',db\_df.sample(5)) #checking the dataset once again.

# Splitting the dataset into the Training set and Test set

#X = iris.iloc[:, [0,1,2, 3]].values

#y = iris.iloc[:, 4].values

x = db\_df.drop(labels='Species', axis=1) #axis=1 means we

y = db\_df['Species']

print("target column:\n",y)

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

train\_x, test\_x, train\_y, test\_y = train\_test\_split(x, y, test\_size = 0.25, random\_state = 0)

print("Train and Test set:\n",y)

print(train\_x.shape)

print(test\_x.shape)

print(train\_y.shape)

print(test\_y.shape)

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

train\_x = sc.fit\_transform(train\_x)

test\_x = sc.transform(test\_x)

# Fitting Logistic Regression to the Training set

from sklearn.linear\_model import LogisticRegression

classifier = LogisticRegression(random\_state = 0, solver='lbfgs', multi\_class='multinomial')

classifier.fit(train\_x, train\_y)

# Predicting the Test set results

y\_pred = classifier.predict(test\_x)

print("\npredicted y:\n",y\_pred)

from sklearn.metrics import accuracy\_score

print("\nAccuracy Score= ",accuracy\_score(test\_y, y\_pred))

from sklearn.metrics import confusion\_matrix

cm = confusion\_matrix(test\_y, y\_pred)

print("\nconfusion\_matrix:\n",cm)

# Predict probabilities

probs\_y=classifier.predict\_proba(test\_x)

print("\n probabilities of y: \n",probs\_y)

**Output:**

Sample:

sepal length (cm) sepal width (cm) petal length (cm) petal width (cm)

4 5.0 3.6 1.4 0.2

70 5.9 3.2 4.8 1.8

41 4.5 2.3 1.3 0.3

129 7.2 3.0 5.8 1.6

14 5.8 4.0 1.2 0.2

Sample with target:

sepal length (cm) sepal width (cm) ... petal width (cm) Species

18 5.7 3.8 ... 0.3 0

27 5.2 3.5 ... 0.2 0

122 7.7 2.8 ... 2.0 2

149 5.9 3.0 ... 1.8 2

147 6.5 3.0 ... 2.0 2

[5 rows x 5 columns]

target column:

0 0

1 0

2 0

3 0

4 0

..

145 2

146 2

147 2

148 2

149 2

Name: Species, Length: 150, dtype: int32

Train and Test set:

(112, 4)

(38, 4)

(112,)

(38,)

predicted y:

[2 1 0 2 0 2 0 1 1 1 2 1 1 1 1 0 1 1 0 0 2 1 0 0 2 0 0 1 1 0 2 1 0 2 2 1 0 2]

Accuracy Score= 0.9736842105263158

confusion\_matrix:

[[13 0 0]

[ 0 15 1]

[ 0 0 9]]

**Practical No. 4: Implement SVM classifier (Iris Dataset)**

**Code:**

#import the dependencies

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import sklearn as sk

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

from sklearn.svm import SVC

from sklearn.metrics import confusion\_matrix

from sklearn.model\_selection import cross\_val\_score

from sklearn.metrics import accuracy\_score

#load the dataset

data=sns.load\_dataset("iris")

print("Sample:\n",data.head())

#Encoding the categorical column

dataset=data.replace({"setosa":1,"versicolor":2,"virginica":3})

print("Encoding the categorical column \n", dataset.head())

#plot the correlation

plt.figure(1)

sns.heatmap(dataset.corr())

plt.title('Correlation On iris Classes')

plt.show()

#splitting dataset

X = dataset.iloc[:,:-1]

y = dataset.iloc[:, -1].values

print("features:\n", X)

print("target:\n", y)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.25, random\_state = 0)

#creating SVM MODEL

classifier = SVC(kernel = 'linear', random\_state = 0)

#Fit the model for the data

classifier.fit(X\_train, y\_train)#train the model

#Make the prediction

y\_pred = classifier.predict(X\_test)

#creating Confusion matrix

cm = confusion\_matrix(y\_test, y\_pred)

print("Confusion matrix:\n", cm)

accuracies = cross\_val\_score(estimator = classifier, X = X\_train, y = y\_train, cv = 10)

print("Accuracy: ",accuracies.mean()\*100," %")

print("Standard Deviation: {:.2f} %".format(accuracies.std()\*100))

**Output:**

Sample:

sepal\_length sepal\_width petal\_length petal\_width species

0 5.1 3.5 1.4 0.2 setosa

1 4.9 3.0 1.4 0.2 setosa

2 4.7 3.2 1.3 0.2 setosa

3 4.6 3.1 1.5 0.2 setosa

4 5.0 3.6 1.4 0.2 setosa

Encoding the categorical column

sepal\_length sepal\_width petal\_length petal\_width species

0 5.1 3.5 1.4 0.2 1

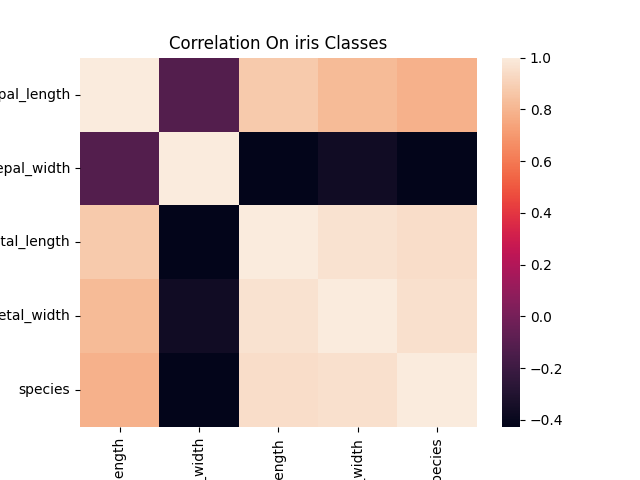
1 4.9 3.0 1.4 0.2 1

2 4.7 3.2 1.3 0.2 1

3 4.6 3.1 1.5 0.2 1

4 5.0 3.6 1.4 0.2 1

Confusion matrix:



Features:

sepal\_length sepal\_width petal\_length petal\_width

0 5.1 3.5 1.4 0.2

1 4.9 3.0 1.4 0.2

2 4.7 3.2 1.3 0.2

3 4.6 3.1 1.5 0.2

4 5.0 3.6 1.4 0.2

.. ... ... ... ...

145 6.7 3.0 5.2 2.3

146 6.3 2.5 5.0 1.9

147 6.5 3.0 5.2 2.0

148 6.2 3.4 5.4 2.3

149 5.9 3.0 5.1 1.8

Target:

[150 rows x 4 columns]

[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2

2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 3

3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3

3 3]

Confusion Matrix:

[[13 0 0]

[ 0 15 1]

[ 0 0 9]]

Accuracy: 98.18181818181819 %

Standard Deviation: 3.64 %

**Practical 5: Train and fine-tune a Decision Tree for the Moons Dataset**

**Code:**

import numpy as np

import matplotlib.pyplot as plt

# This function will help in visualization of our dataset.

print("Visualization of our dataset:\n")

def plot\_dataset(X, y, axes):

plt.figure(figsize=(10,6))

plt.plot(X[:, 0][y==0], X[:, 1][y==0], "bs",alpha = 0.5)

plt.plot(X[:, 0][y==1], X[:, 1][y==1], "g^",alpha = 0.2)

plt.axis(axes)

plt.grid(True, which='both')

plt.xlabel(r"$x\_1$", fontsize=20)

plt.ylabel(r"$x\_2$", fontsize=20, rotation=0)

plt.show()

from sklearn.datasets import make\_moons

X, y = make\_moons(n\_samples=50, noise=0.4, random\_state=21)

plot\_dataset(X, y, [-3, 5, -3, 3])

print("features\n",X)

print("labels\n",y)

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=0)

from sklearn.tree import DecisionTreeClassifier

tree\_clf = DecisionTreeClassifier()

from sklearn.model\_selection import GridSearchCV

parameter = {

'criterion' : ["gini", "entropy"],

'max\_leaf\_nodes': list(range(2,5)),

'min\_samples\_split': [2, 3, 4]

}

clf = GridSearchCV(tree\_clf, parameter, cv = 5,scoring = "accuracy",return\_train\_score=True,n\_jobs=-1)

print(clf)

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

print("\n Best parameters:\n",clf.best\_params\_)

cvres = clf.cv\_results\_

for mean\_score, params in zip(cvres["mean\_train\_score"], cvres["params"]):

print(mean\_score, params)

print(clf.score(X\_train, y\_train))

from sklearn.metrics import confusion\_matrix

pred = clf.predict(X\_train)

print("confusion\_Matrix: \n",confusion\_matrix(y\_train,pred))

from sklearn.metrics import precision\_score, recall\_score

pre = precision\_score(y\_train, pred)

re = recall\_score(y\_train, pred)

print(f"Precision: {pre} Recall: {re}")

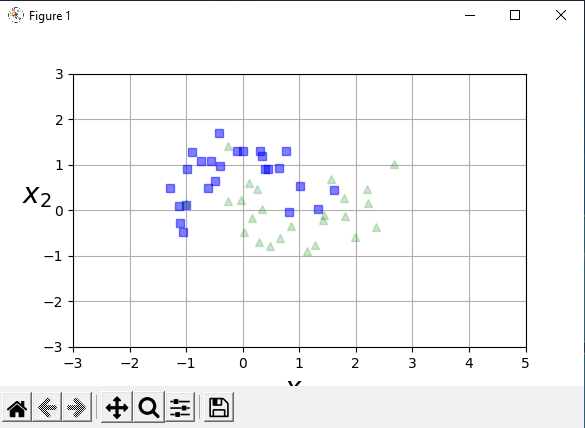
from sklearn.metrics import f1\_score

print("fi score: ",f1\_score(y\_train, pred))

print("Accuracy\_score: ",clf.score(X\_test, y\_test))

**Output:**

Visualization of our dataset:



features

[[ 7.69850946e-01 1.29114646e+00]

[ 1.78381880e+00 2.72265654e-01]

[ 1.80509236e+00 -1.17467925e-01]

[ 1.18672479e-01 5.88301355e-01]

[-4.28139378e-01 1.70417187e+00]

[ 1.33397165e+00 3.56525531e-02]

[-1.10553488e+00 -2.85980462e-01]

[ 4.85040989e-01 -7.78057823e-01]

[-7.42067997e-01 1.09227674e+00]

[ 2.19001768e+00 4.70664529e-01]

[-2.65705548e-01 1.97098604e-01]

[-9.77633526e-01 9.12538503e-01]

[-9.03924399e-01 1.28058182e+00]

[ 2.96590942e-01 1.29051750e+00]

[-1.12293799e+00 9.62894162e-02]

[ 3.45661611e-01 3.43354670e-02]

[ 8.20927027e-01 -4.64336073e-02]

[ 3.37079654e-01 1.19769044e+00]

[-2.70264247e-02 2.31334893e-01]

[ 1.43816386e+00 -1.03632942e-01]

[ 1.55640659e+00 6.78121786e-01]

[ 2.67034253e+00 1.00795935e+00]

[ 2.21961989e+00 1.58867606e-01]

[ 2.35116879e+00 -3.60500780e-01]

[-1.00990060e+00 1.69682810e-01]

[ 1.41344505e+00 -2.17918569e-01]

[ 1.61155087e+00 4.36693971e-01]

[ 1.72597686e-01 -1.80345967e-01]

[-1.05179551e+00 -4.72573783e-01]

[ 2.43706496e-01 4.67742632e-01]

[ 1.00796242e+00 5.33364557e-01]

[-2.69019182e-01 1.41059912e+00]

[-1.29331113e+00 4.95279940e-01]

[ 1.98711986e+00 -5.93587002e-01]

[ 3.89402764e-01 9.10466111e-01]

[-4.07743116e-01 9.80160465e-01]

[ 4.50885213e-01 9.10072703e-01]

[-4.83909999e-01 6.32987104e-01]

[ 8.60089006e-01 -3.34847064e-01]

[-1.00691074e+00 1.19279257e-01]

[ 1.14312357e+00 -9.00825976e-01]

[ 6.67528905e-01 -6.16204909e-01]

[ 1.27284477e+00 -7.68469110e-01]

[ 8.31820260e-04 1.30546121e+00]

[-5.68232154e-01 1.08507795e+00]

[ 2.95289502e-01 -7.07006173e-01]

[ 2.79673458e-02 -4.78145645e-01]

[ 6.44104815e-01 9.23587642e-01]

[-6.09008276e-01 4.87943887e-01]

[-9.77682449e-02 1.30660471e+00]]

labels

[0 1 1 1 0 0 0 1 0 1 1 0 0 0 0 1 0 0 1 1 1 1 1 1 1 1 0 1 0 1 0 1 0 1 0 0 0

0 1 0 1 1 1 0 0 1 1 0 0 0]

GridSearchCV(cv=5, estimator=DecisionTreeClassifier(), n\_jobs=-1,

param\_grid={'criterion': ['gini', 'entropy'],

'max\_leaf\_nodes': [2, 3, 4],

'min\_samples\_split': [2, 3, 4]},

return\_train\_score=True, scoring='accuracy')

Best parameters:

{'criterion': 'gini', 'max\_leaf\_nodes': 3, 'min\_samples\_split': 2}

0.78125 {'criterion': 'gini', 'max\_leaf\_nodes': 2, 'min\_samples\_split': 2}

0.78125 {'criterion': 'gini', 'max\_leaf\_nodes': 2, 'min\_samples\_split': 3}

0.78125 {'criterion': 'gini', 'max\_leaf\_nodes': 2, 'min\_samples\_split': 4}

0.85625 {'criterion': 'gini', 'max\_leaf\_nodes': 3, 'min\_samples\_split': 2}

0.85625 {'criterion': 'gini', 'max\_leaf\_nodes': 3, 'min\_samples\_split': 3}

0.85625 {'criterion': 'gini', 'max\_leaf\_nodes': 3, 'min\_samples\_split': 4}

0.8875 {'criterion': 'gini', 'max\_leaf\_nodes': 4, 'min\_samples\_split': 2}

0.8875 {'criterion': 'gini', 'max\_leaf\_nodes': 4, 'min\_samples\_split': 3}

0.8875 {'criterion': 'gini', 'max\_leaf\_nodes': 4, 'min\_samples\_split': 4}

0.775 {'criterion': 'entropy', 'max\_leaf\_nodes': 2, 'min\_samples\_split': 2}

0.775 {'criterion': 'entropy', 'max\_leaf\_nodes': 2, 'min\_samples\_split': 3}

0.775 {'criterion': 'entropy', 'max\_leaf\_nodes': 2, 'min\_samples\_split': 4}

0.8 {'criterion': 'entropy', 'max\_leaf\_nodes': 3, 'min\_samples\_split': 2}

0.8 {'criterion': 'entropy', 'max\_leaf\_nodes': 3, 'min\_samples\_split': 3}

0.8 {'criterion': 'entropy', 'max\_leaf\_nodes': 3, 'min\_samples\_split': 4}

0.875 {'criterion': 'entropy', 'max\_leaf\_nodes': 4, 'min\_samples\_split': 2}

0.875 {'criterion': 'entropy', 'max\_leaf\_nodes': 4, 'min\_samples\_split': 3}

0.875 {'criterion': 'entropy', 'max\_leaf\_nodes': 4, 'min\_samples\_split': 4}

0.825

confusion\_Matrix:

[[16 6]

[ 1 17]]

Precision: 0.7391304347826086 Recall: 0.9444444444444444

F1 score: 0.8292682926829269

Accuracy\_score: 0.8

**Practical 6: Train an SVM regressor on the California Housing Dataset**

**Code:**

url="https://raw.githubusercontent.com/ageron/handson-ml/master/datasets/housing/housing.csv"

# Import Libraries

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.preprocessing import LabelEncoder,StandardScaler

# Import Dataset

dataset = pd.read\_csv(url)

print("sample:\n",dataset.head()) # Print first 5 observations from dataset using head()

print(dataset.columns)

# Check in which column contains nan values

#print(dataset.isnull().any())

print("Null value check:\n",dataset.isna().sum())

dataset.total\_bedrooms=dataset.total\_bedrooms.fillna(dataset.total\_bedrooms.mean())

print("Removing Null Values:\n",dataset.isnull().sum())

#Convert categorical column in the dataset to numerical data.

from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

dataset['ocean\_proximity']=le.fit\_transform(dataset['ocean\_proximity'])

print("Ocean Proximity Column:\n",dataset['ocean\_proximity'])

# Get column names first

names = dataset.columns

# Create the Scaler object

scaler = StandardScaler()

# Fit your data on the scaler object

scaled\_df = scaler.fit\_transform(dataset)

scaled\_df = pd.DataFrame(scaled\_df, columns=names)

print("Scaled dataset\n",scaled\_df.head())

#Extract input (X) and output (Y) data from the dataset.

X\_Features=['longitude', 'latitude', 'housing\_median\_age', 'total\_rooms',

'total\_bedrooms', 'population', 'households', 'median\_income',

'ocean\_proximity']

X=scaled\_df[X\_Features]

Y=scaled\_df['median\_house\_value']

print("Dataset:",dataset.shape)

print("features:",X.shape)

print("target:",Y.shape)

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 ("Xtrain and Ytrain:\n",x\_train.shape, y\_train.shape)

print ("Xtest and Ytest:\n",x\_test.shape, y\_test.shape)

# Support Vector Regression

from sklearn.svm import SVR

SVR()

model\_svr = SVR(kernel="rbf")

model\_svr.fit(x\_train,y\_train)

# Perform prediction and model score

from sklearn.metrics import mean\_squared\_error

y\_pred = model\_svr.predict(x\_test)

from sklearn.metrics import r2\_score

print("Model Score for Training data: {}".format(model\_svr.score(x\_train,y\_train)))

print("Model Score for Testing data r2\_score: {}".format(r2\_score(y\_test,y\_pred)))

print("Root Mean Squared Error is {}".format(np.sqrt(mean\_squared\_error(y\_test,y\_pred))))

**Output:**

sample:

longitude latitude ... median\_house\_value ocean\_proximity

0 -122.23 37.88 ... 452600.0 NEAR BAY

1 -122.22 37.86 ... 358500.0 NEAR BAY

2 -122.24 37.85 ... 352100.0 NEAR BAY

3 -122.25 37.85 ... 341300.0 NEAR BAY

4 -122.25 37.85 ... 342200.0 NEAR BAY

[5 rows x 10 columns]

Index(['longitude', 'latitude', 'housing\_median\_age', 'total\_rooms',

'total\_bedrooms', 'population', 'households', 'median\_income',

'median\_house\_value', 'ocean\_proximity'],

dtype='object')

Null value check:

longitude 0

latitude 0

housing\_median\_age 0

total\_rooms 0

total\_bedrooms 207

population 0

households 0

median\_income 0

median\_house\_value 0

ocean\_proximity 0

dtype: int64

Removing Null Values:

longitude 0

latitude 0

housing\_median\_age 0

total\_rooms 0

total\_bedrooms 0

population 0

households 0

median\_income 0

median\_house\_value 0

ocean\_proximity 0

dtype: int64

Ocean Proximity Column:

0 3

1 3

2 3

3 3

4 3

..

20635 1

20636 1

20637 1

20638 1

20639 1

Name: ocean\_proximity, Length: 20640, dtype: int32

Scaled dataset

longitude latitude ... median\_house\_value ocean\_proximity

0 -1.327835 1.052548 ... 2.129631 1.291089

1 -1.322844 1.043185 ... 1.314156 1.291089

2 -1.332827 1.038503 ... 1.258693 1.291089

3 -1.337818 1.038503 ... 1.165100 1.291089

4 -1.337818 1.038503 ... 1.172900 1.291089

[5 rows x 10 columns]

Dataset: (20640, 10)

features: (20640, 9)

target: (20640,)

Xtrain and Ytrain:

(16512, 9) (16512,)

Xtest and Ytest:

(4128, 9) (4128,)

Model Score for Training data: 0.7702301410088314

Model Score for Testing data r2\_score: 0.7562480760121925

Root Mean Squared Error is 0.49001681281704

**Practical 7: Implement Batch Gradient Descent with early stopping for Softmax Regression**

**Code:**

#Let's import IRIS data

from sklearn import datasets

iris = datasets.load\_iris()

print(iris.keys())

#print(iris.DESCR)

print('iris.data.shape = ',iris.data.shape)

print('iris.target.shape = ',iris.target.shape)

from sklearn.linear\_model import LogisticRegression

X = iris.data[:, (2,3)]

y = iris.target

softmax\_reg = LogisticRegression(multi\_class = 'multinomial', solver = 'lbfgs', C =10)

softmax\_reg.fit(X,y)

print("Predicted value:" ,softmax\_reg.predict([[5,2]]))

print("probability:", softmax\_reg.predict\_proba([[5,2]]))

import numpy as np

print("bincount:", np.bincount(y))

#Let's now use softmax regression

# Add a bias term in X

X\_with\_bias = np.c\_[np.ones([len(X), 1]), X]

#print(X\_with\_bias)

# Dividing into train-val-test

test\_ratio = 0.2

validation\_ratio = 0.2

total\_size = len(X\_with\_bias)

test\_size = int(total\_size \* test\_ratio)

validation\_size = int(total\_size \* validation\_ratio)

train\_size = total\_size - test\_size - validation\_size

rnd\_indices = np.random.permutation(total\_size)

X\_train = X\_with\_bias[rnd\_indices[:train\_size]]

y\_train = y[rnd\_indices[:train\_size]]

X\_valid = X\_with\_bias[rnd\_indices[train\_size:-test\_size]]

y\_valid = y[rnd\_indices[train\_size:-test\_size]]

X\_test = X\_with\_bias[rnd\_indices[-test\_size:]]

y\_test = y[rnd\_indices[-test\_size:]]

def to\_one\_hot(y):

n\_classes = y.max() + 1

m = len(y)

Y\_one\_hot = np.zeros((m, n\_classes))

Y\_one\_hot[np.arange(m), y] = 1

return Y\_one\_hot

Y\_train\_one\_hot = to\_one\_hot(y\_train)

Y\_valid\_one\_hot = to\_one\_hot(y\_valid)

Y\_test\_one\_hot = to\_one\_hot(y\_test)

def softmax(logits):

exps = np.exp(logits)

exp\_sums = np.sum(exps, axis=1, keepdims=True)

return exps / exp\_sums

n\_inputs = X\_train.shape[1] # == 3 (2 features plus the bias term)

print("n\_input:",n\_inputs)

n\_outputs = len(np.unique(y\_train)) # == 3 (3 iris classes)

print("n\_output:",n\_outputs)

eta = 0.01

n\_iterations = 5001

m = len(X\_train)

epsilon = 1e-7

Theta = np.random.randn(n\_inputs, n\_outputs)

print("Iteration and Loss\n")

for iteration in range(n\_iterations):

logits = X\_train.dot(Theta)

Y\_proba = softmax(logits)

loss = -np.mean(np.sum(Y\_train\_one\_hot \* np.log(Y\_proba + epsilon), axis=1))

error = Y\_proba - Y\_train\_one\_hot

if iteration % 500 == 0:

print(iteration, loss)

gradients = 1/m \* X\_train.T.dot(error)

Theta = Theta - eta \* gradients

**Output:**

dict\_keys(['data', 'target', 'frame', 'target\_names', 'DESCR', 'feature\_names', 'filename', 'data\_module'])

iris.data.shape = (150, 4)

iris.target.shape = (150,)

Predicted value:[2]

Probability:[[6.38014896e-07 5.74929995e-02 9.42506362e-01]]

Bin count:[50 50 50]

n\_input: 3

n\_output: 3

Iteration and Loss

0 1.4042338389689715

500 0.8670740545314494

1000 0.7154713086056899

1500 0.6231545250072532

2000 0.5623730177486371

2500 0.5192161946006294

3000 0.48664147241634687

3500 0.460869354804051

4000 0.4397326723170415

4500 0.4219104999269485

5000 0.40655394373515974

**Practical 8: Implement MLP for classification of handwritten digits (MNIST Dataset)**

**Code:**

import numpy as np

from tensorflow.keras.datasets import mnist

(x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data()

num\_labels = len(np.unique(y\_train))

print("total de labels:t{}".format(num\_labels))

print("labels:ttt{0}".format(np.unique(y\_train)))

from tensorflow.keras.utils import to\_categorical

y\_train = to\_categorical(y\_train)

y\_test = to\_categorical(y\_test)

image\_size = x\_train.shape[1]

input\_size = image\_size \* image\_size

print("x\_train:t{}".format(x\_train.shape))

print("x\_test:tt{}n".format(x\_test.shape))

x\_train = np.reshape(x\_train, [-1, input\_size])

x\_train = x\_train.astype('float32') / 255

x\_test = np.reshape(x\_test, [-1, input\_size])

x\_test = x\_test.astype('float32') / 255

print("x\_train:t{}".format(x\_train.shape))

print("x\_test:tt{}".format(x\_test.shape))

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Activation, Dropout

# Parameters

batch\_size = 128 # It is the sample size of inputs to be processed at each training stage.

hidden\_units = 256

dropout = 0.45

# Nossa MLP com ReLU e Dropout

model = Sequential()

model.add(Dense(hidden\_units, input\_dim=input\_size))

model.add(Activation('relu'))

model.add(Dropout(dropout))

model.add(Dense(hidden\_units))

model.add(Activation('relu'))

model.add(Dropout(dropout))

model.add(Dense(num\_labels))

model.add(Activation('softmax'))

model.summary()

model.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])

model.fit(x\_train, y\_train, epochs=20, batch\_size=batch\_size)

acc = model.evaluate(x\_test,y\_test,batch\_size=batch\_size,verbose=0)

print("\n Accuracy: ",acc)

**Output:**

total de labels: t10

labels: ttt[0 1 2 3 4 5 6 7 8 9]

x\_train: t(60000, 28, 28)

x\_test: tt(10000, 28, 28)n

x\_train: t(60000, 784)

x\_test: tt(10000, 784)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

dense (Dense) (None, 256) 200960

activation (Activation) (None, 256) 0

dropout (Dropout) (None, 256) 0

dense\_1 (Dense) (None, 256) 65792

activation\_1 (Activation) (None, 256) 0

dropout\_1 (Dropout) (None, 256) 0

dense\_2 (Dense) (None, 10) 2570

activation\_2 (Activation) (None, 10) 0

=================================================================

Total params: 269,322

Trainable params: 269,322

Non-trainable params: 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Epoch 1/20

469/469 [==============================] - 4s 4ms/step - loss: 0.4255 - accuracy: 0.8700

Epoch 2/20

469/469 [==============================] - 2s 4ms/step - loss: 0.1950 - accuracy: 0.9420

Epoch 3/20

469/469 [==============================] - 2s 4ms/step - loss: 0.1522 - accuracy: 0.9546

Epoch 4/20

469/469 [==============================] - 2s 5ms/step - loss: 0.1264 - accuracy: 0.9616

Epoch 5/20

469/469 [==============================] - 2s 4ms/step - loss: 0.1124 - accuracy: 0.9662

Epoch 6/20

469/469 [==============================] - 2s 4ms/step - loss: 0.1038 - accuracy: 0.9682

Epoch 7/20

469/469 [==============================] - 2s 4ms/step - loss: 0.0925 - accuracy: 0.9722

Epoch 8/20

469/469 [==============================] - 2s 4ms/step - loss: 0.0893 - accuracy: 0.9729

Epoch 9/20

469/469 [==============================] - 2s 4ms/step - loss: 0.0825 - accuracy: 0.9743

Epoch 10/20

469/469 [==============================] - 2s 5ms/step - loss: 0.0750 - accuracy: 0.9765

Epoch 11/20

469/469 [==============================] - 2s 5ms/step - loss: 0.0721 - accuracy: 0.9762

Epoch 12/20

469/469 [==============================] - 2s 5ms/step - loss: 0.0661 - accuracy: 0.9786

Epoch 13/20

469/469 [==============================] - 2s 4ms/step - loss: 0.0671 - accuracy: 0.9785

Epoch 14/20

469/469 [==============================] - 2s 5ms/step - loss: 0.0636 - accuracy: 0.9800

Epoch 15/20

469/469 [==============================] - 2s 5ms/step - loss: 0.0597 - accuracy: 0.9811

Epoch 16/20

469/469 [==============================] - 2s 5ms/step - loss: 0.0599 - accuracy: 0.9815

Epoch 17/20

469/469 [==============================] - 2s 5ms/step - loss: 0.0573 - accuracy: 0.9818

Epoch 18/20

469/469 [==============================] - 2s 5ms/step - loss: 0.0565 - accuracy: 0.9818

Epoch 19/20

469/469 [==============================] - 2s 5ms/step - loss: 0.0546 - accuracy: 0.9826

Epoch 20/20

469/469 [==============================] - 2s 4ms/step - loss: 0.0509 - accuracy: 0.9835

Accuracy: [0.06719127297401428, 0.9832000136375427]

**Practical 9: Classification of images of clothing using Tensorflow (Fashion MNIST dataset)**

**Code:**

import tensorflow as tf

# Helper libraries

import numpy as np

import matplotlib.pyplot as plt

print(tf.\_\_version\_\_)

fashion\_mnist = tf.keras.datasets.fashion\_mnist

(train\_images, train\_labels), (test\_images, test\_labels) = fashion\_mnist.load\_data()

class\_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',

'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

print("test\_images.shape" ,train\_images.shape)

print("len(train\_labels))",len(train\_labels))

print("train\_labels" ,train\_labels)

print("test\_images.shape" ,test\_images.shape)

print("len(test\_labels)",len(test\_labels))

#plot

print("train\_images[0]\n")

plt.figure()

plt.imshow(train\_images[0])

plt.colorbar()

plt.grid(False)

plt.show()

#scaling

train\_images = train\_images / 255.0

test\_images = test\_images / 255.0

#first 25 images with class names

""

print("first 25 images with class names\n")

plt.figure(figsize=(10,10))

for i in range(25):

plt.subplot(5,5,i+1)

plt.xticks([])

plt.yticks([])

plt.grid(False)

plt.imshow(train\_images[i], cmap=plt.cm.binary)

plt.xlabel(class\_names[train\_labels[i]])

plt.show()

#build model

model = tf.keras.Sequential([

tf.keras.layers.Flatten(input\_shape=(28, 28)),

tf.keras.layers.Dense(128, activation='relu'),

tf.keras.layers.Dense(10)

])

#compile the model

model.compile(optimizer='adam',

loss=tf.keras.losses.SparseCategoricalCrossentropy(from\_logits=True),

metrics=['accuracy'])

#train model

#feed model

model.fit(train\_images, train\_labels, epochs=10)

#evaluate accuracy

"""313/313 [==============================] - 0s 1ms/step - loss: 0.3315 - accuracy: 0.8841 v=1

Epoch 10/10

1875/1875 [==============================] - 3s 2ms/step - loss: 0.2371 - accuracy: 0.9118

313/313 - 0s - loss: 0.3347 - accuracy: 0.8841 - 408ms/epoch - 1ms/step v=2

1875/1875 [==============================] - 3s 2ms/step - loss: 0.2412 - accuracy: 0.9102 v=0"""

test\_loss, test\_acc = model.evaluate(test\_images, test\_labels, verbose=0)

print('\nTest accuracy:', test\_acc)

#make predictions

probability\_model = tf.keras.Sequential([model,

tf.keras.layers.Softmax()])

predictions = probability\_model.predict(test\_images)

print("prediction for 0th image: \n",predictions[0])

print("prediction for 12th image: \n",predictions[12])

#see label

print("label for 0th image: ",np.argmax(predictions[0]))

print("test label: ",test\_labels[0])

print("label for 12th image: ",np.argmax(predictions[12]))

print("test label: ",test\_labels[12])

#Graph this to look at the full set of 10 class predictions.

def plot\_image(i, predictions\_array, true\_label, img):

true\_label, img = true\_label[i], img[i]

plt.grid(False)

plt.xticks([])

plt.yticks([])

plt.imshow(img, cmap=plt.cm.binary)

predicted\_label = np.argmax(predictions\_array)

if predicted\_label == true\_label:

color = 'blue'

else:

color = 'red'

plt.xlabel("{} {:2.0f}% ({})".format(class\_names[predicted\_label],

100\*np.max(predictions\_array),

class\_names[true\_label]),

color=color)

def plot\_value\_array(i, predictions\_array, true\_label):

true\_label = true\_label[i]

plt.grid(False)

plt.xticks(range(10))

plt.yticks([])

thisplot = plt.bar(range(10), predictions\_array, color="#777777")

plt.ylim([0, 1])

predicted\_label = np.argmax(predictions\_array)

thisplot[predicted\_label].set\_color('red')

thisplot[true\_label].set\_color('blue')

#Verify predictions

i = 0

plt.figure(figsize=(6,3))

plt.subplot(1,2,1)

plot\_image(i, predictions[i], test\_labels, test\_images)

plt.subplot(1,2,2)

plot\_value\_array(i, predictions[i], test\_labels)

plt.show()

j = 12

plt.figure(figsize=(6,3))

plt.subplot(1,2,1)

plot\_image(i, predictions[j], test\_labels, test\_images)

plt.subplot(1,2,2)

plot\_value\_array(j, predictions[j], test\_labels)

plt.show()

# Plot the first X test images, their predicted labels, and the true labels.

# Color correct predictions in blue and incorrect predictions in red.

num\_rows = 5

num\_cols = 3

num\_images = num\_rows\*num\_cols

plt.figure(figsize=(2\*2\*num\_cols, 2\*num\_rows))

for i in range(num\_images):

plt.subplot(num\_rows, 2\*num\_cols, 2\*i+1)

plot\_image(i, predictions[i], test\_labels, test\_images)

plt.subplot(num\_rows, 2\*num\_cols, 2\*i+2)

plot\_value\_array(i, predictions[i], test\_labels)

plt.tight\_layout()

plt.show()

# Grab an image from the test dataset.

img = test\_images[1]

print(img.shape)

# Add the image to a batch where it's the only member.

img = (np.expand\_dims(img,0))

print(img.shape)

predictions\_single = probability\_model.predict(img)

print(predictions\_single)

plot\_value\_array(1, predictions\_single[0], test\_labels)

\_ = plt.xticks(range(10), class\_names, rotation=45)

plt.show()

#tf.keras.Model.predict returns a list of lists—one list for each image in the batch of data. Grab the predictions for our (only) image in the batch:

print(np.argmax(predictions\_single[0]))

**Output**:

2.8.0

test\_images.shape (60000, 28, 28)

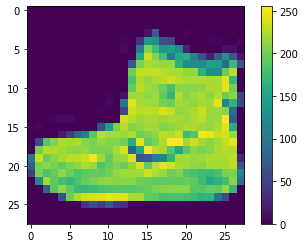
len(train\_labels) 60000

train\_labels [9 0 0 ... 3 0 5]

test\_images.shape (10000, 28, 28)

len(test\_labels) 10000

train\_images[0]



first 25 images with class names



Epoch 1/10

1875/1875 [==============================] - 3s 2ms/step - loss: 0.4944 - accuracy: 0.8274

Epoch 2/10

1875/1875 [==============================] - 3s 2ms/step - loss: 0.3759 - accuracy: 0.8621

Epoch 3/10

1875/1875 [==============================] - 3s 2ms/step - loss: 0.3374 - accuracy: 0.8759

Epoch 4/10

1875/1875 [==============================] - 3s 2ms/step - loss: 0.3126 - accuracy: 0.8849

Epoch 5/10

1875/1875 [==============================] - 3s 2ms/step - loss: 0.2952 - accuracy: 0.8909

Epoch 6/10

1875/1875 [==============================] - 3s 2ms/step - loss: 0.2807 - accuracy: 0.8955

Epoch 7/10

1875/1875 [==============================] - 3s 2ms/step - loss: 0.2675 - accuracy: 0.9008

Epoch 8/10

1875/1875 [==============================] - 4s 2ms/step - loss: 0.2583 - accuracy: 0.9026

Epoch 9/10

1875/1875 [==============================] - 3s 2ms/step - loss: 0.2482 - accuracy: 0.9070

Epoch 10/10

1875/1875 [==============================] - 3s 2ms/step - loss: 0.2404 - accuracy: 0.9101

Test accuracy: 0.8842999935150146

prediction for 0th image:

[6.5588843e-05 1.7892111e-09 8.6709015e-08 5.4267817e-09 6.3819079e-09

6.6716103e-03 3.2332937e-05 5.5502765e-02 9.6100723e-07 9.3772662e-01]

prediction for 12th image:

[3.6343422e-07 2.8975183e-10 3.2890803e-09 8.2366941e-10 4.0584111e-10

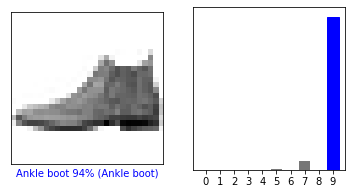
9.8099661e-01 1.1229066e-08 1.6947458e-02 2.0555139e-03 1.4098972e-08]

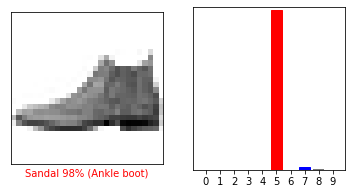
label for 0th image: 9

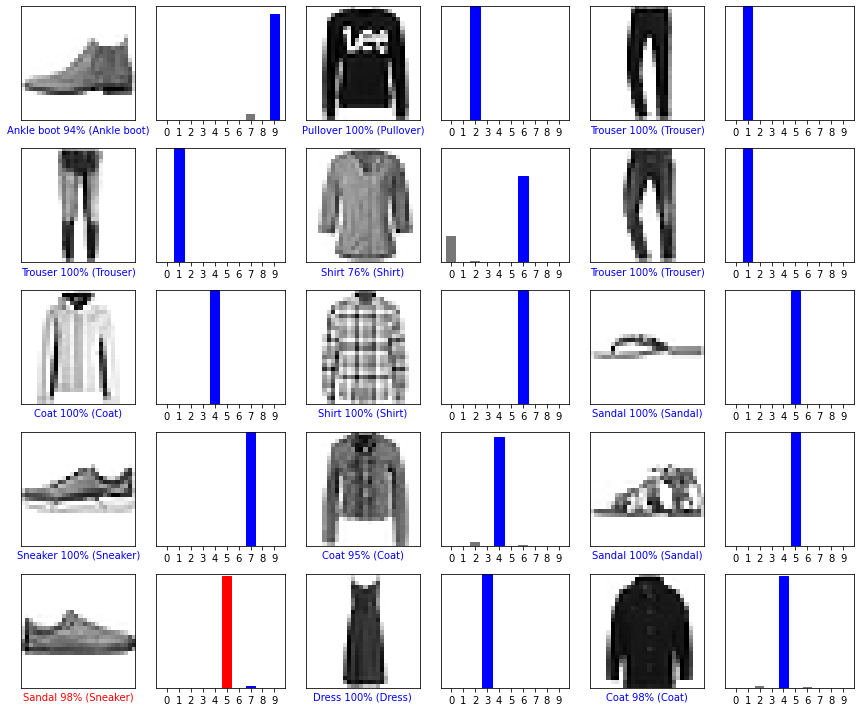
test label: 9

label for 12th image: 5

test label: 7





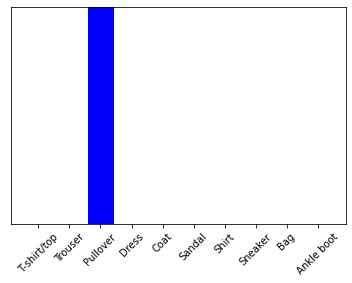


(28, 28)

(1, 28, 28)

[[1.9492356e-05 7.3653132e-15 9.9959320e-01 2.9512664e-12 2.0014345e-04

7.4366037e-14 1.8720150e-04 8.2591490e-23 1.0487984e-11 3.8671073e-21]]



2

**Practical No.10: Implement Regression to predict fuel efficiency using Tensorflow (Auto MPG dataset)**

**Code:**

# Use seaborn for pairplot.  
pip install -q seaborn

import matplotlib.pyplot as plt  
import numpy as np  
import pandas as pd  
import seaborn as sns  
  
# Make NumPy printouts easier to read.

np.set\_printoptions(precision=3, suppress=True)

import tensorflow as tf  
from tensorflow import keras  
from tensorflow.keras import layers  
print(tf.\_\_version\_\_)

#First download and import the dataset using pandas:

url = 'http://archive.ics.uci.edu/ml/machine-learning-databases/auto-mpg/auto-mpg.data'  
column\_names = ['MPG', 'Cylinders', 'Displacement', 'Horsepower', 'Weight',  
                'Acceleration', 'Model Year', 'Origin']  
  
raw\_dataset = pd.read\_csv(url, names=column\_names,  
                          na\_values='?', comment='\t',  
                          sep=' ', skipinitialspace=True)

dataset = raw\_dataset.copy()  
print(dataset.tail())

#The dataset contains a few unknown values:

print('\Check null values:', dataset.isna().sum())

dataset = dataset.dropna()

dataset['Origin'] = dataset['Origin'].map({1: 'USA', 2: 'Europe', 3: 'Japan'})

dataset = pd.get\_dummies(dataset, columns=['Origin'], prefix='', prefix\_sep='')  
print(dataset.tail())

#Split the data into training and test sets

train\_dataset = dataset.sample(frac=0.8, random\_state=0)  
test\_dataset = dataset.drop(train\_dataset.index)

#Inspect the data

sns.pairplot(train\_dataset[['MPG', 'Cylinders', 'Displacement', 'Weight']], diag\_kind='kde')

train\_dataset.describe().transpose()

### #Split features from labels

train\_features = train\_dataset.copy()  
test\_features = test\_dataset.copy()  
  
train\_labels = train\_features.pop('MPG')  
test\_labels = test\_features.pop('MPG')

train\_dataset.describe().transpose()[['mean', 'std']]

normalizer = tf.keras.layers.Normalization(axis=-1)

print('Calculate the mean and variance, and store them in the layer\n')

normalizer.adapt(np.array(train\_features))

print(normalizer.mean.numpy())

first = np.array(train\_features[:1])  
  
with np.printoptions(precision=2, suppress=True):  
  print('First example:', first)  
  print()  
  print('Normalized:', normalizer(first).numpy())

#This model will predict 'MPG' from 'Horsepower'.

horsepower\_model = tf.keras.Sequential([  
    horsepower\_normalizer,  
    layers.Dense(units=1)  
])  
  
horsepower\_model.summary()

horsepower\_model.predict(horsepower[:10])

horsepower\_model.compile(  
    optimizer=tf.optimizers.Adam(learning\_rate=0.1),  
    loss='mean\_absolute\_error')

%%time  
history = horsepower\_model.fit(  
    train\_features['Horsepower'],  
    train\_labels,  
    epochs=100,  
    # Suppress logging.  
    verbose=0,  
    # Calculate validation results on 20% of the training data.  
    validation\_split = 0.2)

hist = pd.DataFrame(history.history)  
hist['epoch'] = history.epoch  
hist.tail()

def plot\_loss(history):  
  plt.plot(history.history['loss'], label='loss')  
  plt.plot(history.history['val\_loss'], label='val\_loss')  
  plt.ylim([0, 10])  
  plt.xlabel('Epoch')  
  plt.ylabel('Error [MPG]')  
  plt.legend()  
  plt.grid(True)

plot\_loss(history)

test\_results = {}  
  
test\_results['horsepower\_model'] = horsepower\_model.evaluate(  
    test\_features['Horsepower'],  
    test\_labels, verbose=0)

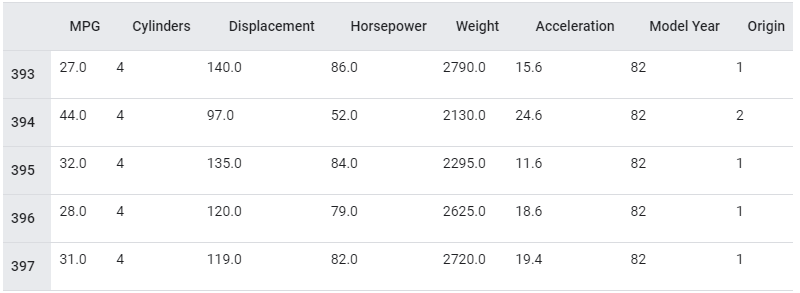
x = tf.linspace(0.0, 250, 251)  
y = horsepower\_model.predict(x)

def plot\_horsepower(x, y):  
  plt.scatter(train\_features['Horsepower'], train\_labels, label='Data')  
  plt.plot(x, y, color='k', label='Predictions')  
  plt.xlabel('Horsepower')  
  plt.ylabel('MPG')  
  plt.legend()

plot\_horsepower(x, y)

**Output:**

2.8.0



Check null values

MPG 0

Cylinders 0

Displacement 0

Horsepower 6

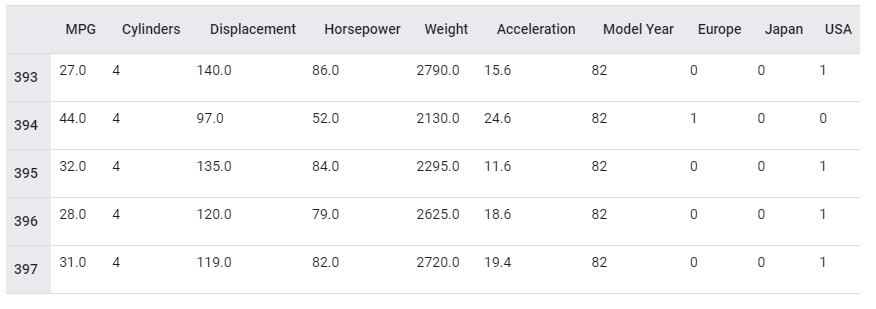
Weight 0

Acceleration 0

Model Year 0

Origin 0

dtype: int64



Calculate the mean and variance, and store them in the layer

[[ 5.478 195.318 104.869 2990.252 15.559 75.898 0.178 0.197 0.624]]

First example: [[ 4. 90. 75. 2125. 14.5 74. 0. 0. 1. ]]

Normalized: [[-0.87 -1.01 -0.79 -1.03 -0.38 -0.52 -0.47 -0.5 0.78]]

Model: "sequential"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

normalization\_1 (Normalizat (None, 1) 3

ion)

dense (Dense) (None, 1) 2

=================================================================

Total params: 5

Trainable params: 2

Non-trainable params: 3

array([[-0.571],

[-0.322],

[ 1.054],

[-0.8 ],

[-0.724],

[-0.284],

[-0.858],

[-0.724],

[-0.189],

[-0.322]], dtype=float32)

CPU times: user 4.88 s, sys: 809 ms, total: 5.69 s

Wall time: 3.82 s

