1\_EXPERIMENT

December 1, 2024

1\_EXPERIMENT: BASICS OF PYTHON

[ ]:

**def**

bitwise\_operations

():

a

=

int

(

input

(

"

enter the first number:

"

))

b

=

int

(

input

(

"

enter the second number:

"

))

print

(

f

"

**{**

a

**}**

&

**{**

b

**}**

=

**{**

a

&

b

**}**

"

)

print

(

f

"

**{**

a

**}**

|

**{**

b

**}**

=

**{**

a

|

b

**}**

"

)

print

(

f

"

**{**

a

**}**

^

**{**

b

**}**

=

**{**

a

^

b

**}**

"

)

bitwise\_operations()

enter the first number:5

enter the second number:8

5&8=0

5|8=13

5^8=13

[ ]: **def** sum\_special\_number():

total=sum(i **for** i **in** range(1,21)**if** i% 2 !=0 **and** i%3!=0 **and** i%5!=0) print("sum of numbers from 1 to 20 that are not divisible by 2,3,or5 is:

↪",total) sum\_special\_number() sum of numbers from 1 to 20 that are not divisible by 2,3,or5 is: 68

[ ]: **def** find\_max(a,b):

**return** a **if** a>b **else** b

num1=int(input("Enter the first number:")) num2=int(input("Enter the second number:")) print("The maximum of the two numbers is:",find\_max(num1,num2))

Enter the first number:6

Enter the second number:7

The maximum of the two numbers is: 7

[ ]:

**def**

slicing\_examples

():

text

=

"

Machinelearning

"

print

(

"

original string:

"

,text)

print

(

"

sliced string(0:7):text[0:7]

"

)

print

(

"

sliced string (-5:):

"

,text[

-

5

:])

1

items

=

list

(

range

(

1

,

11

))

print

(

"

original list:

"

,items)

print

(

"

original list:

"

,items)

print

(

"

sliced list(0:5):

"

,items[

0

:

5

])

print

(

"

sliced list (5:):

"

,items[

5

:])

slicing\_examples()

original string: Machinelearning

sliced string(0:7):text[0:7]

sliced string (-5:): rning original list: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] original list: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] sliced list(0:5): [1, 2, 3, 4, 5] sliced list (5:): [6, 7, 8, 9, 10]

2

# 2\_EXPERIMENT

December 1, 2024

2\_EPERIMENT: DATAFRAME AND DATA ANALYSIS

[ ]:

**import**

**pandas**

**as**

**pd**

data

=

pd

.

read\_csv(

'

/content/exams.csv

'

)

print

(

data

.

head())

gender race/ethnicity parental level of education lunch \

|  |  |  |
| --- | --- | --- |
| 0 male | group A | high school standard |
| 1 female | group D | some high school free/reduced |
| 2 male | group E | some college free/reduced |
| 3 male | group B | high school standard |
| 4 male | group E | associate's degree standard |

test preparation course math score reading score writing score

|  |  |  |  |
| --- | --- | --- | --- |
| 0 completed | 67 | 67 | 63 |
| 1 none | 40 | 59 | 55 |
| 2 none | 59 | 60 | 50 |
| 3 none | 77 | 78 | 68 |
| 4 completed | 78 | 73 | 68 |

[ ]:

print

(

data

.

describe())

math score reading score writing score

|  |  |  |  |
| --- | --- | --- | --- |
| count 1000.000000 | | 1000.000000 | 1000.000000 |
| mean | 66.396000 | 69.002000 | 67.738000 |
| std | 15.402871 | 14.737272 | 15.600985 |
| min | 13.000000 | 27.000000 | 23.000000 |
| 25% | 56.000000 | 60.000000 | 58.000000 |
| 50% | 66.500000 | 70.000000 | 68.000000 |
| 75% | 77.000000 | 79.000000 | 79.000000 |
| max | 100.000000 | 100.000000 | 100.000000 |

[ ]:

filtered\_data

=

data[data[

'

reading score

'

]

>

73

]

print

(

data

.

isnull()

.

sum())

|  |  |
| --- | --- |
| gender | 0 |
| race/ethnicity | 0 |
| parental level of education | 0 |
| lunch | 0 |
| test preparation course | 0 |
| math score | 0 |
| reading score | 0 |
| writing score dtype: int64 | 0 |

[ ]:

filtered\_data

=

data[data[

'

math score

'

]

>

80

]

print

(

filtered\_data

)

|  |  |  |  |
| --- | --- | --- | --- |
| gender race/ethnicity parental level of education | | | lunch \ |
| 7 male | group E | some college | standard |
| 10 male | group E | some college | standard |
| 14 male | group E | some high school | standard |
| 20 female | group C | associate's degree | standard |
| 30 male | group B | some college | standard |
| .. … | … | … | … |
| 970 female | group D | some high school | standard |
| 972 female | group E | associate's degree | standard |
| 991 female | group C | associate's degree | standard |
| 994 male | group E | high school free/reduced | |
| 996 male | group D | associate's degree free/reduced | |

test preparation course math score reading score writing score

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 7 | completed |  | 93 |  | 88 |  | 84 |
| 10 | completed |  | 99 |  | 83 |  | 85 |
| 14 | completed |  | 81 |  | 87 |  | 85 |
| 20 | none |  | 83 |  | 76 |  | 88 |
| 30 | none |  | 87 |  | 79 |  | 77 |
| .. | … | … |  | … |  | … |  |
| 970 | completed |  | 85 |  | 100 |  | 97 |
| 972 | none |  | 81 |  | 77 |  | 74 |
| 991 | none |  | 87 |  | 93 |  | 88 |
| 994 | completed |  | 86 |  | 82 |  | 75 |
| 996 | completed |  | 85 |  | 91 |  | 92 |
| [188 rows x 8 columns] | |

[ ]: grouped\_data=data.groupby('writing score').mean(numeric\_only=**True**) print(grouped\_data)

math score reading score

writing score

|  |  |  |  |
| --- | --- | --- | --- |
| 23 | 25.000000 | | 28.500000 |
| 24 | 43.000000 | | 28.000000 |
| 26 | 36.000000 | | 27.000000 |
| 27 | 37.000000 | | 34.500000 |
| 28 | 31.000000 | | 34.000000 |
| … | … | | … |
| 96 | 88.666667 | | 95.000000 |
| 97 | 86.888889 | | 95.444444 |
| 98 92.000000 | | 100.000000 | |
| 99 86.000000 | | 98.666667 | |
| 100 93.277778 | | 98.555556 | |

[76 rows x 2 columns]

[ ]:

sorted\_data

=

data

.

sort\_values(by

=

'

lunch

'

,ascending

=

**False**

)

print

(

sorted\_data

)

|  |  |  |
| --- | --- | --- |
| gender race/ethnicity parental level of education lunch \ | | |
| 0 male | group A | high school standard |
| 538 female | group D | high school standard |
| 585 male | group C | high school standard |
| 586 female | group B | master's degree standard |
| 587 female | group B | some college standard |
| .. … | … | … … |
| 227 female | group D | bachelor's degree free/reduced |
| 229 male | group C | associate's degree free/reduced |
| 636 male | group E | bachelor's degree free/reduced |
| 232 male | group C | some college free/reduced |
| 500 female | group B | high school free/reduced |

test preparation course math score reading score writing score

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | completed |  | 67 |  | 67 |  | 63 |
| 538 | none |  | 66 |  | 73 |  | 72 |
| 585 | completed |  | 53 |  | 59 |  | 64 |
| 586 | completed |  | 59 |  | 73 |  | 74 |
| 587 | none |  | 78 |  | 96 |  | 89 |
| .. | … | … |  | … |  | … |  |
| 227 | none |  | 61 |  | 71 |  | 73 |
| 229 | completed |  | 94 |  | 83 |  | 82 |
| 636 | none |  | 63 |  | 56 |  | 56 |
| 232 | none |  | 35 |  | 44 |  | 33 |
| 500 | none |  | 53 |  | 64 |  | 63 |

[1000 rows x 8 columns]

[ ]: other\_data=pd.read\_csv('/content/Original\_data\_with\_more\_rows.csv') merged\_data=pd.merge(data,other\_data,on='ReadingScore') print(merged\_data)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Unnamed: 0\_x Gender\_x EthnicGroup\_x | | ParentEduc\_x LunchType\_x \ | |
| 0 | 0 female | group B | bachelor's degree | standard |
| 1 | 0 female | group B | bachelor's degree | standard |
| 2 | 0 female | group B | bachelor's degree | standard |
| 3 | 0 female | group B | bachelor's degree | standard |
| 4 | 0 female | group B | bachelor's degree | standard |
| … | … … | … | … … |  |
| 17829296 | 999 female | group D associate's degree | | standard |

1. 999 female group D associate's degree standard
2. 999 female group D associate's degree standard
3. 999 female group D associate's degree standard
4. 999 female group D associate's degree standard

TestPrep\_x MathScore\_x ReadingScore WritingScore\_x Unnamed: 0\_y \

0 none 72 72 74 0 1 none 72 72 74 13

1. none 72 72 74 32
2. none 72 72 74 67
3. none 72 72 74 90

… … … … … …

1. none 52 68 66 673
2. none 52 68 66 684
3. none 52 68 66 707
4. none 52 68 66 735
5. none 52 68 66 999

Gender\_y EthnicGroup\_y ParentEduc\_y LunchType\_y TestPrep\_y \

0 female group B bachelor's degree standard none 1 male group A some college standard completed 2 female group E master's degree free/reduced none

1. female group C some college standard none
2. female group C bachelor's degree standard none

… … … … … …

1. female group B associate's degree free/reduced none
2. female group C some college free/reduced none
3. male group C some college standard none
4. female group B high school free/reduced completed
5. female group D associate's degree standard none

MathScore\_y WritingScore\_y

1. 72 74
2. 78 70
3. 56 65
4. 60 74
5. 65 74

… … …

1. 53 63
2. 54 67
3. 70 74
4. 50 71
5. 52 66

[17829301 rows x 17 columns]

# 3.1\_EXPERIMENT

December 1, 2024

3\_EXPERIMENT: DATA VISUALIZATION

[ ]: **import matplotlib.pyplot as plt import seaborn as sns import pandas as pd** *# Load the iris dataset* iris = sns.load\_dataset('iris')

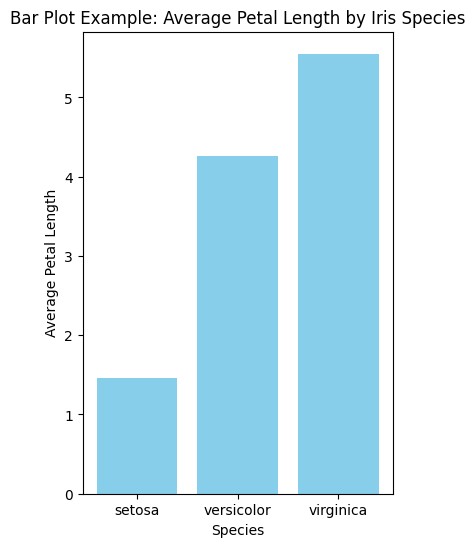
*# Prepare data: Calculate the average petal length for each species*

avg\_petal\_length = iris.groupby('species')['petal\_length'].mean()

*# Create a bar plot*

plt.figure(figsize=(4,6)) plt.bar(avg\_petal\_length.index, avg\_petal\_length.values, color="skyblue")

*# Add labels and title* plt.xlabel('Species') plt.ylabel('Average Petal Length') plt.title('Bar Plot Example: Average Petal Length by Iris Species') plt.show()



2

3.2\_EXPERIMENT

December 1, 2024

3.2: HISTOGRAM

[ ]:

**from**

**sklearn**

**.**

**datasets**

**import**

fetch\_california\_housing

**import**

**matplotlib**

**.**

**pyplot**

**as**

**plt**

*# Load the California housing dataset*

california\_data

=

fetch\_california\_housing(as\_frame

=

**True**

)

df

=

california\_data[

'

data

'

]

*# Select a feature, e.g., 'MedInc' (Median Income)*

median\_income

=

df[

'

MedInc

'

]

*# Plot a histogram*

plt

.

figure(figsize

=

(

6

,

4

))

plt

.

hist(median\_income, bins

=

10

, color

=

'

green

'

, edgecolor

=

'

black

'

)

plt

.

xlabel(

'

Median Income

'

)

plt

.

ylabel(

'

Frequency

'

)

plt

.

title(

'

Histogram Example: Median Income Distribution

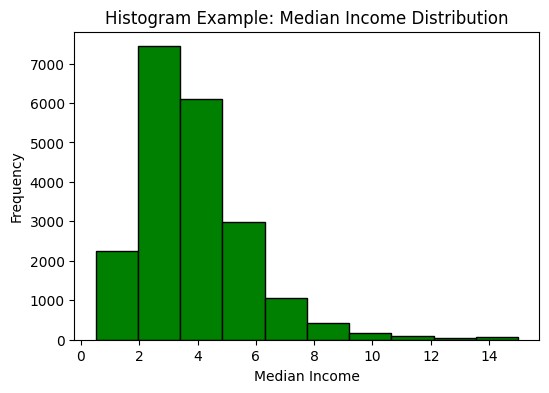
'

)

plt

.

show()



2

3.3\_EXPERIMENT

December 1, 2024

3.3: DISTIBUTION PLOT

[ ]:

**from**

**sklearn**

**.**

**datasets**

**import**

load\_wine

**import**

**seaborn**

**as**

**sns**

**import**

**pandas**

**as**

**pd**

**import**

**matplotlib**

**.**

**pyplot**

**as**

**plt**

*# Load the Wine dataset*

wine\_data

=

load\_wine(as\_frame

=

**True**

)

df

=

wine\_data[

'

data

'

]

*# Select a feature, e.g., 'alcohol'*

alcohol\_content

=

df[

'

alcohol

'

]

*# Plot a distribution plot using Seaborn*

sns

.

set(style

=

'

white

'

)

plt

.

figure(figsize

=

(

6

,

4

))

sns

.

histplot(alcohol\_content, kde

=

**False**

, color

=

'

yellow

'

, bins

=

10

)

*# Add labels and title*

plt

.

xlabel(

'

Alcohol Content

'

)

plt

.

title(

'

Distribution Plot Example: Alcohol Content in Wine

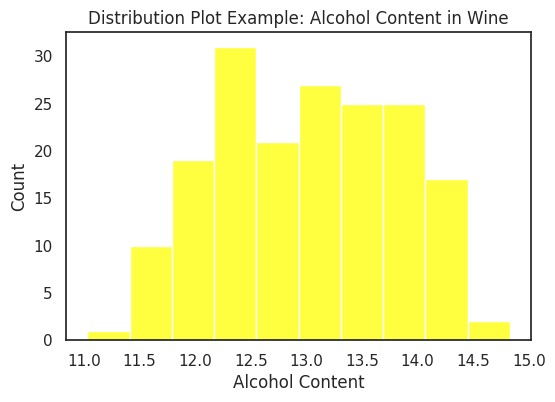
'

)

plt

.

show()



# 3.4\_EXPERIMENT

December 1, 2024

3.4:

BOX PLOT

[ ]:

**from**

**sklearn**

**.**

**datasets**

**import**

load\_diabetes

**import**

**matplotlib**

**.**

**pyplot**

**as**

**plt**

*# Load the Diabetes dataset*

diabetes

=

load\_diabetes(as\_frame

=

**True**

)

df

=

diabetes[

'

data

'

]

target

=

diabetes[

'

target

'

]

*# Select BMI and the target*

x

=

df[

'

bmi

'

]

y

=

target

*# Box plot example*

plt

.

figure(figsize

=

(

8

,

5

))

plt

.

boxplot(x, vert

=

**False**

, patch\_artist

=

**True**

,

␣

↪

boxprops

=

dict

(

facecolor

=

'

lightblue

'

, color

=

'

blue

'

)

,

␣

↪

medianprops

=

dict

color

(

=

'

red

'

))

plt

.

xlabel(

'

BMI

'

)

plt

.

title(

'

Box Plot Example: Distribution of BMI

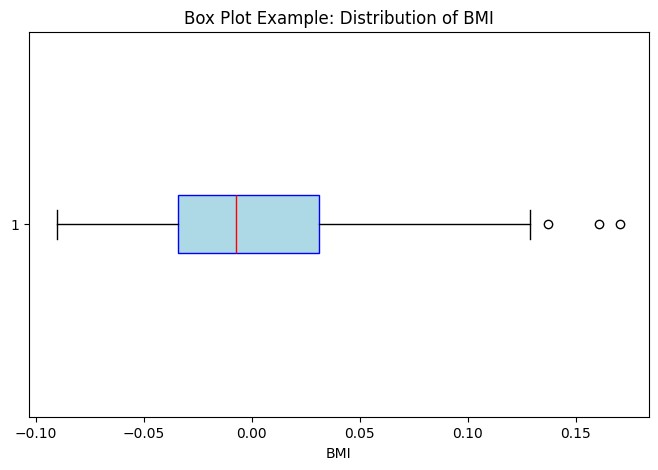
'

)

plt

.

show()



3.5\_EXPERIMENT

December 1, 2024

3.5: SCATTER PLOT

[ ]:

**from**

**sklearn**

**.**

**datasets**

**import**

load\_diabetes

**import**

**matplotlib**

**.**

**pyplot**

**as**

**plt**

*# Load the Diabetes dataset*

diabetes

=

load\_diabetes(as\_frame

=

**True**

)

df

=

diabetes[

'

data

'

]

target

=

diabetes[

'

target

'

]

*# Select two variables: BMI and the target*

x

=

df[

'

bmi

'

]

y

=

target

*# Scatter plot example*

plt

.

figure(figsize

=

(

8

,

5

))

plt

.

scatter(x, y, color

=

'

blue

'

, edgecolor

=

'

k

'

, alpha

=

0.7

)

*# Add labels and title*

plt

.

xlabel(

'

BMI

'

)

plt

.

ylabel(

'

Disease Progression

'

)

plt

.

title(

'

Scatter Plot Example: BMI vs Disease Progression

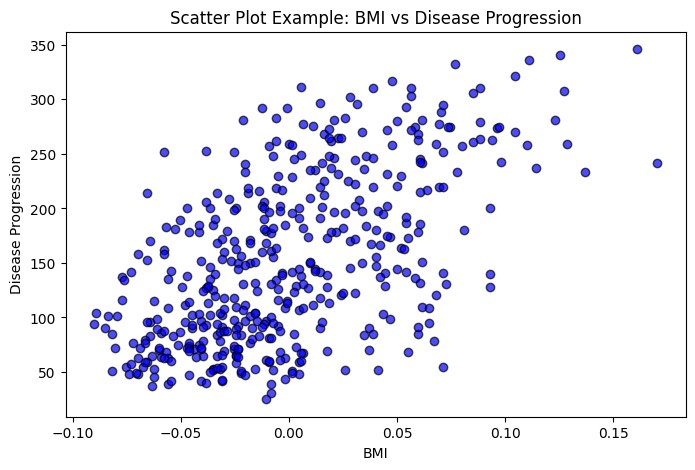
'

)

plt

.

show()



4\_EXPERIMENT

December 1, 2024

4\_EXP: SIMPLE LINEAR REGRESSION

[ ]:

**import**

**numpy**

**as**

**np**

**import**

**matplotlib**

**.**

**pyplot**

**as**

**plt**

**from**

**sklearn**

**.**

**linear\_model**

**import**

LinearRegression

*# Create synthetic data with a clear linear relationship*

X

=

np

.

array([

1

,

2

,

3

,

4

,

5

])

.

reshape(

-

1

,

1

)

*# X values*

y

=

2

\*

X

+

1

*# Linear relationship: y = 2x + 1*

*# Fit the Linear Regression model*

model

=

LinearRegression()

model

.

fit(X, y)

*# Predict*

y\_pred

=

model

.

predict(X)

*# Plot*

plt

.

figure(figsize

=

(

8

,

5

))

plt

.

scatter(X, y, color

=

'

black

'

, label

=

'

Actual Data

'

)

plt

.

plot(X, y\_pred, color

=

'

blue

'

, label

=

'

Regression Line

'

)

*# Add labels and title*

plt

.

xlabel(

'

X

'

)

plt

.

ylabel(

'

y

'

)

plt

.

title(

'

Simple Linear Regression

'

)

plt

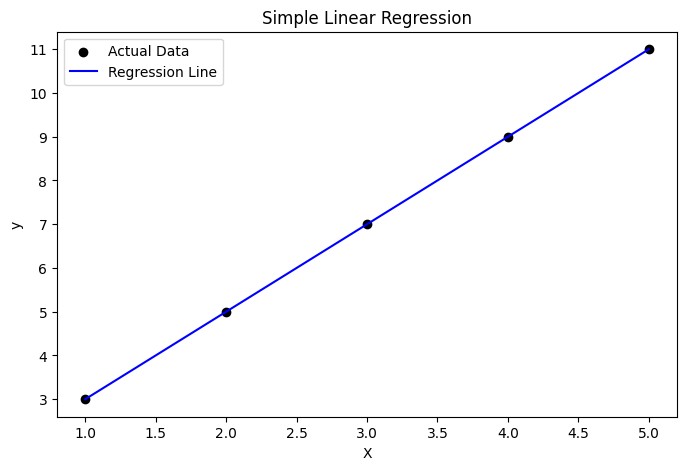
.

legend()

plt

.

show()



# 5\_EXPERIMENT

December 1, 2024

5\_EXP: MUNLTIPLE LINEAR REGRESSION

[ ]:

**import**

**numpy**

**as**

**np**

**import**

**pandas**

**as**

**pd**

**from**

**sklearn**

**.**

**linear\_model**

**import**

LinearRegression

**from**

**sklearn**

**.**

**datasets**

**import**

load\_linnerud

*# Load the Linnerud dataset*

linnerud

=

load\_linnerud()

X

=

linnerud

.

data

*# Features (Exercise1, Exercise2, Exercise3)*

y

=

linnerud

.

target

*# Target (Weight)*

*# Fit the model*

model

=

LinearRegression()

model

.

fit(X, y)

*# Predict*

y\_pred

=

model

.

predict(X)

*# Output predicted values*

print

(

"

Predicted values:

"

, y\_pred)

Predicted values: [[176.17362115 35.05740701 57.09006881]

[188.91995665 37.56552215 54.90132724]

[189.94576449 37.70676695 53.32586244]

[183.11724289 35.75511023 55.37949501]

[173.71124883 34.18885434 56.86335161]

[188.25376149 37.15128655 55.05550522]

[185.98130258 36.49191125 55.17763306]

[181.89233676 35.85351992 56.12527493]

[161.2883643 31.59644105 59.28711761]

[168.78332946 35.13998817 55.24591309]

[177.56945242 34.49368332 55.9858289 ]

[167.04288012 33.56483506 57.49568229]

[164.5483877 32.94654574 58.00151183]

[201.52708749 39.84285384 52.67310369]

[193.02894726 37.82027187 54.07880803]

[167.98334833 33.84157327 57.34730565]

[195.59763445 38.32953699 53.8331394 ]

[160.38051047 32.05276714 59.36526986]

[158.91736879 31.51115416 59.36563417]

[187.33745435 37.08997099 55.40216714]]

# 6.2\_EXPERIMENT

December 1, 2024

6.2

\_EXP: DECISION TREE

[ ]:

**import**

**pandas**

**as**

**pd**

*# Import pandas*

**from**

**sklearn**

**.**

**model\_selection**

**import**

train\_test\_split

**from**

**sklearn**

**.**

**tree**

**import**

DecisionTreeClassifier

**from**

**sklearn**

**.**

**datasets**

**import**

load\_iris

*# Load dataset*

data

=

load\_iris()

*# Create DataFrame from the iris dataset*

df

=

pd

.

DataFrame(data

.

data, columns

=

data

.

feature\_names)

df[

'

target

'

]

=

data

.

target

print

(

df

.

head())

X

=

data

.

data

y

=

data

.

target

*# Split data into training and testing sets*

X\_train, X\_test, y\_train, y\_test

=

train\_test\_split(X, y, test\_size

=

0.11

,

␣

↪

random\_state

=

100

)

*# Fit Decision Tree model*

tree\_clf

=

DecisionTreeClassifier()

tree\_clf

.

fit(X\_train, y\_train)

*# Predict and evaluate*

accuracy

=

tree\_clf

.

score(X\_test, y\_test)

print

(

f

'

Accuracy:

**{**

accuracy

**:**

f

.4

**}**

'

)

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

1. 5.1 3.5 1.4 0.2
2. 4.9 3.0 1.4 0.2
3. 4.7 3.2 1.3 0.2
4. 4.6 3.1 1.5 0.2
5. 5.0 3.6 1.4 0.2

target

1. 0
2. 0
3. 0
4. 0
5. 0

Accuracy: 1.0000

2

# 6\_EXPERIMENT

December 1, 2024

6\_EXP: LOGISTIC REGRESSION AND DEICION TREE MODELS

[ ]: **from sklearn.model\_selection import** train\_test\_split **from sklearn.linear\_model import** LogisticRegression **from sklearn.datasets import** load\_digits

**import pandas as pd** *# Load the Digits dataset* data = load\_digits() df = pd.DataFrame(data.data, columns=data.feature\_names) df['target'] = data.target print(df.head())

*# Split the data into training and testing sets*

X\_train, X\_test, y\_train, y\_test = train\_test\_split(data.data, data.target,␣ ↪test\_size=0.1, random\_state=4)

*# Fit Logistic Regression model* log\_reg = LogisticRegression(max\_iter=100) log\_reg.fit(X\_train, y\_train) *# Predict and evaluate* accuracy = log\_reg.score(X\_test, y\_test) print("Logistic Regression Accuracy:", accuracy)

pixel\_0\_0 pixel\_0\_1 pixel\_0\_2 pixel\_0\_3 pixel\_0\_4 pixel\_0\_5 \

0 0.0 0.0 5.0 13.0 9.0 1.0 1 0.0 0.0 0.0 12.0 13.0 5.0 2 0.0 0.0 0.0 4.0 15.0 12.0 3 0.0 0.0 7.0 15.0 13.0 1.0 4 0.0 0.0 0.0 1.0 11.0 0.0

pixel\_0\_6 pixel\_0\_7 pixel\_1\_0 pixel\_1\_1 … pixel\_6\_7 pixel\_7\_0 \

0 0.0 0.0 0.0 0.0 … 0.0 0.0 1 0.0 0.0 0.0 0.0 … 0.0 0.0 2 0.0 0.0 0.0 0.0 … 0.0 0.0 3 0.0 0.0 0.0 8.0 … 0.0 0.0 4 0.0 0.0 0.0 0.0 … 0.0 0.0

pixel\_7\_1 pixel\_7\_2 pixel\_7\_3 pixel\_7\_4 pixel\_7\_5 pixel\_7\_6 \

1. 0.0 6.0 13.0 10.0 0.0 0.0
2. 0.0 0.0 11.0 16.0 10.0 0.0
3. 0.0 0.0 3.0 11.0 16.0 9.0
4. 0.0 7.0 13.0 13.0 9.0 0.0 4 0.0 0.0 2.0 16.0 4.0 0.0

pixel\_7\_7 target

0 0.0 0 1 0.0 1 2 0.0 2 3 0.0 3

4 0.0 4

[5 rows x 65 columns]

Logistic Regression Accuracy: 0.9555555555555556

/usr/local/lib/python3.10/dist-packages/sklearn/linear\_model/\_logistic.py:469:

ConvergenceWarning: lbfgs failed to converge (status=1): STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max\_iter) or scale the data as shown in: https://scikit-learn.org/stable/modules/preprocessing.html

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear\_model.html#logistic-

regression

n\_iter\_i = \_check\_optimize\_result(

2

7\_EXPERIMENT

December 1, 2024

7\_EXP: SPLITTING DATA AND MODEL EVALUATION

[ ]:

**import**

**numpy**

**as**

**np**

**from**

**sklearn**

**.**

**model\_selection**

**import**

train\_test\_split

**from**

**sklearn**

**.**

**linear\_model**

**import**

LinearRegression

**from**

**sklearn**

**.**

**metrics**

**import**

mean\_squared\_error

*# Sample data*

X

=

np

.

array([

1

,

2

,

3

,

4

,

5

])

.

reshape(

-

1

,

1

)

y

=

np

.

array([

1.2

,

1.9

,

2.6

,

3.8

,

4.1

])

*# Split data*

X\_train, X\_val, y\_train, y\_val

=

train\_test\_split(X, y, test\_size

=

0.2

,

␣

↪

random\_state

=

70

)

*# Train model*

model

=

LinearRegression()

model

.

fit(X\_train, y\_train)

*# Validate model*

y\_pred

=

model

.

predict(X\_val)

mse

=

mean\_squared\_error(y\_val, y\_pred)

print

(

"

Mean Squared Error on Validation Set:

"

, mse)

Mean Squared Error on Validation Set: 0.16000000000000028

# 8\_EXPERIMENT

December 1, 2024

8\_EXP: K-NEAREST NEIIGHBORS(K-NN ALGORITHM)

[ ]: **from sklearn.model\_selection import** train\_test\_split **from sklearn.neighbors import** KNeighborsClassifier **from sklearn.datasets import** load\_wine **from sklearn.metrics import** accuracy\_score

**import pandas as pd** *# Load the Wine dataset* data = load\_wine() df = pd.DataFrame(data.data, columns=data.feature\_names) df['target'] = data.target

print(df.head())

*# Split the data into training and testing sets*

X\_train, X\_test, y\_train, y\_test = train\_test\_split(data.data, data.target,␣ ↪test\_size=0.8, random\_state=70)

*# Fit k-NN model*

knn = KNeighborsClassifier(n\_neighbors=6) knn.fit(X\_train, y\_train) *# Predict and evaluate* y\_pred = knn.predict(X\_test) accuracy = accuracy\_score(y\_test, y\_pred) print("k-NN Accuracy:", accuracy)

alcohol malic\_acid ash alcalinity\_of\_ash magnesium total\_phenols \

1. 14.23 1.71 2.43 15.6 127.0 2.80
2. 13.20 1.78 2.14 11.2 100.0 2.65
3. 13.16 2.36 2.67 18.6 101.0 2.80
4. 14.37 1.95 2.50 16.8 113.0 3.85
5. 13.24 2.59 2.87 21.0 118.0 2.80

flavanoids nonflavanoid\_phenols proanthocyanins color\_intensity hue \

0 3.06 0.28 2.29 5.64 1.04 1 2.76 0.26 1.28 4.38 1.05 2 3.24 0.30 2.81 5.68 1.03 3 3.49 0.24 2.18 7.80 0.86

4 2.69 0.39 1.82 4.32 1.04

od280/od315\_of\_diluted\_wines proline target

0 3.92 1065.0 0

1

|  |  |  |
| --- | --- | --- |
| 1 3.40 | 1050.0 | 0 |
| 2 3.17 | 1185.0 | 0 |
| 3 3.45 | 1480.0 | 0 |
| 4 2.93  k-NN Accuracy: 0.6573426573426573 | 735.0 | 0 |

2

# 9\_EXPERIMENT

December 1, 2024

9\_EXP: SUPPOER VECTOR MACHINE(SVM)

[ ]: **from sklearn.model\_selection import** train\_test\_split **from sklearn.svm import** SVC **from sklearn.datasets import** fetch\_openml **from sklearn.metrics import** accuracy\_score

**import pandas as pd** *# Load the MNIST dataset*

data = fetch\_openml('mnist\_784', version=1) df = pd.DataFrame(data.data) df['target'] = data.target print(df.head())

*# Convert target variable to integer type* df['target'] = df['target'].astype(int)

*# Split the data into training and testing sets*

X\_train, X\_test, y\_train, y\_test = train\_test\_split(data.data, data.target,␣ ↪test\_size=0.3, random\_state=100)

*# Fit SVM model with linear kernel* svm\_clf = SVC(kernel='linear') svm\_clf.fit(X\_train, y\_train) *# Predict and evaluate* y\_pred = svm\_clf.predict(X\_test) accuracy = accuracy\_score(y\_test, y\_pred) print("SVM Accuracy:", accuracy)

1

# 10\_EXPERMENT

December 1, 2024

10.1 Implementing word cloud using IMDB Movie Review datset

[ ]: *# Import necessary libraries* **import pandas as pd from wordcloud import** WordCloud **import matplotlib.pyplot as plt from sklearn.feature\_extraction.text import** CountVectorizer **import numpy as np**

*# Helper function to handle file upload (useful for Google Colab)* **def** upload\_file\_if\_needed():

**import os**

*# Check if the file exists in the current directory* **if not** os.path.exists('IMDB-Movie-Data.csv'):

**try**:

**from google.colab import** files uploaded = files.upload() *# Prompt user to upload the file*

print("File uploaded successfully!")

**except ImportError**:

**raise FileNotFoundError**("IMDB-Movie-Data.csv not found. Upload or␣

↪place the file in the working directory.")

*# Step 1: Load IMDB data* upload\_file\_if\_needed() data\_path = 'IMDB-Movie-Data.csv' df = pd.read\_csv(data\_path)

*# Step 2: Extract movie descriptions* **if** 'Description' **not in** df.columns:

**raise ValueError**("The 'Description' column is missing from the dataset.")

descriptions = df['Description'].dropna().tolist() *# Drop missing descriptions*

*# Step 3: Compute word frequencies using CountVectorizer* vectorizer = CountVectorizer(stop\_words='english') count\_matrix = vectorizer.fit\_transform(descriptions) *# Fit and transform*␣

↪*descriptions* word\_counts = np.sum(count\_matrix.toarray(), axis=0) *# Sum counts across all*␣ ↪*documents*

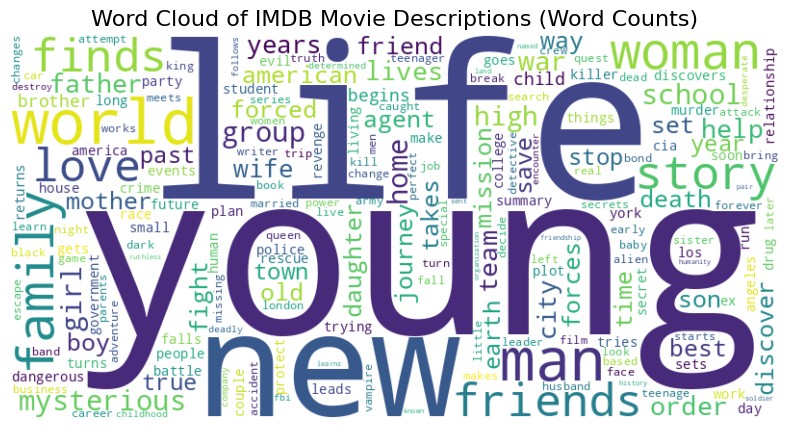
words = vectorizer.get\_feature\_names\_out() *# Get feature names (words)* word\_count\_dict = dict(zip(words, word\_counts)) *# Create a dictionary of words*␣ ↪*and their counts*

*# Step 4: Generate and display the word cloud* wordcloud = WordCloud(width=800, height=400, background\_color='white'). ↪generate\_from\_frequencies(word\_count\_dict)

plt.figure(figsize=(10, 5)) plt.imshow(wordcloud, interpolation='bilinear') plt.axis('off') plt.title("Word Cloud of IMDB Movie Descriptions (Word Counts)", fontsize=16) plt.show()

<IPython.core.display.HTML object>

Saving IMDB-Movie-Data.csv to IMDB-Movie-Data.csv File uploaded successfully!



10.2 Implementing TF\_IDf, stop wrods and Bag of Words using IMDB Movie Review Dataset

[8]:

**import**

**pandas**

**as**

**pd**

**from**

**wordcloud**

**import**

WordCloud

**import**

**matplotlib**

**.**

**pyplot**

**as**

**plt**

**from**

**sklearn**

**.**

**feature\_extraction**

**.**

**text**

**import**

CountVectorizer, TfidfVectorizer

**import**

**numpy**

**as**

**np**

*# Step 1: Load IMDB data*

data\_path = '/content/IMDB-Movie-Data.csv'

*# Try loading the dataset and handle errors* **try**:

df = pd.read\_csv('/content/IMDB-Movie-Data.csv') print("Dataset loaded successfully!")

**except FileNotFoundError**:

print(f"Error: File not found at '**{**data\_path**}**'. Please ensure the file␣

↪exists at this location.") **raise**

*# Step 2: Create corpus from the 'Description' column* **if** 'Description' **not in** df.columns:

**raise ValueError**("The 'Description' column is missing from the dataset.")

corpus = df['Description'].dropna().tolist() *# Drop missing values and convert*␣ ↪*to a list*

*# Step 3: Calculate TF-IDF scores* tfidf\_vectorizer = TfidfVectorizer(stop\_words='english') *# Remove English stop*␣

↪*words* tfidf\_matrix = tfidf\_vectorizer.fit\_transform(corpus) tfidf\_scores = np.sum(tfidf\_matrix.toarray(), axis=0) tfidf\_words = tfidf\_vectorizer.get\_feature\_names\_out() tfidf\_dict = dict(zip(tfidf\_words, tfidf\_scores)) *# Create a dictionary of*␣ ↪*words and their TF-IDF scores*

*# Step 4: Calculate word frequencies using Bag of Words (BoW)* bow\_vectorizer = CountVectorizer(stop\_words='english') bow\_matrix = bow\_vectorizer.fit\_transform(corpus) bow\_scores = np.sum(bow\_matrix.toarray(), axis=0) bow\_words = bow\_vectorizer.get\_feature\_names\_out() bow\_dict = dict(zip(bow\_words, bow\_scores)) *# Create a dictionary of words and*␣ ↪*their frequencies*

*# Step 5: Generate and display TF-IDF word cloud*

wordcloud\_tfidf = WordCloud(width=800, height=400, background\_color='white') \

.generate\_from\_frequencies(tfidf\_dict) plt.figure(figsize=(10, 5)) plt.title("TF-IDF Word Cloud") plt.imshow(wordcloud\_tfidf, interpolation='bilinear') plt.axis('off') plt.show()

*# Step 6: Generate and display Bag of Words word cloud*

wordcloud\_bow = WordCloud(width=800, height=400, background\_color='white') \

.generate\_from\_frequencies(bow\_dict) plt.figure(figsize=(10, 5))

plt

.

title(

"

Bag of Words Word Cloud

"

)

plt

.

imshow(wordcloud\_bow, interpolation

=

'

bilinear

'

)

plt

.

axis(

'

off

'

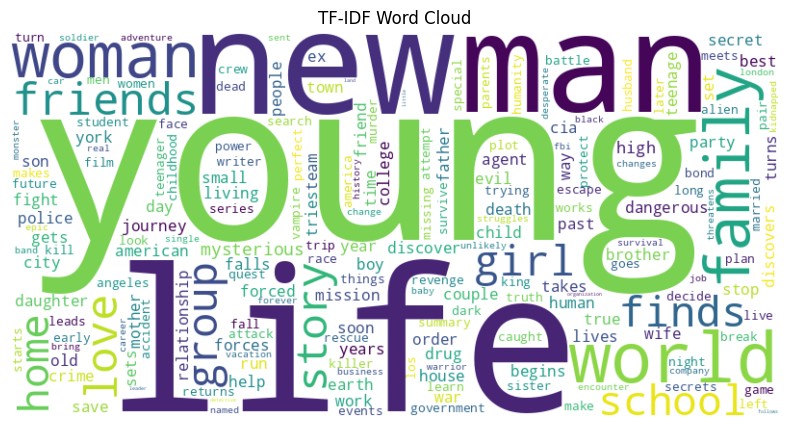
)

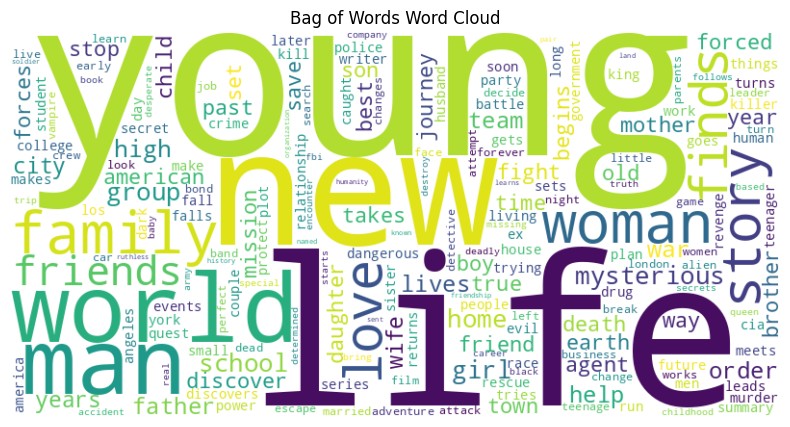
plt

.

show()

Dataset loaded successfully!





# 11.\_Find-S\_Algorithm

December 1, 2024

[1]: **import csv**

[2]: print("**\n** The most general hypothesis :['?','?','?','?','?','?']**\n**") print("**\n** The most specific hypothesis :['0','0','0','0','0','0']**\n**")

The most general hypothesis :['?','?','?','?','?','?']

The most specific hypothesis :['0','0','0','0','0','0']

[3]:

a

=

[]

[4]:

**with**

open

(

'

datasets/enjoysport.csv

'

,

'

r

'

)

**as**

csvFile:

reader

=

csv

.

reader(csvFile)

**for**

row

**in**

reader:

a

.

append(row)

print

(

row

)

csvFile

.

close()

num\_attributes

=

len

(

a

[

0

])

-

1

['sky', 'airTemp', 'humidity', 'wind', 'water', 'forecast', 'enjoySport']

['sunny', 'warm', 'normal', 'strong', 'warm', 'same', 'yes']

['sunny', 'warm', 'high', 'strong', 'warm', 'same', 'yes']

['rainy', 'cold', 'high', 'strong', 'warm', 'change', 'no']

['sunny', 'warm', 'high', 'strong', 'cool', 'change', 'yes']

[5]:

num\_attributes

[5]: 6

[6]: a

[6]: [['sky', 'airTemp', 'humidity', 'wind', 'water', 'forecast', 'enjoySport'],

['sunny', 'warm', 'normal', 'strong', 'warm', 'same', 'yes'],

['sunny', 'warm', 'high', 'strong', 'warm', 'same', 'yes'],

['rainy', 'cold', 'high', 'strong', 'warm', 'change', 'no'],

1

['sunny', 'warm', 'high', 'strong', 'cool', 'change', 'yes']]

[7]:

hypothesis

=

[

'

0

'

]

\*

num\_attributes

print

(

'

Initial Value of the hypothesis:

'

)

print

(

hypothesis

)

Initial Value of the hypothesis:

[

'0', '0', '0', '0', '0', '0'

]

[8]:

**for**

j

**in**

range

(

0

, num\_attributes):

hypothesis[j]

=

a[

1

][

j

]

[9]:

hypothesis

[9]: ['sunny', 'warm', 'normal', 'strong', 'warm', 'same']

[10]:

print

(

'

**\n**

Find S: Finding Maximally specific hypothesis

**\n**

'

)

**for**

i

**in**

range

(

1

,

len

(a)):

**if**

a[i][num\_attributes]

==

'

yes

'

:

**for**

j

**in**

range

(

num\_attributes

):

**if**

a[i][j]

!=

hypothesis[j]:

hypothesis[j]

=

'

?

'

**else**

:

hypothesis[j]

=

a[i][j]

print

(

"

For training example no :

**{0}**

the hypothesis is

"

.

format(

i), hypothesis)

Find S: Finding Maximally specific hypothesis

For training example no :1 the hypothesis is ['sunny', 'warm', 'normal', 'strong', 'warm', 'same']

For training example no :2 the hypothesis is ['sunny', 'warm', '?', 'strong',

'warm', 'same']

For training example no :3 the hypothesis is ['sunny', 'warm', '?', 'strong',

'warm', 'same']

For training example no :4 the hypothesis is ['sunny', 'warm', '?', 'strong', '?', '?']

[11]: print("**\n** The Maximally Specific Hypothesis for a given Training Examples:**\n**") print(hypothesis)

The Maximally Specific Hypothesis for a given Training Examples: ['sunny', 'warm', '?', 'strong', '?', '?']

2

# 12.\_Candidate\_Elimination

December 1, 2024



# 13.\_ID3\_\_Decistion\_Tree\_

December 1, 2024

[1]:

**import**

**csv**

**import**

**math**

[2]:

**def**

major\_class

attributes, data, target

):

(

freq

=

{}

index

=

attributes

.

index(target)

**for**

t

**in**

data:

**if**

t[index]

**in**

freq:

freq[t[index]]

+

=

1

**else**

:

freq[t[index]]

=

1

m

=

0

major

=

"

"

**for**

key

**in**

freq

.

keys():

**if**

freq[key]

>

m:

m

=

freq[key]

major

=

key

**return**

major

[3]:

**def**

entropy

(

attributes, data, targetAttr

):

freq

=

{}

data\_entropy

=

0.0

i

=

0

**for**

entry

**in**

attributes:

**if**

targetAttr

==

entry:

**break**

i

+

=

1

**for**

entry

**in**

data:

**if**

entry[i]

==

'

PlayTennis

'

:

**pass**

**else**

:

**if**

entry[i]

**in**

freq:

freq[entry[i]]

+

=

1.0

**else**

:

freq[entry[i]]

=

1.0

**for**

f

**in**

freq

.

values():

data\_entropy

+

=

(

-

f

/

len

(

data

))

\*

math

.

log(f

/

len

(

data),

2

)

**return**

data\_entropy

[4]:

**def**

info\_gain

(

attributes, data, attr, targetAttr

):

freq

=

{}

subset\_entropy

=

0.0

i

=

attributes

.

index(attr)

**for**

entry

**in**

data:

**if**

entry[i]

==

attr:

**pass**

**else**

:

**if**

entry[i]

**in**

freq:

freq[entry[i]]

+

=

1.0

**else**

:

freq[entry[i]]

=

1

**for**

val

**in**

freq

.

keys():

p

=

sum

(

freq

.

values())

val\_prob

=

freq[val]

/

(

p

)

data\_subset

=

[

entry

**for**

entry

**in**

data

**if**

entry[i]

==

val]

subset\_entropy

+

=

val\_prob

\*

entropy(attributes, data\_subset,

␣

↪

targetAttr)

data\_subset

=

[

entry

**for**

entry

**in**

data

**if**

entry[

0

]

!=

'

Outlook

'

]

**return**

entropy(attributes, data\_subset, targetAttr

(

)

-

subset\_entropy)

[5]:

**def**

attr\_choose

(

data, attributes, target

):

best

=

attributes[

0

]

max\_gain

=

0

**for**

attr

**in**

attributes:

**if**

attr

!=

target:

new\_gain

=

info\_gain(attributes, data, attr, target)

**if**

new\_gain

>

max\_gain:

max\_gain

=

new\_gain

best

=

attr

**return**

best

[6]:

**def**

get\_values

(

data, attributes, attr

):

i

=

attributes

.

index(attr)

values

=

[]

**for**

entry

**in**

data:

**if**

entry[i]

==

attr:

**pass**

**else**

:

**if**

entry[i]

**not**

**in**

values:

values

.

append(entry[i])

**return**

values

[7]:

**def**

get\_data

(

data, attributes, best, val

):

new\_data

=

[[]]

i

=

attributes

.

index(best)

**for**

entry

**in**

data:

**if**

entry[i]

==

val:

new\_entry

=

[]

**for**

j

**in**

range

(

len

(

entry

)):

**if**

j

!=

i:

new\_entry

.

append(entry[j])

new\_data

.

append(new\_entry)

new\_data

.

remove([])

**return**

new\_data

[8]:

**def**

build\_tree

(

):

data, attributes, target

print

)

(

data

print

attributes

(

)

print

(

target

)

data

=

data[:]

print

(

attributes

.

index(target))

vals

=

[

record[attributes

.

index(target)]

**for**

record

**in**

data]

print

(

vals

)

default

=

major\_class(attributes, data, target)

**if**

**not**

data

**or**

(

len

(

attributes

)

-

1

)

<

=

0

:

**return**

default

**elif**

vals

.

count(vals[

0

])

==

len

(

vals

):

**return**

vals[

0

]

**else**

:

best

=

attr\_choose(data, attributes, target)

tree

=

best

{

: {}}

**for**

val

**in**

get\_values(data, attributes, best):

new\_data

=

get\_data(data, attributes, best, val)

new\_attr

=

attributes[:]

new\_attr

.

remove(best)

subtree

=

build\_tree(new\_data, new\_attr, target)

tree[best][val]

=

subtree

**return**

tree

[9]:

**def**

test

(

attributes, instance, tree

):

attribute

=

next

(

iter

tree

(

))

i

=

attributes

.

index(attribute)

**if**

instance[i]

**in**

tree[attribute]

.

keys():

result

=

tree[attribute][instance[i]]

**if**

isinstance

(

result,

dict

):

**return**

test(attributes, instance, result)

**else**

:

**return**

result

**else**

:

**return**

'

NULL

'

[10]:

**def**

execute\_decision\_tree

():

data

=

[]

**with**

open

(

'

datasets/PlayTennis.csv

'

)

**as**

tsv:

**for**

line

**in**

csv

.

reader(tsv):

data

.

append(

tuple

(

line

))

print

(

'

Number of records:

'

,

len

(

data

))

attributes

=

[

'

Outlook

'

,

'

Temperature

'

,

'

Humidity

'

,

'

Wind

'

,

'

PlayTennis

'

]

target

=

attributes[

-

1

]

training\_set

=

[

x

**for**

i, x

**in**

enumerate

)]

(

data

print

(

training\_set

)

tree

=

build\_tree(training\_set, attributes, target)

print

(

'

Decision Tree is as below:

**\n**

'

)

print

(

tree

)

instance

=

[

'

Rain

'

,

'

Mild

'

,

'

High

'

,

'

Strong

'

]

print

(

'

Testing instance is:

'

, instance)

result

=

test(attributes, instance, tree)

print

(

'

The Target value for the testing instance is:

'

)

print

(

result

)

[11]:

execute\_decision\_tree()

Number of records: 15

[('', 'PlayTennis', 'Outlook', 'Temperature', 'Humidity', 'Wind'), ('0', 'No',

'Sunny', 'Hot', 'High', 'Weak'), ('1', 'No', 'Sunny', 'Hot', 'High', 'Strong'),

('2', 'Yes', 'Overcast', 'Hot', 'High', 'Weak'), ('3', 'Yes', 'Rain', 'Mild',

'High', 'Weak'), ('4', 'Yes', 'Rain', 'Cool', 'Normal', 'Weak'), ('5', 'No',

'Rain', 'Cool', 'Normal', 'Strong'), ('6', 'Yes', 'Overcast', 'Cool', 'Normal',

'Strong'), ('7', 'No', 'Sunny', 'Mild', 'High', 'Weak'), ('8', 'Yes', 'Sunny',

'Cool', 'Normal', 'Weak'), ('9', 'Yes', 'Rain', 'Mild', 'Normal', 'Weak'),

('10', 'Yes', 'Sunny', 'Mild', 'Normal', 'Strong'), ('11', 'Yes', 'Overcast',

'Mild', 'High', 'Strong'), ('12', 'Yes', 'Overcast', 'Hot', 'Normal', 'Weak'),

('13', 'No', 'Rain', 'Mild', 'High', 'Strong')]

[('', 'PlayTennis', 'Outlook', 'Temperature', 'Humidity', 'Wind'), ('0', 'No',

'Sunny', 'Hot', 'High', 'Weak'), ('1', 'No', 'Sunny', 'Hot', 'High', 'Strong'),

('2', 'Yes', 'Overcast', 'Hot', 'High', 'Weak'), ('3', 'Yes', 'Rain', 'Mild',

'High', 'Weak'), ('4', 'Yes', 'Rain', 'Cool', 'Normal', 'Weak'), ('5', 'No',

'Rain', 'Cool', 'Normal', 'Strong'), ('6', 'Yes', 'Overcast', 'Cool', 'Normal',

'Strong'), ('7', 'No', 'Sunny', 'Mild', 'High', 'Weak'), ('8', 'Yes', 'Sunny',

'Cool', 'Normal', 'Weak'), ('9', 'Yes', 'Rain', 'Mild', 'Normal', 'Weak'),

('10', 'Yes', 'Sunny', 'Mild', 'Normal', 'Strong'), ('11', 'Yes', 'Overcast',

'Mild', 'High', 'Strong'), ('12', 'Yes', 'Overcast', 'Hot', 'Normal', 'Weak'),

('13', 'No', 'Rain', 'Mild', 'High', 'Strong')]

['Outlook', 'Temperature', 'Humidity', 'Wind', 'PlayTennis']

PlayTennis

4

['Humidity', 'High', 'High', 'High', 'High', 'Normal', 'Normal', 'Normal',

'High', 'Normal', 'Normal', 'Normal', 'High', 'Normal', 'High']

[['PlayTennis', 'Outlook', 'Temperature', 'Humidity', 'Wind']]

['Temperature', 'Humidity', 'Wind', 'PlayTennis']

PlayTennis

3

['Humidity']

[['No', 'Sunny', 'Hot', 'High', 'Weak']]

['Temperature', 'Humidity', 'Wind', 'PlayTennis']

PlayTennis

3

['High']

[['No', 'Sunny', 'Hot', 'High', 'Strong']]

['Temperature', 'Humidity', 'Wind', 'PlayTennis']

PlayTennis

3

['High']

[['Yes', 'Overcast', 'Hot', 'High', 'Weak']]

['Temperature', 'Humidity', 'Wind', 'PlayTennis']

PlayTennis

3

['High']

[['Yes', 'Rain', 'Mild', 'High', 'Weak']]

['Temperature', 'Humidity', 'Wind', 'PlayTennis']

PlayTennis

3

['High']

[['Yes', 'Rain', 'Cool', 'Normal', 'Weak']]

['Temperature', 'Humidity', 'Wind', 'PlayTennis']

PlayTennis

3

['Normal']

[['No', 'Rain', 'Cool', 'Normal', 'Strong']]

['Temperature', 'Humidity', 'Wind', 'PlayTennis']

PlayTennis

3

['Normal']

[['Yes', 'Overcast', 'Cool', 'Normal', 'Strong']]

['Temperature', 'Humidity', 'Wind', 'PlayTennis']

PlayTennis

3

['Normal']

[['No', 'Sunny', 'Mild', 'High', 'Weak']]

['Temperature', 'Humidity', 'Wind', 'PlayTennis']

PlayTennis

3

['High']

[['Yes', 'Sunny', 'Cool', 'Normal', 'Weak']]

['Temperature', 'Humidity', 'Wind', 'PlayTennis']

PlayTennis

3

['Normal']

[['Yes', 'Rain', 'Mild', 'Normal', 'Weak']]

['Temperature', 'Humidity', 'Wind', 'PlayTennis']

PlayTennis

3

['Normal']

[['Yes', 'Sunny', 'Mild', 'Normal', 'Strong']]

['Temperature', 'Humidity', 'Wind', 'PlayTennis']

PlayTennis

3

['Normal']

[['Yes', 'Overcast', 'Mild', 'High', 'Strong']]

['Temperature', 'Humidity', 'Wind', 'PlayTennis']

PlayTennis

3

['High']

[['Yes', 'Overcast', 'Hot', 'Normal', 'Weak']]

['Temperature', 'Humidity', 'Wind', 'PlayTennis']

PlayTennis

3

['Normal']

[['No', 'Rain', 'Mild', 'High', 'Strong']]

['Temperature', 'Humidity', 'Wind', 'PlayTennis']

PlayTennis

3

['High']

Decision Tree is as below:

{'Outlook': {'': 'Humidity', '0': 'High', '1': 'High', '2': 'High', '3': 'High', '4': 'Normal', '5': 'Normal', '6': 'Normal', '7': 'High', '8': 'Normal', '9':

'Normal', '10': 'Normal', '11': 'High', '12': 'Normal', '13': 'High'}} Testing instance is: ['Rain', 'Mild', 'High', 'Strong'] The Target value for the testing instance is: NULL

[ ]:

# 14.\_Back-propagation\_Algorithm

December 1, 2024

14

\_exp Back-propagation Algorithm

[1]:

**import**

**numpy**

**as**

**np**

*# Input and Output datasets*

X

=

np

.

array(([

2

,

9

]

,

[

1

,

5

[

]

,

3

,

6

])

, dtype

=

float

)

y

=

np

.

array(([

92

]

,

[

86

]

,

[

89

])

, dtype

=

float

)

*# Normalize input and output*

X

=

X

/

np

.

amax(X, axis

=

0

)

y

=

y

/

100

*# Sigmoid and derivative functions*

**def**

sigmoid

):

(

x

**return**

1

/

(

1

+

np

.

exp(

-

x))

**def**

derivatives\_sigmoid

(

x

):

**return**

x

\*

(

1

-

x)

*# Parameters*

epoch

=

500

*# Reduced number of epochs*

learning\_rate

=

0.1

inputlayer\_neurons

=

2

hiddenlayer\_neurons

=

3

outputlayer\_neurons

=

1

*# Initialize weights and biases*

wh

=

np

.

random

.

uniform(size

=

inputlayer\_neurons, hiddenlayer\_neurons

))

(

*#*

␣

↪

*Weights for hidden layer*

bh

=

np

.

random

.

uniform(size

=

(

1

, hiddenlayer\_neurons))

*# Bias for hidden layer*

wo

=

np

.

random

.

uniform(size

=

(

hiddenlayer\_neurons, outputlayer\_neurons

))

*#*

␣

↪

*Weights for output layer*

bo

=

np

.

random

.

uniform(size

=

(

1

, outputlayer\_neurons))

*# Bias for output layer*

*# Training the neural network*

**for**

i

**in**

range

(

epoch

):

*# Forward Propagation*

net\_h

=

np

.

dot(X, wh)

+

bh

*# Input to hidden layer*

1

sigma\_h = sigmoid(net\_h) *# Activation function for hidden layer* net\_o = np.dot(sigma\_h, wo) + bo *# Input to output layer* output = sigmoid(net\_o) *# Activation function for output layer*

*# Backpropagation*

deltaK = (y - output) \* derivatives\_sigmoid(output) *# Delta for output*␣

↪*layer* deltaH = deltaK.dot(wo.T) \* derivatives\_sigmoid(sigma\_h) *# Delta for*␣ ↪*hidden layer*

*# Update weights and biases* wo += sigma\_h.T.dot(deltaK) \* learning\_rate wh += X.T.dot(deltaH) \* learning\_rate bo += np.sum(deltaK, axis=0, keepdims=**True**) \* learning\_rate bh += np.sum(deltaH, axis=0, keepdims=**True**) \* learning\_rate

*# Calculate and display error*

error = np.mean(np.square(y - output)) *# Mean Squared Error*

**if** i % 100 == 0 **or** i == epoch - 1: *# Display every 100 epochs or on the*␣

↪*last epoch* print(f"Epoch -> **{**i**}**, Learning Rate -> **{**learning\_rate**}**, Error -> **{**error**:** ↪.6f**}**")

Epoch -> 0, Learning Rate -> 0.1, Error -> 0.002709

Epoch -> 100, Learning Rate -> 0.1, Error -> 0.000685

Epoch -> 200, Learning Rate -> 0.1, Error -> 0.000494

Epoch -> 300, Learning Rate -> 0.1, Error -> 0.000465

Epoch -> 400, Learning Rate -> 0.1, Error -> 0.000459

Epoch -> 499, Learning Rate -> 0.1, Error -> 0.000457

2

# 15.\_Naive\_Bayes\_Classifier\_PIMA\_

December 1, 2024

[1]:

**import**

**csv**

**import**

**random**

**import**

**numpy**

**as**

**np**

**import**

**pandas**

**as**

**pd**

**import**

**math**

[2]:

**def**

mean

(

numbers

):

**return**

sum

numbers

)

(

/

float

(

len

)

numbers

(

-

1

)

[3]:

**def**

stddev

):

numbers

(

avg

=

mean(numbers)

variance

=

sum

([

pow

(

x

-

avg,

2

)

**for**

x

**in**

numbers])

/

float

(

len

(

numbers

)

-

1

)

**return**

math

.

sqrt(variance)

[4]:

data

=

[]

[5]:

lines

=

csv

.

reader(

open

(

'

datasets/pima-indians.csv

'

,

'

r

'

))

[6]:

data

=

list

(

lines

)

[7]:

**for**

i

**in**

range

(

len

(

data

)):

data[i]

=

[

float

)

x

(

**for**

x

**in**

data[i]]

[8]:

trainset

=

[]

[9]:

test

=

list

)

data

(

[10]:

**while**

len

(

trainset

)

<

564

:

index

=

random

.

randrange(

len

(

test

))

trainset

.

append(test

.

pop(index))

[11]:

seperated

=

{}

[12]:

**for**

i

**in**

range

(

564

):

vector

=

data[i]

**if**

vector[

-

1

]

**not**

**in**

seperated:

seperated[vector[

-

1

]]

=

[]

seperated[vector[

-

1

]]

.

append(vector[

0

:

-

1

])

[13]:

summaries

=

{}

[14]:

**for**

classvalue, instances

**in**

seperated

.

items():

**for**

attribute

**in**

zip

(

\*

instances):

summaries[classvalue]

=

[(

mean(attribute), stddev(attribute

))

**for**

␣

↪

attribute

**in**

zip

(

\*

instances)]

[15]:

prediction

=

[]

[16]:

**for**

i

**in**

range

(

204

):

probabilities

=

{}

vector

=

test[i]

**for**

classvalue, classummary

**in**

summaries

.

items():

probabilities[classvalue]

=

1

**for**

j

**in**

range

(

len

(

classummary

)):

smean, sstdev

=

classummary[j]

x

=

vector[j]

expo

=

math

.

exp(

-

(

math

.

pow(x

-

smean,

2

)

/

(

2

\*

math

.

pow(sstdev,

2

))))

probabilities[classvalue]

\*

=

(

1

/

(

math

.

sqrt(

2

\*

math

.

pi)

\*

␣

↪

sstdev))

\*

expo

bestlabel, bestprob

=

**None**

,

-

1

**for**

classvalue, probability

**in**

probabilities

.

items():

**if**

bestlabel

**is**

**None**

**or**

probability

>

bestprob:

bestprob

=

probability

bestlabel

=

classvalue

result

=

bestprob, bestlabel

prediction

.

append(result)

[17]:

correct

=

0

[18]:

**for**

i

**in**

range

(

204

):

print

(

test[i

][

-

1

]

,

"

"

, prediction[i][

-

1

])

**if**

test[i][

-

1

]

==

prediction[i][

-

1

]:

correct

+

=

1

0.0

1.0

1.0

0.0

1.0 1.0 0.0 1.0 0.0 1.0 0.0 1.0 1.0 1.0 0.0 0.0 1.0 1.0 0.0 0.0 1.0 1.0 0.0 0.0

0.0 1.0

1.0

0.0 1.0 1.0 0.0 0.0 1.0 0.0 1.0 0.0 1.0 0.0 0.0 1.0 1.0 0.0 1.0 0.0 1.0 0.0 0.0 0.0 1.0 1.0 0.0 1.0 1.0 0.0 1.0 0.0 1.0 0.0 0.0 0.0 1.0 1.0 0.0 0.0 1.0 1.0 0.0 1.0 1.0 1.0 0.0 1.0 1.0 0.0 0.0 1.0 1.0 0.0 0.0 0.0 1.0 0.0 1.0 1.0 1.0 0.0 0.0 0.0 1.0 0.0 0.0 1.0 1.0 0.0 0.0 0.0 1.0 1.0 0.0 1.0 1.0 0.0 0.0 1.0 1.0 0.0 1.0 1.0 1.0 1.0 0.0 1.0 1.0

1.0 0.0

0.0

1.0

0.0

1.0 1.0 0.0 0.0 1.0 1.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 1.0 1.0 1.0 0.0 1.0 0.0 1.0 0.0 1.0 1.0 0.0 0.0 1.0 0.0 0.0 1.0 1.0 1.0 1.0 1.0 1.0 0.0 0.0 1.0 1.0 1.0 0.0 0.0 1.0 0.0 0.0 1.0 1.0 1.0 0.0 0.0 1.0 0.0 0.0 1.0 1.0 0.0 0.0 1.0 1.0 0.0 1.0 0.0 1.0 1.0 0.0 0.0 1.0 0.0 0.0 0.0 1.0 0.0 1.0 0.0 1.0 1.0 1.0 1.0 1.0

0.0 1.0

1.0 1.0 0.0

0.0

0.0

1.0 1.0 1.0 0.0 1.0 1.0 1.0 0.0 0.0 1.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 1.0 0.0 1.0 0.0 1.0 0.0 0.0 0.0 1.0 0.0 1.0 0.0 1.0 0.0 0.0 1.0 1.0 0.0 1.0 0.0 1.0 0.0 1.0 1.0 1.0 0.0 0.0 0.0 1.0 1.0 0.0 0.0 1.0 0.0 0.0 0.0 1.0 1.0 1.0 1.0 1.0 0.0 0.0 0.0 1.0 0.0 1.0 0.0 1.0 1.0 0.0 0.0 1.0 0.0 1.0 0.0 1.0 0.0 0.0 1.0 1.0

0.0 1.0

0.0 0.0 1.0

1.0

1.0

0.0 1.0 0.0 1.0 1.0 1.0 0.0 1.0 0.0 1.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 1.0 0.0 0.0 1.0 1.0 0.0 0.0 0.0 1.0 0.0 1.0 1.0 1.0 0.0 0.0 0.0 1.0 1.0 1.0 0.0 1.0 1.0 0.0 0.0 1.0 0.0 0.0 1.0 1.0 0.0 1.0 1.0 1.0 0.0 1.0 1.0 1.0 0.0 1.0 0.0 1.0 1.0 1.0 1.0 1.0 0.0 0.0 0.0 1.0 0.0 1.0 1.0 1.0 0.0 0.0 1.0 1.0 0.0 0.0 0.0 1.0 0.0 1.0 1.0 1.0

1.0 0.0

0.0 1.0 0.0

0.0

[19]:

print

(

correct

)

97

[20]: print('Accuracy: **{0}**'.format((correct / 204.0) \* 100))

Accuracy: 47.549019607843135

# 16.\_Naive\_Bayes\_Classifier\_\_Doc\_Classification\_

December 1, 2024

**import**

**pandas**

**as**

**pd**

[2]:

msg

=

pd

.

read\_csv(

'

datasets/text\_doc.csv

'

, names

=

[

'

message

'

,

'

label

'

])

[3]:

print

(

"

Total Instances of Dataset:

"

, msg

.

shape[

0

])

Total Instances of Dataset: 18

[4]:

msg[

'

labelnum

'

]

=

msg

.

label

.

map({

'

pos

'

:

1

,

'

neg

'

:

0

})

[6]:

X

=

msg

.

message

[7]:

y

=

msg

.

labelnum

[8]:

**from**

**sklearn**

**.**

**model\_selection**

**import**

train\_test\_split

[ ]:

Xtrain, Xtest, ytrain, ytest

=

train\_test\_split(X, y)

[ ]:

**from**

**sklearn**

**.**

**feature\_extraction**

**.**

**text**

**import**

CountVectorizer

[ ]:

count\_v

=

CountVectorizer()

[ ]:

Xtrain\_dm

=

count\_v

.

fit\_transform(Xtrain)

[ ]:

Xtest\_dm

=

count\_v

.

transform(Xtest)

[ ]:

df

=

pd

.

DataFrame(Xtrain\_dm

.

toarray(),columns

=

count\_v

.

get\_feature\_names())

[ ]:

print

(

df

[

0

:

5

])

[ ]:

**from**

**sklearn**

**.**

**naive\_bayes**

**import**

MultinomialNB

[ ]:

clf

=

MultinomialNB()

[ ]:

clf

.

fit(Xtrain\_dm, ytrain)

[ ]:

pred

=

clf

.

predict(Xtest\_dm)

[ ]:

**for**

doc, p

**in**

zip

(

Xtrain, pred

):

p

=

'

pos

'

**if**

p

==

1

**else**

'

neg

'

print

(

"

**%s**

->

**%s**

"

%

(

doc, p

))

[ ]:

**from**

**sklearn**

**.**

**metrics**

**import**

accuracy\_score, confusion\_matrix, precision\_score,

␣

↪

recall\_score

[ ]:

print

(

'

Accuracy Metrics:

**\n**

'

)

print

(

'

Accuracy:

'

, accuracy\_score(ytest, pred))

[ ]:

print

(

'

Recall:

'

, recall\_score(ytest, pred))

[ ]:

print

(

'

Precision:

'

, precision\_score(ytest, pred))

[ ]:

print

(

'

Confusion Matrix:

**\n**

'

, confusion\_matrix(ytest, pred))

[ ]:

# 17.1\_Bayes\_Network\_\_Diabetes\_

December 1, 2024

**import**

**numpy**

**as**

**np**

**import**

**pandas**

**as**

**pd**

[2]:

**from**

**pgmpy**

**.**

**models**

**import**

BayesianModel

**from**

**pgmpy**

**.**

**estimators**

**import**

MaximumLikelihoodEstimator

**from**

**pgmpy**

**.**

**inference**

**import**

VariableElimination

[3]:

heartDisease

=

pd

.

read\_csv(

'

datasets/pima-indians.csv

'

, names

=

[

'

preg

'

,

␣

↪

'

glucose

'

,

'

bp

'

,

'

skinthick

'

,

'

insulin

'

,

'

bmi

'

,

'

diapedigree

'

,

'

age

'

,

␣

↪

'

class

'

])

[4]:

heartDisease

.

head()

[4]:

preg

glucose

bp

skinthick

insulin

bmi

diapedigree

age

class

1. 6 148 72 35 0 33.6 0.627 50 1
2. 1 85 66 29 0 26.6 0.351 31 0
3. 8 183 64 0 0 23.3 0.672 32 1
4. 1 89 66 23 94 28.1 0.167 21 0
5. 0 137 40 35 168 43.1 2.288 33 1

[5]: model = BayesianModel([('preg', 'glucose'), ('bp', 'skinthick'), ('insulin',␣

↪'bmi'), ('bmi', 'diapedigree'), ('age', 'class'), ('insulin', 'class')])

[6]: model.fit(heartDisease, estimator=MaximumLikelihoodEstimator)

[7]: heart\_infer = VariableElimination(model)

[8]: print("Probability of heart disease given age = 28**\n**") q = heart\_infer.query(variables=['class'], evidence={'age': 28}) print(q['class'])

Probability of heart disease given age = 28

class phi(class)

class\_0 0.2598

class\_1 0.7402

[9]: print("Probability of heart disease given bp = 60**\n**") q = heart\_infer.query(variables=['class'], evidence={'insulin': 60}) print(q['class'])

Probability of heart disease given bp = 60

class phi(class)

class\_0 0.4896

class\_1 0.5104

[ ]:

# 2.\_BAYESIAN\_NETWORK\_\_USING\_API\_

December 1, 2024

**from bayespy.nodes import** Categorical, Mixture **from bayespy.inference import** VB

**import numpy as np**

FALSE = 0 TRUE = 1

**def** \_or(p\_false, p\_true): **return** np.take([p\_false, p\_true], [[FALSE, TRUE], [TRUE, TRUE]], axis=0)

asia = Categorical([0.5, 0.5])

tuberculosis = Mixture(asia, Categorical, [[0.99, 0.01], [0.8, 0.2]]) smoking = Categorical([0.5, 0.5]) lung = Mixture(smoking, Categorical, [[0.98, 0.02], [0.25, 0.75]]) bronchitis = Mixture(smoking, Categorical, [[0.97, 0.03], [0.08, 0.92]]) xray = Mixture(tuberculosis, Mixture, lung, Categorical,\_or([0.96, 0.04], [0.

↪115, 0.885])) dyspnea = Mixture(bronchitis, Mixture, tuberculosis, Mixture, lung,␣

↪Categorical,[\_or([0.6, 0.4],[0.18, 0.82]),\_or([0.11, 0.89], [0.04, 0.96])]) tuberculosis.observe(TRUE) smoking.observe(FALSE) bronchitis.observe(TRUE)

Q = VB(dyspnea, xray, bronchitis, lung, smoking, tuberculosis, asia)

Q.update(repeat=100)

print("P(asia):", asia.get\_moments()[0][TRUE]) print("P(tuberculosis):", tuberculosis.get\_moments()[0][TRUE]) print("P(smoking):", smoking.get\_moments()[0][TRUE]) print("P(lung):", lung.get\_moments()[0][TRUE]) print("P(bronchitis):", bronchitis.get\_moments()[0][TRUE]) print("P(xray):", xray.get\_moments()[0][TRUE]) print("P(dyspnea):", dyspnea.get\_moments()[0][TRUE])

Iteration 1: loglike=-6.453500e+00 (0.007 seconds)

Iteration 2: loglike=-6.453500e+00 (0.005 seconds) Converged at iteration 2.

P(asia): 0.9523809523809524

P(tuberculosis): 1.0

P(smoking): 0.0

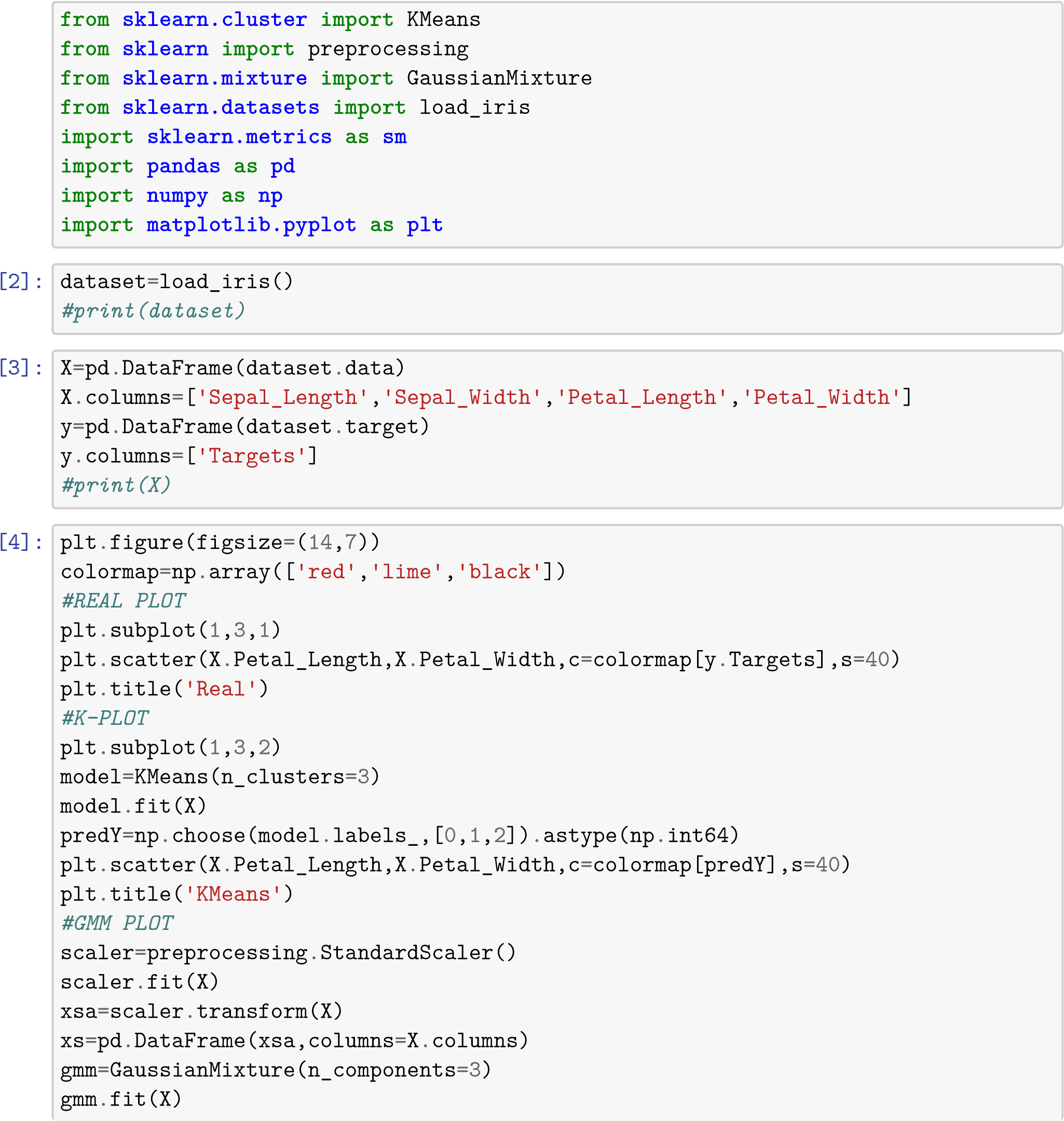
P(lung): 0.02

P(bronchitis): 1.0

P(xray): 0.885 P(dyspnea): 0.96 [ ]:

# 18.\_EM\_\_\_K-MEANS\_ALGORITHM\_\_USING\_API\_

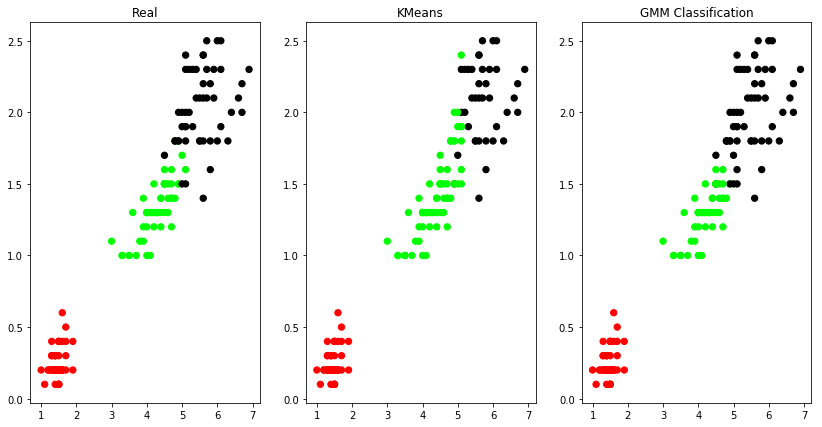
December 1, 2024



y\_cluster\_gmm=gmm.predict(X) plt.subplot(1,3,3)

plt.scatter(X.Petal\_Length,X.Petal\_Width,c=colormap[y\_cluster\_gmm],s=40) plt.title('GMM Classification')

[4]: Text(0.5,1,'GMM Classification')



[5]:

sm

.

accuracy\_score(y, predY)

[5]: 0.8933333333333333

[6]:

sm

.

confusion\_matrix(y, predY)

[6]: array([[50, 0, 0], [ 0, 48, 2], [ 0, 14, 36]])

[ ]:

19.\_k-

# NEAREST\_NEIGHBOUR\_ALGORITHM\_\_USING\_API\_

December 1, 2024

[1]:

**from**

**sklearn**

**.**

**datasets**

**import**

load\_iris

**from**

**sklearn**

**.**

**neighbors**

**import**

KNeighborsClassifier

**from**

**sklearn**

**.**

**model\_selection**

**import**

train\_test\_split

**import**

**numpy**

**as**

**np**

[6]:

dataset

=

load\_iris()

*#print(dataset)*

X\_train,X\_test,y\_train,y\_test

=

train\_test\_split(dataset[

"

data

"

]

,dataset

[

"

target

"

]

,random\_state

=

[3]:

print

(

len

(

dataset

[

"

data

"

]))

print

(

len

(

X\_train

))

print

(

len

(

X\_test

))

*#print(y\_train)*

*#print(y\_test)*

150

112

weights='uniform')

[5]:

**for**

i

**in**

range

(

len

(

X\_test

)):

x

=

X\_test[i]

x\_new

=

np

.

array([x])

prediction

=

kn

.

predict(x\_new)

␣

↪

print

(

"

TARGET=

"

,y\_test[i],dataset[

"

target\_names

"

][

y\_test[i]],

"

PREDICTED=

"

,prediction,dataset

print

(

kn

.

score(X\_test,y\_test))

38

[4]:

kn

=

KNeighborsClassifier(n\_neighbors

=

1

)

kn

.

fit(X\_train,y\_train)

[4]: KNeighborsClassifier(algorithm='auto', leaf\_size=30, metric='minkowski', metric\_params=None, n\_jobs=1, n\_neighbors=1, p=2,

TARGET= 2 virginica PREDICTED= [2] ['virginica']

TARGET= 1 versicolor PREDICTED= [1] ['versicolor']

TARGET= 0 setosa PREDICTED= [0] ['setosa']

TARGET= 2 virginica PREDICTED= [2] ['virginica']

TARGET= 0 setosa PREDICTED= [0] ['setosa']

TARGET= 2 virginica PREDICTED= [2] ['virginica']

TARGET= 0 setosa PREDICTED= [0] ['setosa']

TARGET= 1 versicolor PREDICTED= [1] ['versicolor']

TARGET= 1 versicolor PREDICTED= [1] ['versicolor']

TARGET= 1 versicolor PREDICTED= [1] ['versicolor']

TARGET= 2 virginica PREDICTED= [2] ['virginica']

TARGET= 1 versicolor PREDICTED= [1] ['versicolor']

TARGET= 1 versicolor PREDICTED= [1] ['versicolor']

TARGET= 1 versicolor PREDICTED= [1] ['versicolor']

TARGET= 1 versicolor PREDICTED= [1] ['versicolor']

TARGET= 0 setosa PREDICTED= [0] ['setosa']

TARGET= 1 versicolor PREDICTED= [1] ['versicolor']

TARGET= 1 versicolor PREDICTED= [1] ['versicolor']

TARGET= 0 setosa PREDICTED= [0] ['setosa']

TARGET= 0 setosa PREDICTED= [0] ['setosa']

TARGET= 2 virginica PREDICTED= [2] ['virginica']

TARGET= 1 versicolor PREDICTED= [1] ['versicolor']

TARGET= 0 setosa PREDICTED= [0] ['setosa']

TARGET= 0 setosa PREDICTED= [0] ['setosa']

TARGET= 2 virginica PREDICTED= [2] ['virginica']

TARGET= 0 setosa PREDICTED= [0] ['setosa']

TARGET= 0 setosa PREDICTED= [0] ['setosa']

TARGET= 1 versicolor PREDICTED= [1] ['versicolor']

TARGET= 1 versicolor PREDICTED= [1] ['versicolor']

TARGET= 0 setosa PREDICTED= [0] ['setosa']

TARGET= 2 virginica PREDICTED= [2] ['virginica']

TARGET= 1 versicolor PREDICTED= [1] ['versicolor']

TARGET= 0 setosa PREDICTED= [0] ['setosa']

TARGET= 2 virginica PREDICTED= [2] ['virginica']

TARGET= 2 virginica PREDICTED= [2] ['virginica']

TARGET= 1 versicolor PREDICTED= [1] ['versicolor']

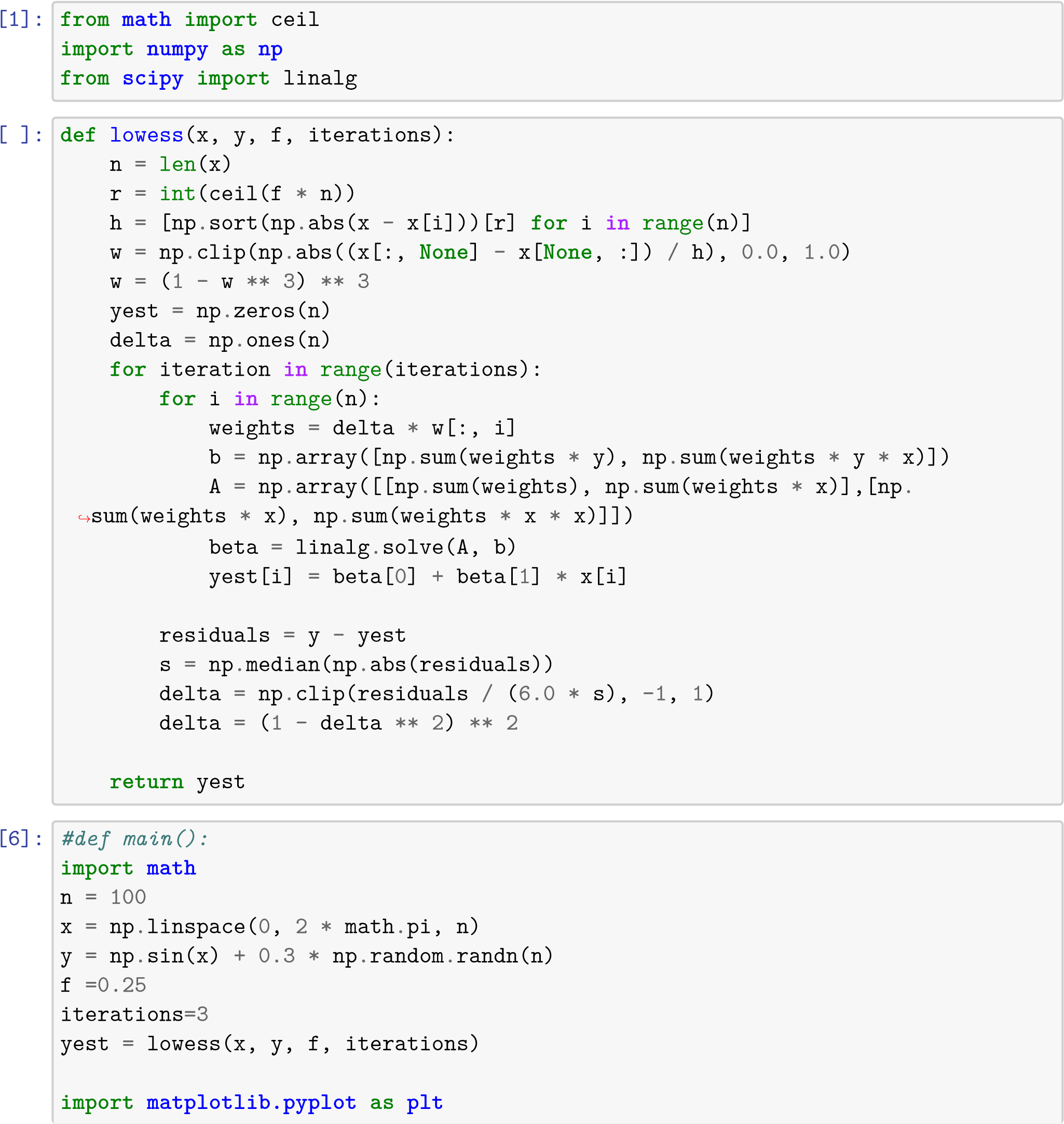
TARGET= 0 setosa PREDICTED= [0] ['setosa']

TARGET= 1 versicolor PREDICTED= [2] ['virginica']

0.9736842105263158

# 20.\_LOCALLY\_WEIGHTED\_REGRESSION

December 1, 2024



%

**matplotlib**

inline

plt

.

plot(x,y,

"

r.

"

)

plt

.

plot(x,yest,

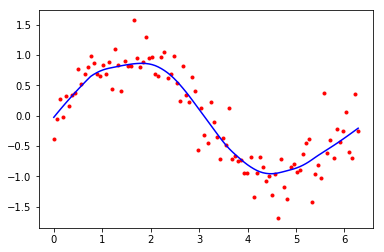
"

b-

"

)

[6]: [<matplotlib.lines.Line2D at 0x1101ce0f0>]



[7]:

test

=

np

.

sin(x)

plt

.

plot(x,y,

"

r.

"

)

plt

.

plot(x,test,

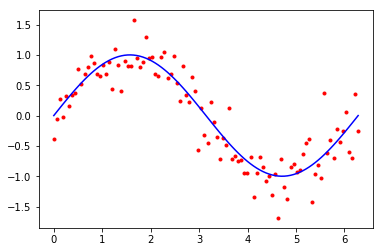
"

b-

"

)

[7]: [<matplotlib.lines.Line2D at 0x111763c18>]



[5]: x

[5]: array([0. , 0.06346652, 0.12693304, 0.19039955, 0.25386607,

0.31733259, 0.38079911, 0.44426563, 0.50773215, 0.57119866,

0.63466518, 0.6981317 , 0.76159822, 0.82506474, 0.88853126,

0.95199777, 1.01546429, 1.07893081, 1.14239733, 1.20586385,

1.26933037, 1.33279688, 1.3962634 , 1.45972992, 1.52319644,

1.58666296, 1.65012947, 1.71359599, 1.77706251, 1.84052903,

1.90399555, 1.96746207, 2.03092858, 2.0943951 , 2.15786162, 2.22132814, 2.28479466, 2.34826118, 2.41172769, 2.47519421, 2.53866073, 2.60212725, 2.66559377, 2.72906028, 2.7925268 , 2.85599332, 2.91945984, 2.98292636, 3.04639288, 3.10985939, 3.17332591, 3.23679243, 3.30025895, 3.36372547, 3.42719199, 3.4906585 , 3.55412502, 3.61759154, 3.68105806, 3.74452458, 3.8079911 , 3.87145761, 3.93492413, 3.99839065, 4.06185717, 4.12532369, 4.1887902 , 4.25225672, 4.31572324, 4.37918976, 4.44265628, 4.5061228 , 4.56958931, 4.63305583, 4.69652235, 4.75998887, 4.82345539, 4.88692191, 4.95038842, 5.01385494, 5.07732146, 5.14078798, 5.2042545 , 5.26772102, 5.33118753, 5.39465405, 5.45812057, 5.52158709, 5.58505361, 5.64852012,

5.71198664, 5.77545316, 5.83891968, 5.9023862 , 5.96585272, 6.02931923, 6.09278575, 6.15625227, 6.21971879, 6.28318531])

[ ]:

3