Lab-1

Date 21/3/24
Page

⇒ Write a python program to import and export data using pandas library functions

Code:

```
import pandas as pd
iris_data = pd.read_csv("iris.csv")
iris_data.head()
iris.columns = col_names
```

output:

	sl	sw	pl	pw	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.3	0.2	Iris-setosa
2	4.7	3.2	1.4	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.4	Iris-setosa

Code:

```
url = "https://archive.ics.uci.edu/ml/machine-learning-
      database/iris/iris.data"
col_names = ["sepal_length_in_cm",
             "sepal_width_in_cm",
             "petal_length_in_cm",
             "petal_width_in_cm",
             "class"]
iris_data = pd.read_csv(url, names=col_names)
iris_data.head()
```

output:

	sepal_length_in_cm	sepal_width_in_cm	petal length_in cm	pw	class
0	5.1	3.5	1.4	0.2	Iris-setosa

```
iris_data.to_csv("cleaned_iris_data.csv")
```

LAB-2

28/3/24

Get the Data

→ Download the data

```
import os
import tarfile
import urllib
```

- Using these libraries, download and extract the housing_data and load the data into "housing.csv".

```
housing.head()
housing.info()
housing.describe()
```

- Using the above commands, inspect the attributes of the loaded housing dataset.

- Using matplotlib and seaborn libraries, by plotting the histplot detect the outliers

Create the Test set

- Splitting the dataset on test ratio 0.2, i.e., training data is 80% of dataset and testing data is 20% of dataset

- Stratified sampling is when random chosen data are representative of a whole target population each homogeneous subgroup is called strata.

LAB-3

Date 4/4/24
Page

Discover and visualize the Data to gain insights

Visualize the data using matplotlib and seaborn libraries.
Calculating the standard correlation coefficient of every pair of columns

Prepare the data for machine learning algorithms

Data cleaning, handling text and categorical data, custom Transformers, feature scaling, Transformation pipelines etc., are done here

Select and Train model

At first linear regression model is used to train but the model is overfitting the data.
To tackle this, DecisionTreeRegressor model is used as it is capable of finding non-linear relationships within the data.
But, the decisiontree model is also overfitting so badly that it performs worse than the linear regression model.
At last RandomForest regressor model is used It is much better.

Fine tune your model

At last fine tuning of model is carried out, evaluating on the test set and then launch, monitor and maintaining the system.

18/04/2024

LAB-4

→ python implementation of linear regression

```
import numpy as np
import matplotlib.pyplot as plt

def estimate_coef(x,y):
    n = np.size(x)
    m_x = np.mean(x)
    m_y = np.mean(y)
    ss_xy = np.sum(y*x) - n*m_y*m_x
    ss_xx = np.sum(x*x) - n*m_x*m_x

    b_1 = ss_xy / ss_xx
    b_0 = m_y - b_1*m_x
    return(b_0, b_1)

def plot_regression_line(x,y,b):
    plt.scatter(x,y, color="m", marker="o", s=30)
    y_pred = b[0] + b[1]*x
    plt.plot(x, y_pred, color="g")
    plt.xlabel('x')
    plt.ylabel('y')

def main():
    x = np.array([0,1,2,....,9])
    y = np.array([1,3,2...12])
    b = estimate_coef(x,y)
    print(b)
    plot_regression_line(x,y,b)
```

Output :

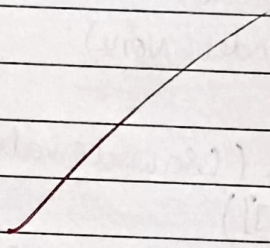
$(b_0, b_1) = (1.2363\ldots, 1.16969\ldots)$

→ multiple linear regression

```
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model,
metrics

data_url = "http://lib.stat.cmu.edu/datasets/boston"
raw_df = pd.read_csv(data_url, sep="\s+", skiprows=22
                     header=None)

x = np.hstack([raw_df.values[::2, :], raw_df.values
    [1::2, :2]])
y = raw_df.values[1::2, 2]

x_train, x_test, y_train, y_test = train_test_split(x, y,
  test_size=0.4, random_state=1)

reg = linear_model.LinearRegression()
reg.fit(x_train, y_train)

print("coefficients:", reg.coef_)

print("variance score: {}".format(reg.score(x_test,
                                   y_test)))

plt.style.use('fivethirtyeight')

plt.scatter(reg.predict(x_train), reg.predict(x_
    -train) - y_train, color="green",
    s=10, label="Train data")
```

```
plt.scatter(reg.predict(X_test),
            reg.predict(X_test) - y_test,
            color = "green", s=10,
            label = "Test data")

plt.hlines(y=0, xmin=0, xmax=50, linewidth=2)

plt.legend(loc='upper right')

plt.title("Residual errors")

plt.show()
```

blue



LAB-5

Date 25/04/2024
Page

Decision tree - ID3

```
import numpy as np
import pandas as pd
eps = np.finfo(float).eps
from numpy import log2 as log

from google.colab import drive
drive.mount ('/content/drive')
path = 'drive/My drive/datasets/play_tennis.csv'
df = pd.read_csv(path)
df.head()
df = df.drop ('day', axis=1)

def find_entropy (df):
    target = df.keys () [-1]
    entropy = 0
    values = df [target].unique()
    for value in values:
        fraction = df [target].value_counts() [value] /
                                        len(df [target])
        entropy += -fraction * np.log2 (fraction)
    return entropy

def avg_information (df, attribute):
    target = df.keys () [-1]
    target_variables = df [target].unique()
    variables = df [attribute].unique()
    entropy2 = 0
    for variable in variables:
        entropy = 0
        for target_variable in target_variables:
```

```
        num = len(df[attribute][df[attribute]==variable]
                  [df[target]==target_variable])
        den = len(df[attribute][df[attribute]==variable])
        fraction = num/den+eps
        entropy += -fraction * log(fraction+eps)
    fraction2 = den/len(df)
    entropy2 += -fraction2 * entropy
return abs(entropy2)

def find_winner(df):
    IG = []
    for key in df.keys()[:-1]:
        IG.append(find_entropy(df) - avg_information(
    print(IG)
    info = df.keys()[:-1][np.argmax(IG)]
    print(info)
    return df.keys()[:-1][np.argmax(IG)]

def get_subtable(df, node, value):
    res = df[df[node]==value].reset_index(drop=True)
    return res

def build_tree(df, tree=None):
    target = df.keys()[-1]
    node = find_winner(df)
    attvalue = np.unique(df[node])
    if tree is None:
        tree = {}
        tree[node] = {}
    for value in attvalue:
        subtable = get_subtable(df, node, value)
        clvalue, counts = np.unique(subtable[target]
                                    return_counts=True)
```

```
if len(counts) == 1:
    tree[node][value] = dvalue[0]
else:
    tree[node][value] = buildTree(subtable)
return tree

tree = buildTree(df)
import pprint
pprint.pprint(tree)
```

Output:

key))

```
{'outlook': {'overcast': 'Yes',
             'Rain': {'wind': {'strong': 'NO', 'weak':
                              'Yes'}},
             'sunny': {'humidity': {'high': 'NO', 'Norm
                                    al': 'Yes'}}}}
```

09-05-

Lab-6

09/05/2024

Logistic regression

```
import pandas as pd
data = pd.read_csv("cet_data.csv")

from sklearn.model_selection import train_test_split
from matplotlib import pyplot as plt
%.matplotlib inline

plt.scatter(data['CET_SCORE'], data['Admitted'],
marker='.', color='purple')
X_train, X_test, y_train, y_test = train_test_split(data
[['CET_SCORE']], data['Admitted'], train_size=0.8)

from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(X_train, y_train)
y_predicted = model.predict(X_test)
model.score(X_test, y_test)
print(y_predicted)
print(X_test)

from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X_train, y_train)
print("coefficient (m):", model.coef_)
print("Intercept (b):", model.intercept_)
```

```
import math
def Sigmoid(x):
    return 1/(1+math.exp(-x))

x=0
sigmoid_value = sigmoid(x)
print("Sigmoid value at x=", x, ":", sigmoid_value)

import math
def prediction_function(CET_Score):
    m=0.042
    b=-1.53
    z = m* CET_Score +b
    y= sigmoid(z)
    return y

predicted_probability = prediction_function(500)
print("predicted probability when CET_score is 500:"
      predicted_probability)
```

Output:

predicted probability when CET_score is 500: 0.9999

KNN implementation:

```
import numpy as np
import pandas as pd
from google.colab import drive
drive.mount('/content/drive')
dataset = pd.read_csv('/content/drive/MyDrive/iris.csv')
dataset.head()
```

id	sepallength	sepalwidth	petallength	petalwidth	Species
1	5.1	3.5	1.4	0.2	Iris-Set
2	4.9	3.0	1.4	0.2	Iris-Set
3	4.7	3.2	1.3	0.2	Iris-Set
4	4.6	3.1	1.5	0.2	Iris-Set
5	5.0	3.6	1.4	0.2	Iris-Set

```
dataset.groupby('species').size()
```

species	
Iris-setosa	50
Iris-versicolor	50
Iris-virginia	50

```
feature_columns = ['sepallength', 'sepalwidth', 'petallength',
                   'petalwidth']
X = dataset[feature_columns].values
y = dataset['species'].values

from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
y = le.fit_transform(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)
```

```
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

from sklearn.neighbours import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.model_selection import cross_val_score

classifier = KNeighborsClassifier(n_neighbors = 3)

classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred) * 100
print('Accuracy of our model is equal' +
      str(round(accuracy, 2)) + '%.')
```

output: Accuracy of our model is equal 96.67%

Lab-7

23/5/2024

K-means implementation

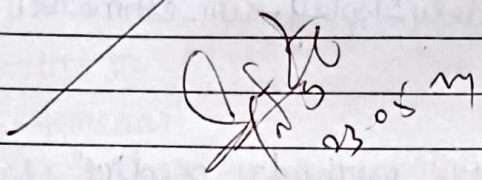
```
import pandas as pd
data = pd.read_csv("iris.csv")
data.head()

import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.cluster import KMeans

X,y = load_iris(return_X_y=True)

sse=[]
for k in range(1,11):
    km = KMeans
kmeans = KMeans(n_clusters=3, random_state=2)
kmeans.fit(X)

pred = kmeans.fit_predict(X)
pred
```

SVM :

```
from sklearn.datasets import load_breast_cancer
import matplotlib.pyplot as plt
from sklearn.inspection import DecisionBoundaryDisplay
from sklearn.svm import SVC

cancer = load_breast_cancer()
X = cancer.data[:, :2]
y = cancer.target

svm = SVC (kernel="rbf", gamma=0.5, C=1.0)
svm.fit = (X, y)

DecisionBoundaryDisplay.from_estimator(
        svm,
        X,
        response_method="predict",
        cmap = plt.cm.Spectral,
        alpha = 0.8,
        xlabel = cancer.feature_names[0],
        ylabel = cancer.feature_names[1],
    )

plt.scatter (X[:,0], X[:, 1], c=y, s=20, edgecolors="k")

plt.show()
```

PCA

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

from sklearn.datasets import load_breast_cancer
data = load_breast_cancer()
data.keys()

print (data['feature_names'])
```



```
df1 = pd.DataFrame (data['data'], columns = data['
                                    feature_names'])
scaling = StandardScaler()
scaling.fit (df1)
scaled_data = scaling.transform(df1)
principal = PCA (n_components=3)
principal.fit (scaled_data)
 x = principal.transform (scaled_data)

plt.figure (figsize = (10,10))
plt.scatter (x[:,0], x[:,1], c= data['target'], cmap='
plt.xlabel ('pc1')
plt.ylabel ('pc2')
```

Lab-8

Date 30/07/2024
Page

Random forest (classification)

Dataset : Titanic dataset

```
import pandas as pd
import warnings
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report

warnings.filterwarnings('ignore')

url = "./content/titanic.csv"
titanic_data = pd.read_csv(url)
titanic_data
X = titanic_data[['pclass', 'sex', 'age', 'sibsp', 'parch', 'Fare']]
y = titanic_data['survived']

X.loc[:, 'sex'] = X['sex'].map({'female':0, 'male':1})

X_train, X_test, y_train, y_test = train_test_split(X,
    y, test_size=0.2, random_state=42)

rf_classifier = RandomForestClassifier(n_estimators=100, rando
                                       state=42)

rf_classifier.fit(X_train, y_train)

y_pred = rf_classifier.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
```

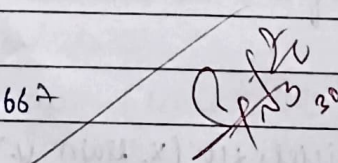
Output : Accuracy : 0.80

AdaBoost

```
from sklearn.datasets import load_iris
from sklearn.ensemble import AdaBoostClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

iris = load_iris()
X = iris.data
y = iris.target

X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.4, random_state=32)

adaboost_clf = AdaBoostClassifier(n_estimators=30,
learning_rate=1.0, random_state=42)
adaboost_clf.fit(X_train, y_train)

y_pred = adaboost_clf.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

output:

Accuracy: 0.966666



Lab-9

Date 05/06
Page

ANN:

```
import numpy as np
X = np.array([[2,9], [1,5], [3,6]), dtype = float)
y = np.array([[92], [86], [89]), dtype = float)
X = X/np.amax(X, axis=0)
y = y/100

epoch = 5000
lr = 0.1
inputlayer_neurons = 2
hiddenlayer_neurons = 3
output_neurons = 1

wh = np.random.uniform (size = (inputlayer_neurons, hiddenla
                                -yer_neurons))
bh = np.random.uniform (size = (1, hiddenlayer_neurons))
wout = np.random.uniform (size = (hiddenlayer_neurons, output_neur
bout = np.random.uniform (size = (1, output_neurons))      -ons))

def sigmoid(x):
  return 1/(1+np.exp(-x))
def derivatives_sigmoid(x):
  return x*(1-x)

for i in range (epoch):

  hinp1 = np.dot(X, wh)
  hinp = hinp1 + bh
  hlayer_act = sigmoid(hinp)
  outinp1 = np.dot(hlayer_act, wout)
  outinp = outinp1 + bout
  output = sigmoid(outinp)
```

```
EO = y - output
outgrad = derivates_sigmoid(output)
d_output = EO * outgrad
EH = d_output.dot(wout.T)
hiddengrad = derivates_sigmoid(hlayer_act)
d_hiddenlayer = EH * hiddengrad

wout += hlayer_act.T.dot(d_output) * lr
wh += X.T.dot(d_hiddenlayer) * lr

print("Input: \n" + str(x))
print("Actual output: \n" + str(y))
print("predicted output: \n", output)
```

Input:

```
[[0.666667 1.      ]
 [0.333333 0.555556]
 [1.       0.666667]]
```

Actual output:

```
[[0.92]
 [0.86]
 [0.89]]
```

predicted output:

```
[[0.86729245]
 [0.8455565]
 [0.8640413]]
```