import pandas as pd

import iris.csv ("/content/cleaned_iris.csv")

import data.head()

unnamed	0	sepal length in cm	sepal width	petal length	petal width
0	0				
1	1				
2	2				
3	2				
4	4				

import from url

import pandas as pd

url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"

define coulumns

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()

	sepal_length	sepal_width	petal_length	petal_width	class
0	5.1	8.5	1.4	0.2	iris-setosa
1					
2					
3					
4					

export the dataframe to the csv file

iris_data.to_csv("cleaned_iris_data.csv")

21/3/2024

Lab - 2 HANDS ON LEARNING WITH SCIKIT LEARN VERSION 1.0 FF

1. Framing the problem & looking at the bigger Picture

about the data: California housing data to build a model of housing prices

in the state features of the data include

- Population
- Median income
- Median housing price for each bik group in California

- The problem statement involves creating or building a pipeline

so that this models output can be fed into another model that attempts

to increase the ROI of the company on investing on a given district

- As most of the data is numerical in nature, except one feature,

the most obvious method of choose is Regression & the

performance measure chosen to visualize or interpret the data

is root mean square

$x^{(i)}$ → is a vector containing all of the input feature values excluding label

ex.

$y^{(i)}$ is the desired output of input $x^{(i)}$

X matrix → contains all feature values excluding labels & target

- describe ignores null values
- to plot histogram, you need to use .hist() with parameters, bins & fig size
- The encode function converts a variable-length string into a character string that is a text representation of the hex value of the binary seq

get the data

Download the data.

~~import~~ os

import tarfile

import urllib

use these diff libs to download tarfiles web-based dataset that is housing.tgz

extract the data into housing data retrieve that data into housing.tgz and read the data into housing.csv

housing.head()

housing.info()

housing.describe()

→ using these commands, inspect the attributes of the housing

import matplotlib.pyplot as plt

import seaborn as sns

housing.hist(bins=50, figsize=(20,15))

plt.show()

→ using the matplotlib & seaborn libraries detecting the outliers

lab - 3

Create testset

- splitting the dataset on test ratio = 0.2 ie, training data is 20% of the dataset
- Stratified Sampling is when random chosen data are representation of a whole target population. each homogenous subgroup is called strata

Discover & validate the Data to gain insights.

- Visualize the data using matplotlib & seaborn libraries
- Calculating the standard corr co-eff of every pair of columns

Prepare the data for machine learning algorithms.

- Data cleaning, handling text & categorical data, custom Transformers, feature scaling, tranformation pipelines etc are done here

Select & train model.

- at first linear regression model is used to train but the model is overfitting the data.
- To tackle this, Descicion tree Regressor model is used as it is capable of finding non-linear relationships within the data
  But the decision tree model is due overfitting so badly that it performs worse than the linear regression model.
  At last Randomforest regressor model is used. it is much better.

finetune your model.

- At last finetuning of model is carried out evaluating on the test set & then launch monitor and maintaining the systems
- evaluate your system on the test set by using mean-square method.

Launch, Monitor & Maintain your System

We can automate the process by

- collecting fresh data regularly & labeling it
- writing script to train model w/ the fine-tuned hyper parameters
- writing script to evaluate the model

Week - 4

Python implementation of lenial 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, 3, 4, 5, 6, 7, 8, 9])
    y = np.array([1, 3, 2, 5, 7, 8, 9, 10, 12])
    b = estimate_coef(x, y)
    print("Estimate co-eff : \nb_0 = {} \nb_1 = {}".
          format(b))
```

Output :

(b_0, b_1) = (1.2363 ... 1.16969 ...)

multiple linear regression

```
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import numpy as np
from sklearn import dataset, linear_model, multivariate

data_url = "url"
raw_df = pd.read_csv(data_url, sep="\s+", skiprows=22, header
        = none)
x = np.stack([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=
    random_state=1)
reg = linear_model.LinearRegression()
reg.fit(x_train, y_train)
print("coeff =", 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='blue', s=10, label='test data')
plt.hlines(y=0, xmin=0, xmax=0, xmax=50, linewidth=2)
plt.legend(loc='upper right')
plt.title("residual core")
plt.show()
```

```
Decision tree ID3

import numpy as np
import pandas as pd
df = pd.read_csv('...')
df.head()
df.info()
df.describe()

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 built_tree(df, tree=None):
    target = df.keys()[-1]
    node = find_minner(df)
    attr = np.unique(df[node])
    if tree is None:
        tree = {}
        tree[node] = {}
    for value in attr:
        sub = get_subtable(df, node, value)
        clvalue, counts = np.unique(suitable[target], return_counts=True)
        if len(counts) == 1:
            tree[node][value] = clvalue[0]
        else:
            tree[node][value] = built_tree(suitable
    return tree
tree = build_tree(df)
import pprint
```

Decision tree (sklearn)

```
import pandas as pd
import numpy as np
import sklearn.model_selection import DecisionTree
from sklearn.tree import plot_tree
df = pd.read_csv('path')
df.head()
df.info()
df.isNull().sum()
cols = df.columns[0:-1]
for i in cols:
    sns.boxplot(y = df[i])
    plt.show()
x = df.drop('species', axis=1)
y = df['species']
x_train, x_test, y_train, y_test = train_test_split(x, y,
        test_size=0.3)
dt = DecisionTreeClassifier(max_depth = 3)
dt.fit(x, y)
y_pred_train = dt.predict(x_train)
y_pred = dt.predict(x_test)
accuracy_score(y_pred, y_test)
```

09.05.24

KNN.

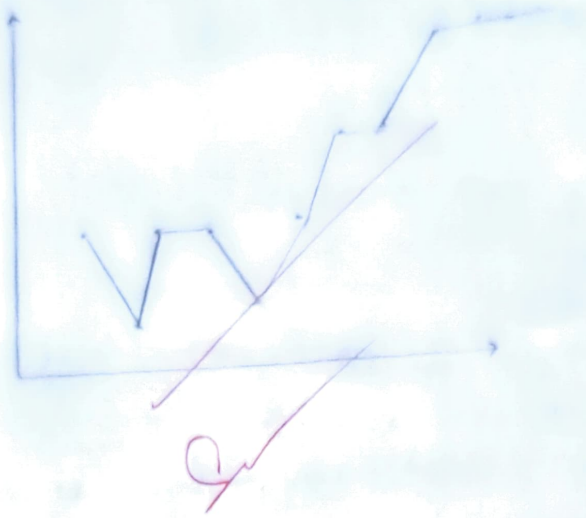
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

df = pd.read_csv('prostate.csv')
df.head()

from sklearn.preprocessing import StandardScaler
Scaler = StandardScaler()

Scaler.fit(df.drop('Target', axis=1))
Scaled_features = scaler.transform(df.drop('Target', axis=1))
df_feat = pd.DataFrame(scaled_features, columns = df.columns[:-1])
df_feat.head()

from sklearn import *

x_train, x_test, y_train, y_test = train_test_split(scaled_features,
                                                    df['Target'],
knn = KNeighborsClassifier(n_neighbors=1), test_size=0.50)
knn.fit(x_train, y_train)
pred = knn.predict(x_test)
print(confusion_matrix(y_test, pred))
print(classification_report(y_test, pred))

error_rate = []

for i in range(1,40):
    knn = KNeighborsClassifier(n_neighbors = i)
    knn.fit(x_train, y_train)
    pred_i = knn.predict(x_test)
    error_rate.append(np.mean(pred_i != y_test))

plt.figure(figsize=(10,6))
plt.plot(range(1,40), error_rate, color='blue',
    linestyle='dashed', marker='o'
```

plt.title ('Exponential Growth')

plt.xlabel ('x')

plt.ylabel ('exponential')

plt.show ()



Logistic Regression

```
from sklearn.datasets import load_breast_cancer
from sklearn.linear_model import LogisticRegression
from sklearn.model_ import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

x, y = load_breast_cancer(return_x_y = True)

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.10,
    random_state = 23)

clf = LogisticRegression(random_state=0)
clf.fit(x_train, y_train)
y_pred = clf.predict(x_test)

acc = accuracy_score(y_test, y_pred)
print("Logistic Regression model accuracy(in %):", acc*100)

digits = dataset.load_digits()
x = digits.data
y = digits.target

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

reg = Lenial_model.LogisticRegression()
reg.fit(x_train, y_train)
y_pred = reg.predict(x_test)
Print("Logic Regression model accuracy(in%):",
```

23.05.24

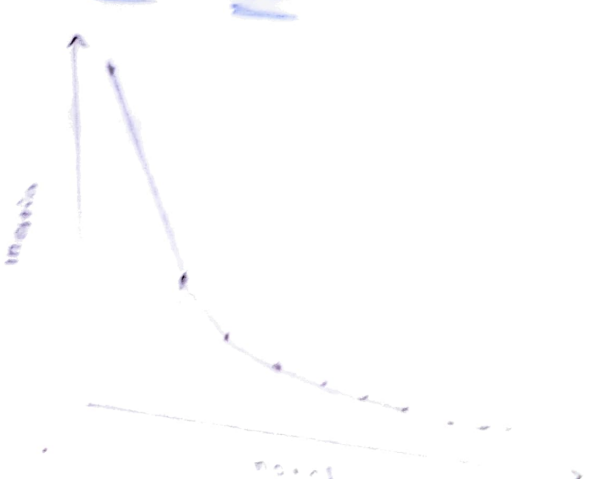
K - means clustering

import matplotlib.pyplot as plt

```
x = [4, 5, 10, 4, 3, 11, 14, 6, 10, 12]
y = [21, 19, 24, 17, 16, 25, 24, 22, 21, 21]
plt.scatter(x, y)
plt.show()
```

```
from sklearn.cluster import KMeans
data = list(zip(x, y))
inertias = []

for i in range(1, 11):
    kmeans = KMeans(n_clusters = i)
    kmeans.fit(data)
    inertias.append(kmeans.inertia_)

plt.plot(range(1, 11), inertias, marker = 'o')
plt.title('elbow method')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
plt.show()
```

Elbow method



```
import pandas as pd
import numpy as np
from sklearn.datasets import load_breast_cancer

cancer = load_breast_cancer(as_frame = True)
df = cancer.frame

print("original dataframe shape :", df.shape)

X = df[cancer['feature_names']]

print("input Dataframe shape :", X.shape)

X_mean = X.mean()
X_std = X.std()
Z = (X - X_mean) / X_std
C = Z.cov()

import matplotlib.pyplot as plt
import seaborn as sns
sns.heatmap(C)
plt.show()

# compute eigen values & Eigen vector
eigenvector = np.linalg.eig(C)
print("values :\n", eigenvalues)
print("values shape :", eigenvalues.shape)
print("Eigen vector shape :", eigenvector.shape)

# sort eigen values in descending order
idx = eigenvalues.argsort()[::-1]
eigenvalues = eigenvalues[idx]
eigenvectors = eigenvectors[:, idx]
explained_var = np.cumsum(eigenvalues) / np.sum(eigenvalues)
explained_var

# determine no. of principal components
n_comp = np.argmax(explained_var >= 0.90)+1
n_components
```

pca component as unit matrix

u = eigenvector[:, :n_components]

pca_comp = pd.DataFrame(u,

index = cancer['feature_names'],

columns = ['PC1', 'PC2']

)

plot heatmap.

plt.figure(figsize=(5,7))

sns.heatmap(pca_component)

plt.title('PCA components')

plt.show()

find projection in PCA

z_pca = z @ pca_component

z_pca.rename({'PC1': 'PCA1', 'PC2': 'PCA2'}, axis=1, inplace=True)

print(z_pca)

```
from sklearn.decomposition import PCA
pca = PCA(n_components = 2)
pca.fit(z)
x_pca = pca.transform(z)
df_pca1 = pd.DataFrame(x_pca,
                       columns = ['pc{}'.
                       format(i+1)
                       for i in range(n_components)])
                       print(df_pca1)
plt.figure(figsize = (8, 6))
plt.scatter(x_pca[:, 0], x_pca[:,1],
            c = cancer['target'],
            cmap = 'plasma')
plt.xlabel('first PCA')
plt.ylabel('2nd PCA')
plt.show().
pca.components_
```

SVM Support Vector Machine

```
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)
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,
edgecolor = 'k')

plt.show()
```

lab-8

a) implement random forest ensemble method on given dataset

b) implement Bagging ensemble method on given dataset

c) import pandas as pd

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn import datasets
iris = datasets.load_iris()
import pandas as pd
data = pd.DataFrame({
    'sepal length': iris.data[:, 0],
    'sepal width': iris.data[:, 1],
    'petal length': iris.data[:, 2],
    'petal width': iris.data[:, 3],
    'species': iris.target
})
data.head()
from sklearn.model_selection import train_test_split
x = data[['sepal length', 'sepal width', 'petal length', 'petal wi
    th']]
y = data['species']
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(n_estimators=100)
clf.fit(x_train, y_train)
y_pred = clf.predict(x_test)
from sklearn import metrics
print("Accuracy:", metrics.accuracy_score(y_test, y_pred))
confusion_matrix = metrics.confusion_matrix(y_test, y_pred)
print(confusion_matrix)
```

output: ['setosa' 'versicolor' 'virginica']

['sepal length' 'sepal width' 'petal length' 'petal wid

Accuracy = 0.8888

conf_mat = [[10 0 0]
[0 15 1]
[0 4 15]]

Implement boosting

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score

cricket_df = pd.read_csv('/content/cricket_dataset.csv')
x = cricket_df.drop(columns=['player_Name', 'Player_Type'])
y = cricket_df['player_Type']

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)

boosting = GradientBoostingClassifier(n_estimators=100, learning_rate=0.1, random_state=42)
boosting.fit(x_train, y_train)
boosting_predictions = boosting.predict(x_test)
boosting_accuracy = accuracy_score(y_test, boosting_predictions)
print("Boosting Accuracy:", boosting_accuracy)
```

Output

Boosting Accuracy: 0.35

30.05.24