Assignment No : 1

Assignment Name : Introduction to pycharm , Pandas Library, DataFrames, And Loading CSV File in DataFrame

---

```
import pandas as pd
'''pd.__version__'''

df1 = pd.DataFrame({"A": [1, 2, 3], "B": [2, 3, 4]}, index=[0, 1, 2])
print("df1:\n", df1)

df2 = pd.DataFrame({"B": [4, 5, 7], "C": ["x", "y", "z"]}, index=[4, 5, 6])
print("\ndf2:\n", df2)

df3 = df1.combine_first(df2)
print("\n combination of df1 and df2:\n", df3)


classes = pd.Series(["mathematics", "chemistry", "physics", "history", "geography",
"german"]) grades = pd.Series([90, 54, 77, 22, 25, 40]) year = pd. Series([2015, 2016, 2017,
2018, 2019, 2020])
df4 = pd. DataFrame({"Classes": classes, "Grades": grades, "Year": year})
print("\n", df4)

# upload a csv file  in sample_data section
# load the .csv in data frame

data_frame = pd.read_csv("C:/Users/sejal/PycharmProjects/dataset.csv")
print("\n", data_frame)
```

Practical No 2:-implements the find-s inductive learning algorithm

---

Code:

```
import pandas as pd
print(pd.__version__)

import numpy as np
print(np.__version__)

data = pd.read_csv("C:/Users/Vaishnavi/Desktop/dataset1.CSV")
print("Given Data set") print(data,"n")

d=np.array(data)[:,:-1] print("n
the attributes are:",d)

target=np.array(data)[:,-1]
print("n the target is :",target)

def train(c,t):
  for i,val in enumerate(t):
if val=='yes':
        sp_hp = c[i].copy()
break   print("initially
hypothesis=")
print(sp_hp,"\n")

  for i,val in enumerate(c):
    if t[i]=='yes':           for x in
range(len(sp_hp)):                  if
val[x]!=sp_hp[x]:
            sp_hp[x]='?'          else:
pass     print("hypothesis is
",i,"=",sp_hp)   return sp_hp
print("\n the final hypothesis is :",train(d,target))
```

Practical No:-3

Practical Name:-Implement the Candidate-Elimination Inductive Learning algorithm.

```
import numpy as np import
pandas as pd
data = pd.read_csv('C:/Users/comp/Desktop/datafile.csv')
concepts = np.array(data.iloc[:,0:-1])
print("\nInstances are:\n",concepts) target =
np.array(data.iloc[:,-1]) print("\nTarget values are:
",target) def learn(concepts,target):     specific_h =
concepts[0].copy()
    print("\nInitialization of specific_h and general_h")
print("\nSpecific Boundary: ", specific_h)
    general_h = [["?"for i in range(len(specific_h))] for i in
range(len(specific_h))]
    print("\nGeneric Boundary: ",general_h)
for i,h in enumerate(concepts):
        print("\nInstance",i+1 , "is ", h)
if target[i] == "Yes":
            print("Instance is positive ")              for i in
range(len(specific_h)):                         if h[x]!=
specific_h[x]:                           specific_h[x] ='?'
general_h[x][x] ='?'          else:                 print("Instance
is Negative")              for x in range(len(specific_h)):
if h[x]!= specific_h[x] and specific_h[x]!='?':
general_h[x][x] = specific_h[x]                          else:
                    general_h[x][x] = '?'
        print("Specific Boundary after",i+1,"Instance is",specific_h)
print("Generic Boundary after", i + 1, "Instance is",general_h)
print("\n")

    indices = [i for i,val in enumerate(general_h) if val ==
['?','?','?','?','?','?']]
for i in indices:
        general_h.remove(['?','?','?','?','?','?'])
return specific_h,general_h s_final,g_final =
learn(concepts,target)

print("Final Specific_h:",s_final,sep="\n") print("Final
General_h:",g_final,sep="\n")
```

C:\Users\comp\AppData\Local\Programs\Python\Python310\python.exe

C:/Users/comp/Desktop/ml.py/ml2.py

Practical No:-4
Practical Name:-Implementation of simple linear regression and mean square error.

```
import numpy as np

def     estimate_coef(x,     y):
#no.of     observations/points
n = np.size(x)
   #mean of x and y vector
m_x = np.mean(x)      m_y
= np.mean(y)
   #calculating cross-deviation and deviation about x
   SS_xy = np.sum(y * x) - n * m_y * m_x
   SS_xx = np.sum(x * x) - n * m_x * m_x

   #calculating regression coeificient
b_1 = SS_xy / SS_xx     b_0 = m_y
- b_1 * m_x
   return (b_0, b_1)

def main():
   #observations/data     x = np.array([0, 1, 2, 3,
4, 5, 6, 7, 8, 9, 10])     y = np.array([1, 3, 2, 5, 7,
8, 8, 9, 10, 12, 15])
   #estimating coefficients     b = estimate_coef(x, y)     print("Estimated
coefficients:\nb_0 = {} \nb_1 = {}".format(b[0], b[1]))     y_pred = b[0] +
b[1] * x     print("x input :", x)     print("Original y:", y)     print("y_pred:",
y_pred)     e = y - y_pred     merror = np.sum(e*e)     n = np.size(x)
   print("Mean square error =", merror/(2*n))

if __name__ == "__main__":
   main()
```

Practical No:-5 (5.1)

Practical Name:-Write a program to implement Decision Tree using Python/R/Programming language of your choice.

```
import matplotlib.pyplot as plt import
pandas as pd
from sklearn.datasets import load_iris data_b =
load_iris()
df=pd.DataFrame(data_b.data,columns=data_b.feature_names)   df['target']=
data_b.target
#df['target'] print(df)
#print(data_b)
print("Dataset Labels=",data_b.target_names)
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics from sklearn
import tree
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(df[data_b.feature_names], df['target'])
print(x_train) print(x_test) print(y_train) print(y_test)
clf = DecisionTreeClassifier(max_depth = 5,random_state =1, criterion='gini') #'gini
clf = clf.fit(x_train, y_train) y_pred = clf.predict(x_test) print(y_test, y_pred)
print("Accurancy:",metrics.accuracy_score(y_test, y_pred))
fn=['sepal length (cm)', 'sepal width(cm)', 'petal length(cm)', 'petal width(cm)']
cn=['setosa', 'versicolor', 'virqinica']

fig,axes = plt.subplots(nrows = 1, ncols = 1, figsize = (4, 4), dpi = 300)
tree.plot_tree(clf, feature_names = fn, class_names = cn, filled = True);
fig.savefig('dsting.png')
```

Practical No 05:- Implement To Decision Tree To Popular Attribute Selection Measure Like Information Gain,Gini Index Etc.For Decision Tree

---

```
import matplotlib.pyplot as plt import
pandas as pd
from sklearn.datasets import load_iris data_b
= load_iris()
df=pd.DataFrame(data_b.data,columns=data_b.feature_names) df['target']
= data_b.target
#df['target'] print(df) #print(data_b)
print("Dataset Labels=",data_b.target_names)
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics from sklearn
import tree
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(df[data_b.feature_names], df['target'])
print(x_train) print(x_test) print(y_train) print(y_test)
clf = DecisionTreeClassifier(max_depth = 5,random_state =1, criterion='gini')  #'gini'
clf = clf.fit(x_train, y_train) y_pred = clf.predict(x_test) print(y_test, y_pred)
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
fn=['sepal length  (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)'] cn=['setosa',
'versicolor', 'virginica']

fig, axes = plt.subplots(nrows = 1, ncols = 1, figsize = (4, 4), dpi = 300)
tree.plot_tree(clf, feature_names = fn, class_names = cn, filled = True); fig.savefig('dstimg.png')
```

Practical No: 6

Practical Name: Implement simple KNN using Euclidean distance in Python.

Code: KNN using Euclidean distance

```
from pandas import DataFrame from
sklearn.datasets import load_iris
data_b = load_iris()
df= DataFrame(data_b.data, columns=data_b.feature_names) df['target']
= data_b.target
#print(df)
#print(data_b.DESCR)
print("Dataset      Labels=",data_b.target_names)
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
from sklearn.metrics import confusion_matrix from
sklearn.model_selection import train_test_split
X_train, X_test, Y_train, y_test = train_test_split(df[data_b.feature_names],
df['target'], random_state=1) print(X_train.head(6)) print(Y_train.head(6))
print(X_test.head())
clf = KNeighborsClassifier(n_neighbors=6)
clf.fit(X_train, Y_train) # model is trained
y_pred=clf.predict(X_test) #print(y_test,
y_pred)
print("Accuracy:",metrics.accuracy_score(y_test,
y_pred)) cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:") print(cm)
```

Code: For Breast Cancer Data Set from pandas

```
import DataFrame #from sklearn.datasets import
load_iris from sklearn.datasets import
load_breast_cancer

from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics from sklearn.metrics
import confusion_matrix
from sklearn.model_selection import train_test_split
#data_b = load_iris() data_b = load_breast_cancer() df =
DataFrame(data_b.data, columns=data_b.feature_names)
df['target'] = data_b.target
# print(df)
# print(data_b.DESCR) print("Dataset Labels=",
data_b.target_names)

X_train, X_test, Y_train, y_test = train_test_split(df[data_b.feature_names], df['target'],
random_state=1) print(X_train.head(6)) print(Y_train.head(6)) print(X_test.head()) clf
= KNeighborsClassifier(n_neighbors=6) clf.fit(X_train, Y_train)  # model is trained
y_pred = clf.predict(X_test) # print(y_test, y_pred) print("Accuracy:",
metrics.accuracy_score(y_test, y_pred)) cm = confusion_matrix(y_test, y_pred)


print("Confusion Matrix:") print(cm)
# corr = cm[0, 0] + cm[1, 1] + cm[2, 2]   # ----for iris
# corr = cm[0, 0] + cm[1, 1]  #----for breast cancer
corr = 0 for i in range(len(data_b.target_names)):
   corr = corr + cm[i, i] wrg = len(y_test) - corr
print("Number of correct predictions=", corr)
print("Number of wrong predictions = ", wrg)
```

Practical No. :- 8

Practical Name:- . Write a program for confusion matrix and calculate precision, recall, f-measure

---

```
from sklearn.datasets import load_iris, load_breast_cancer from
sklearn.model_selection import train_test_split from sklearn.neighbors import
KNeighborsClassifier from sklearn.metrics import confusion_matrix, precision_score,
recall_score, f1_score

# Load the Irish dataset
iris = load_iris() X_iris =
iris.data y_iris =
iris.target

# Split the Irish dataset into training and testing sets
X_train_iris, X_test_iris, y_train_iris, y_test_iris = train_test_split(X_iris, y_iris,
test_size=0.2, random_state=42)

# Train the KNN classifier on the Irish d3ataset knn_iris
= KNeighborsClassifier() knn_iris.fit(X_train_iris,
y_train_iris)

# Make predictions on the Irish testing set y_pred_iris
= knn_iris.predict(X_test_iris)

# Calculate the confusion matrix for Irish dataset
cm_iris = confusion_matrix(y_test_iris, y_pred_iris)
print("Confusion Matrix (Irish Dataset):") print(cm_iris)

# Calculate precision, recall, and F-measure for Irish dataset
precision_iris = precision_score(y_test_iris, y_pred_iris, average='macro')
recall_iris = recall_score(y_test_iris, y_pred_iris, average='macro') f1_iris
= f1_score(y_test_iris, y_pred_iris, average='macro')

print("Precision (Irish Dataset):", precision_iris)
print("Recall (Irish Dataset):", recall_iris) print("F-measure
(Irish Dataset):", f1_iris)


# Load the Breast Cancer dataset
cancer = load_breast_cancer()
```

```
X_cancer = cancer.data y_cancer
= cancer.target

# Split the Breast Cancer dataset into training and testing sets
X_train_cancer, X_test_cancer, y_train_cancer, y_test_cancer = train_test_split(X_cancer,
y_cancer,test_size=0.2, random_state=42)

# Train the KNN classifier on the Breast Cancer dataset
knn_cancer = KNeighborsClassifier()
knn_cancer.fit(X_train_cancer, y_train_cancer)

# Make predictions on the Breast Cancer testing set
y_pred_cancer = knn_cancer.predict(X_test_cancer)

# Calculate the confusion matrix for Breast Cancer dataset
cm_cancer = confusion_matrix(y_test_cancer, y_pred_cancer)
print("\nConfusion Matrix (Breast Cancer Dataset):")
print(cm_cancer)

# Calculate precision, recall, and F-measure for Breast Cancer dataset
precision_cancer = precision_score(y_test_cancer, y_pred_cancer)
recall_cancer = recall_score(y_test_cancer, y_pred_cancer) f1_cancer
= f1_score(y_test_cancer, y_pred_cancer)

print("Precision (Breast Cancer Dataset):", precision_cancer)
print("Recall (Irish Dataset):", recall_cancer) print("F-measure
(Irish Dataset):", f1_cancer)
```

Practical no. :- 9

Practical name :- . Write a program for linear regression and find parameters like sum of squared errors (sse), total sum of squares (sst), r2, adjusted r2, etc

---

```
import numpy as np from sklearn.linear_model
import LinearRegression from sklearn.metrics
import r2_score

# Input data
X = np.array([[1, 1], [1, 2], [2, 2], [2, 3]]) y
= np.array([3, 4, 5, 6])

model = LinearRegression()   # Create a linear regression model

model.fit(X, y)   # Fit the model to the data

y_pred = model.predict(X)  # Predict the output

sse = np.sum((y_pred - y) ** 2)   # Calculate SSE (Sum of Squared Errors)

sst = np.sum((y - np.mean(y)) ** 2)  # Calculate SST (Total Sum of Squares)

r2 = r2_score(y, y_pred)  # Calculate R2 score

# Calculate adjusted R2 n = X.shape[0]  #
Number of samples p = X.shape[1]  #
Number of predictors adjusted_r2 = 1 - (1 -
r2) * (n - 1) / (n - p - 1)

# Print the results print("Sum of Squared
Errors(SSE):- ", sse) print("Total Sum of
Squares(SST):- ", sst) print("R Square(R2):-
", r2)
print("Adjusted Square(R2):- ", adjusted_r2 )
```

Assignment No:-10

Assignment Name:-Write the program to implement the naive Bayesian Classifier for a sample training dataset stored as a .CSV file. Compute the accuracy of the classifier considering a few test dataset.

```
import numpy as np import
pandas as pd import
matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn import datasets from
sklearn.naive_bayes import GaussianNB from
sklearn.metrics import confusion_matrix iris =
datasets.load_iris() #load dataset x = iris.data
#input y = iris.target #traget
print("Features :", iris['feature_names'])

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.25, random_state = 0)
NB = GaussianNB()
NB.fit(x_train, y_train) y_pred =
NB.predict(x_test) cm =
confusion_matrix(y_test,y_pred)
print("Confusion Matrix") print(cm)
```

NO:-11.1

Practical Name:- implementing agglomerative clustering in python.

```
from sklearn.cluster import

AgglomerativeClustering from sklearn.datasets

import make_blobs import matplotlib.pyplot as plt


# Generate sample data

X, y = make_blobs(n_samples=200, centers=4, random_state=0)


# Create an instance of AgglomerativeClustering

clustering = AgglomerativeClustering(n_clusters=4)


# Perform clustering

clustering.fit(X)


# Retrieve the cluster labels labels

= clustering.labels_


# Plot the data points with their corresponding cluster labels

plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis')

plt.xlabel("Feature 1") plt.ylabel("Feature 2")

plt.title("Agglomerative Clustering") plt.show()
```

Assignment No.:- 11.2

Assignment Name:-Write a Program for Fuzzy c-means clustering in python.

```
import numpy as np import
skfuzzy as fuzz from skfuzzy
import control as ctrl

# Generate some example
data np.random.seed(0) data =
np.random.rand(100, 2)

# Define the number of clusters
n_clusters = 3

# Apply fuzzy c-means clustering cntr, u, u0, d, jm, p, fpc =
fuzz.cluster.cmeans(   data.T,   n_clusters,   2,   error=0.005,
maxiter=1000, init=None

)

# Predict cluster membership for each data point
cluster_membership = np.argmax(u, axis=0)

# Print the cluster centers
print('Cluster Centers:', cntr)

# Print the cluster membership for each data point
print('Cluster Membership:', cluster_membership)
```

Practical no. :- 12

Practical name :- implement the non-parametric locally weighted regression algorithm in order to fit data points. Select the appropriate data set for your experiment and draw graphs.

```
from math import ceil import
numpy as np
from scipy import linalg


def lowess(x, y, f, iterations):
   n = len(x)     r =
int(ceil(f * n))
   h = [np.sort(np.abs(x - x[i]))[r] for i in range(n)]     w =
np.clip(np.abs((x[:, None] - x[None, :]) / h), 0.0, 1.0)     w =
(1 - w ** 3) ** 3     yest = np.zeros(n)     delta = np.ones(n)
for iteration in range(iterations):        for i in range(n):
weights = delta * w[:, i]
          b = np.array([np.sum(weights * y), np.sum(weights * y * x)])
          A = np.array([[np.sum(weights), np.sum(weights * x)], [np.sum(weights * x), np.sum(weights *
x * x)]])
          beta = linalg.solve(A, b)
yest[i] = beta[0] + beta[1] * x[i]

       residuals = y - yest
       s = np.median(np.abs(residuals))
delta = np.clip(residuals / (6.0 * s), -1, 1)
delta = (1 - delta ** 2) ** 2

   return yest


import math

n = 100
x = np.linspace(0, 2 * math.pi, n) y =
np.sin(x) + 0.3 * np.random.randn(n) f =
0.25 iterations = 3
yest = lowess(x, y, f, iterations)

import matplotlib.pyplot as plt
plt.plot(x, y, "r.") plt.plot(x,
yest, "b-")
plt.show()
```

Practical No.: 13.1
Practical Name: Construction Of simple Neural Network using Python

---

Code:

```
import numpy as np from scipy.special import expit as
activation_function from scipy.stats import truncnorm

# define the network
# generate numbers within a truncated (bounded)
# normal Distribution


def truncated_normal(mean=0, sd=1, low=0, upp=10):
   return truncnorm((low - mean) / sd, (upp - mean) / sd, loc=mean, scale=sd)

# creat the Network class and define the arguments:
# set the no. of neurons/nodes for each layer
# and initialize the weight matrices


class Nnetwork:
   def __init__(self, no_of_in_nodes, no_of_out_nodes, no_of_hidden_nodes,
learning_rate):
      self.no_of_in_nodes = no_of_in_nodes
self.no_of_out_nodes = no_of_out_nodes
self.no_of_hidden_nodes = no_of_hidden_nodes
self.learning_rate = learning_rate
self.create_weight_matrices()

   def create_weight_matrices(self):
      """A method to initialize the weight matrices of the neural network"""       rad
= 1 / np.sqrt(self.no_of_in_nodes)  # rad = 0.2707        x =
truncated_normal(mean=0, sd=1, low=-rad, upp=rad)        self.weight_in_hidden
= x.rvs((self.no_of_hidden_nodes, self.no_of_in_nodes))
print("weights_in_hidden = ", self.weight_in_hidden)        rad =
1/np.sqrt(self.no_of_hidden_nodes)        x = truncated_normal(mean=0, sd=1,
low=-rad, upp=rad)        self.weight_in_hidden_out = x.rvs((self.no_of_out_nodes,
self.no_of_hidden_nodes))        print("weights_in_hidden_out = ",
self.weight_in_hidden_out)

   def train(self, input_vector, target_vector):
pass

   def run(self, input_vector):
      input_vector = np.array(input_vector, ndmin=2).T
print("Input = ", input_vector)
```

```
        input_hidden = activation_function(self.weight_in_hidden @ input_vector)
print("Hidden = ", input_hidden)

        output_vector = activation_function(self.weight_in_hidden_out @ input_hidden)
print("Output = ", output_vector)         return output_vector

simple_network = Nnetwork(no_of_in_nodes=2, no_of_out_nodes=2,
no_of_hidden_nodes=4, learning_rate=0.6)

#run simple network for arrays, lists and tuples with shape (2):

y = simple_network.run([2,3]) print("Y
= ", y)
```

Practical No: 13.2
Practical Name: Classification Of Iris Dataset By Applying Artificial Neural Network With Back-Propogation Algorithm

---

classification of iris data set by aplying artificial neural network using Back-propogation algorithm **import** numpy **as** np **import** pandas **as** pd
from sklearn.datasets **import** load_iris **from** sklearn.model_selection **import** train_test_split
import matplotlib.pyplot **as** plt

# load dataset
data = load_iris()

# Get features and target
x = data.data y =
data.target print(**"Y="**,
y)

y = pd.get_dummies(y).values
print(y[:3])

split data into train and test data
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=20, random_state=4)

# initialize variable
learning_rate = 0.1
iteration = 6000
N = y_train.size

number of input features input_size
= 4

number of hidden layers neurons hidden_size
= 2

mo. of neurons at output layers output_size
= 3
results = pd.DataFrame(columns=[**"mse"**, **"accuracy"**])

# initialize weights np.random.seed(10)
# initialiizing weight for the hidden layers
W1 = np.random.normal(scale=0.5, size=(input_size, hidden_size)) print(**"weight 1"**, W1)

# initializing weight for the output layers
W2 = np.random.normal(scale=0.5, size=(hidden_size, output_size)) print(**"weight 2"**, W2)

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


def mean_squared_error(y_pred, y_true):
    return (((y_pred - y_true) ** 2).sum()) / (2 * y_pred.size)


def    accuracy(y_pred,     y_true):                    acc     =
y_pred.argmax(axis=1)       ==       y_true.argmax(axis=1)
return acc.mean()


for itr in range(iteration):

    # feedforward propagation
    # on hidden layer
    Z1 = np.dot(x_train, W1)
    A1 = sigmoid(Z1)

# on output layer
    Z2 = np.dot(A1, W2)
    A2 = sigmoid(Z2)

# calculating error
    mse = mean_squared_error(A2, y_train)
    acc = accuracy(A2, y_train)
    results = results._append({"mse": mse, "accuracy": acc}, ignore_index=True)

# backpropagation
    E1 = A2 - y_train      dw1
= E1 * A2 * (1 - A2)

    E2 = np.dot(dw1, W2.T)
    dw2 = E2 * A1 * (1 - A1)

# weight updates
    W2_update = np.dot(A1.T, dw1) / N
    W1_update = np.dot(x_train.T, dw2) / N

    W2 = W2 - learning_rate * W2_update
    W1 = W1 - learning_rate * W1_update

results.mse.plot(title="Mean squared Error")
```

```
results.accuracy.plot(title="Accuracy")

# feedforward
Z1 = np.dot(x_test, W1)
A1 = sigmoid(Z1)

Z2 = np.dot(A1, W2)
A2 = sigmoid(Z2)

acc = accuracy(A2, y_test) print("Accuracy:
{}".format(acc))
```