Pratical Exam

1 – Hadoop and hdfs

<https://www.oracle.com/java/technologies/downloads/#java8-windows>

<https://hadoop.apache.org/releases.html>

move files to c drive

jave install de 8 kit

developing tools – next – extracting – next – java – ok - install

copy files to java

Windows->settings->system->environment variables for system->edit the system environment variables

Seeting – one drive – ok – java home – jdk java copy path – down path – edit – new path – ok done

Go to java

Cmd

Javac

Java – version

Install haddop folder c me haddop

hadoop->etc-> Hadoop

Core-site.xml

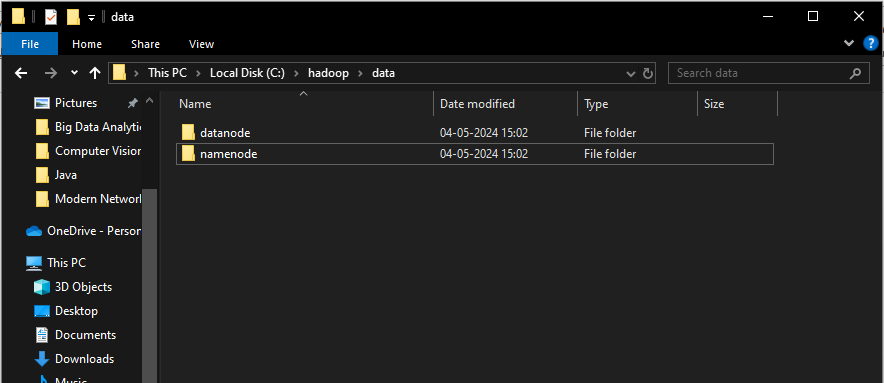
Hdfs-site.xml

Mapred-site.xml

Yarn-site.xml

Hadoop-env windows command prompt file

Open all these files in notepad



Setting new path me jaaake Hadoop bhi dalldo

Cmd

Hdfs namenode -format

cd..

cd..

cd Hadoop

cd sbin

start-all.cmd

start-yarn.cmd

2 –

**Implement an application that stores big data in Hbase / MongoDB and manipulate it using R / Python.**

<https://www.mongodb.com/try/download/community>

5.0.26 onstall

C:\data\db directory

C:\Program Files\MongoDB\Server\5.0\bin

Mongod on cmd

Go to

C:\Program Files\MongoDB\Server\5.0\bin

Start command prompt from this location

Fire the command: mongo

show dbs

my\_database:

db.books.insert({“name”:”It Ends with us”})

show collections;

db.books.find()

seeitng mw jaooo nwea pth and mango add karo

restart karkr mondod and monsgo

!pip install pymongo

Code

import pymongo

mongo\_uri = "mongodb://localhost:27017/"

client = pymongo.MongoClient(mongo\_uri)

print(client.list\_database\_names())

db = client.my\_database

print(db.list\_collection\_names())

table=db.books

print(table.count\_documents({}))

print('Ayush Patel - 53004230035')

3 –

**Implement Regression Model to import a data from web storage. Name the dataset and now do Logistic Regression to find out relation between variables. Also check if the model is fit or not.**

**Code**

import matplotlib.pyplot as plt

import pandas as pd

import numpy as np

from sklearn import datasets, linear\_model

from sklearn.metrics import mean\_squared\_error, r2\_score

# Load Dataset

diabetes = datasets.load\_diabetes()

# Load the diabetes dataset

diabetes\_X, diabetes\_y = datasets.load\_diabetes(return\_X\_y = True)

# Description of the dataset

print(diabetes['DESCR'])

print(diabetes.feature\_names)

# Use only one feature

diabetes\_X = diabetes\_X[:, np.newaxis, 2]

# Split the data intro training and testing datasets

diabetes\_X\_train = diabetes\_X[:-20]

diabetes\_X\_test = diabetes\_X[-20:]

# Split the targets into training and testing datasets

diabetes\_y\_train = diabetes\_y[:-20]

diabetes\_y\_test = diabetes\_y[-20:]

# Create linear regression object

regr = linear\_model.LinearRegression()

# Train the model using the training sets

regr.fit(diabetes\_X\_train, diabetes\_y\_train)

# Make predictions using the testing sets

diabetes\_y\_pred = regr.predict(diabetes\_X\_test)

# The Coefficients

print('Coefficients : \n', regr.coef\_)

# The mean squared error

print("Mean Squared Error: %.2f" % mean\_squared\_error(diabetes\_y\_test,diabetes\_y\_pred))

# The coefficient of determination: 1 is perfect prediction

print("Coefficient of determination: %.2f" % r2\_score(diabetes\_y\_test,diabetes\_y\_pred))

# plot the outputs

plt.scatter(diabetes\_X\_test, diabetes\_y\_test, color='blue')

plt.plot(diabetes\_X\_test, diabetes\_y\_pred, color='red', linewidth=3)

plt.xticks(())

plt.yticks(())

plt.title('Linear Regression')

plt.show()

print('Ayush PAtel - 53004230035')

4 –

**Apply Multiple Regression on a dataset having a continuous independent variable.**

**Code**

import matplotlib.pyplot as plt

import pandas as pd

import numpy as np

from sklearn import datasets, linear\_model

from sklearn.metrics import mean\_squared\_error, r2\_score

diabetes = datasets.load\_diabetes()

diabetes\_X, diabetes\_y = datasets.load\_diabetes(return\_X\_y=True)

# Description of the dataset

print(diabetes['DESCR'])

print(diabetes.feature\_names)

diabetes\_X = diabetes\_X[:, np.newaxis, 0]

diabetes\_X\_train = diabetes\_X[:-30]

diabetes\_X\_test = diabetes\_X[-30:]

diabetes\_y\_train = diabetes\_y[:-30]

diabetes\_y\_test = diabetes\_y[-30:]

regr = linear\_model.LinearRegression()

regr.fit(diabetes\_X\_train, diabetes\_y\_train)

diabetes\_y\_pred = regr.predict(diabetes\_X\_test)

print('Age')

print("Coefficients: \n", regr.coef\_)

print("Mean squared error: %.2f" % mean\_squared\_error(diabetes\_y\_test,

diabetes\_y\_pred))

print("Coefficient of determination: %.2f" % r2\_score(diabetes\_y\_test,

diabetes\_y\_pred))

plt.scatter(diabetes\_X\_test, diabetes\_y\_test, color="red")

plt.plot(diabetes\_X\_test, diabetes\_y\_pred, color="red", linewidth=2,

label='Age')

plt.xticks(())

plt.yticks(())

plt.title('Multiple Regression')

#plt.xlabel('Age')

plt.ylabel('Disease Progression')

diabetes\_X, diabetes\_y = datasets.load\_diabetes(return\_X\_y=True)

print(diabetes.feature\_names)

diabetes\_X = diabetes\_X[:, np.newaxis, 3]

diabetes\_X\_train = diabetes\_X[:-30]

diabetes\_X\_test = diabetes\_X[-30:]

diabetes\_y\_train = diabetes\_y[:-30]

diabetes\_y\_test = diabetes\_y[-30:]

regr = linear\_model.LinearRegression()

regr.fit(diabetes\_X\_train, diabetes\_y\_train)

# Make predictions using the testing set

diabetes\_y\_pred = regr.predict(diabetes\_X\_test)

print('BP')

print("Coefficients: \n", regr.coef\_)

print("Mean squared error: %.2f" % mean\_squared\_error(diabetes\_y\_test,

diabetes\_y\_pred))

print("Coefficient of determination: %.2f" % r2\_score(diabetes\_y\_test,

diabetes\_y\_pred))

plt.scatter(diabetes\_X\_test, diabetes\_y\_test, color="blue")

plt.plot(diabetes\_X\_test, diabetes\_y\_pred, color="blue", linewidth=2,

label='BP')

plt.xticks(())

plt.yticks(())

plt.title('Multiple Regression')

plt.ylabel('Disease Progression')

diabetes\_X, diabetes\_y = datasets.load\_diabetes(return\_X\_y=True)

print(diabetes.feature\_names)

diabetes\_X = diabetes\_X[:, np.newaxis, 2]

diabetes\_X\_train = diabetes\_X[:-30]

diabetes\_X\_test = diabetes\_X[-30:]

diabetes\_y\_train = diabetes\_y[:-30]

diabetes\_y\_test = diabetes\_y[-30:]

regr = linear\_model.LinearRegression()

regr.fit(diabetes\_X\_train, diabetes\_y\_train)

diabetes\_y\_pred = regr.predict(diabetes\_X\_test)

print('BMI')

print("Coefficients: \n", regr.coef\_)

print("Mean squared error: %.2f" % mean\_squared\_error(diabetes\_y\_test,

diabetes\_y\_pred))

print("Coefficient of determination: %.2f" % r2\_score(diabetes\_y\_test,

diabetes\_y\_pred))

plt.scatter(diabetes\_X\_test, diabetes\_y\_test, color="black")

plt.plot(diabetes\_X\_test, diabetes\_y\_pred, color="black", linewidth=2,

label='BMI')

plt.xticks(())

plt.yticks(())

plt.title('Multiple Regression')

plt.ylabel('Disease Progression')

plt.legend()

plt.show()

print('Ayush PAtel - 53004230035')

5 –

**Build a Classification Model**

**Code**

import pandas as pd

col\_names = ['pregnant','glucose','bp','skin','insulin','bmi','pedigree','age','label']

# Load Dataset

pima = pd.read\_csv('diabetes.csv', header=None, names=col\_names)

pima.head()

# Split dataset in features and target variable

feature\_cols = ['pregnant','insulin','bmi','age','glucose','bp','pedigree']

X = pima[feature\_cols] # Features

Y = pima.label # Target variable

# Split X and Y into training and testing sets

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

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X,Y,test\_size=0.25,random\_state=16)

# import the class

from sklearn.linear\_model import LogisticRegression

logreg = LogisticRegression(random\_state=16)

# fit the model with data

logreg.fit(X\_train,Y\_train)

Y\_pred = logreg.predict(X\_test)

# import the metrics class

from sklearn import metrics

cnf\_matrix = metrics.confusion\_matrix(Y\_test,Y\_pred)

cnf\_matrix

# Visualizing confusion matrix using HeatMap

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

class\_names = [0,1]

fig, ax = plt.subplots()

tick\_marks = np.arange(len(class\_names))

plt.xticks(tick\_marks, class\_names)

plt.yticks(tick\_marks, class\_names)

# create HeatMap

sns.heatmap(pd.DataFrame(cnf\_matrix), annot=True, cmap='YlGnBu', fmt='g')

ax.xaxis.set\_label\_position('top')

plt.tight\_layout()

plt.title('Confusion Matrix', y=1.1)

plt.ylabel('Actual Label')

plt.xlabel('Predicted Label')

from sklearn.metrics import classification\_report

target\_names = ['Without Diabetes', 'With Diabetes']

print('Classification Report:-')

print(classification\_report(Y\_test,Y\_pred,target\_names=target\_names))

# ROC Curve

Y\_pred\_proba = logreg.predict\_proba(X\_test)[::,1]

fpr, tpr, \_ = metrics.roc\_curve(Y\_test, Y\_pred\_proba)

auc = metrics.roc\_auc\_score(Y\_test, Y\_pred\_proba)

plt.plot(fpr, tpr, label = 'data 1, auc = '+str(auc))

plt.legend(loc=4)

plt.show()

print('Ayush Patel - 53004230035')

6 –

**Build a clustering model**

**Code**

# k-means clustering

from numpy import unique

from numpy import where

from sklearn.datasets import make\_classification

from sklearn.cluster import KMeans

from matplotlib import pyplot

# define dataset

X, \_ = make\_classification(n\_samples=1000, n\_features=2, n\_informative=2, n\_redundant=0, n\_clusters\_per\_class=1, random\_state=4)

# define the model

model = KMeans(n\_clusters=2)

# fit the model

model.fit(X)

# assign a cluster to each example

yhat = model.predict(X)

# retrieve unique clusters

clusters = unique(yhat)

# create scatter plot for samples from each cluster

for cluster in clusters:

# get row indexes for samples with this cluster

row\_ix = where(yhat == cluster)

# create scatter of these samples

pyplot.scatter(X[row\_ix, 0], X[row\_ix, 1])

# show the plot

pyplot.show()

print("K-Means Clustering")

print("Ayush Patel - 53004230035")

# Agglomerative clustering

from numpy import where

from sklearn.datasets import make\_classification

from sklearn.cluster import AgglomerativeClustering

from matplotlib import pyplot

# define dataset

X, \_ = make\_classification(n\_samples=1000, n\_features=2,

n\_informative=2, n\_redundant=0, n\_clusters\_per\_class=1,

random\_state=4)

# define the model

model = AgglomerativeClustering(n\_clusters=2)

# fit model and predict clusters

yhat = model.fit\_predict(X)

# retrieve unique clusters

clusters = unique(yhat)

# create scatter plot for samples from each cluster

for cluster in clusters:

# get row indexes for samples with this cluster

row\_ix = where(yhat == cluster)

# create scatter of these samples

pyplot.scatter(X[row\_ix, 0], X[row\_ix, 1])

# show the plot

pyplot.show()

print('Ayush Patel - 53004230035')

print('Agglomerative Clustering')

7 –

**Implement SVM classification technique**

**Code**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

# Load Dataset

dataset = pd.read\_csv('Social\_Network\_Ads.csv')

dataset.head()

# Split Dataset into X and Y

X = dataset.iloc[:, [2,3]].values

Y = dataset.iloc[:, 4].values

# Split the X and Y dataset into Training set and Testing set

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

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

# Perform feature scaling- feature scaling helps us to normalize the data within a particular range

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)

# Fit SVM to the training set

from sklearn.svm import SVC

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

classifier.fit(X\_train, Y\_train)

# Predict the test set results

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

# Make the confusion matrix

from sklearn.metrics import confusion\_matrix, accuracy\_score

cnf = confusion\_matrix(Y\_test, Y\_pred)

print('Confusion Matrix:-')

print(cnf)

print('Accuracy Score:-')

accuracy\_score(Y\_test, Y\_pred)

# Visualise the test set results

from matplotlib.colors import ListedColormap

X\_set, Y\_set = X\_test, Y\_test

X1, X2 = np.meshgrid(np.arange(start=X\_set[:,0].min()-1, stop=X\_set[:,0].max()+1,step=0.01),

np.arange(start=X\_set[:,1].min()-1, stop=X\_set[:,1].max()+1,step=0.01))

plt.contour(X1, X2, classifier.predict(np.array([X1.ravel(),X2.ravel()]).T).reshape(X1.shape), alpha=0.75, cmap=ListedColormap(('red','green')))

plt.xlim(X1.min(),X1.max())

plt.ylim(X2.min(),X2.max())

for i,j in enumerate(np.unique(Y\_set)):

plt.scatter(X\_set[Y\_set==j,0], X\_set[Y\_set==j,1], c=ListedColormap(('red','green'))(i),label=j)

plt.title('SVM (Test Set)')

plt.xlabel('Age')

plt.ylabel('Estimated Salary')

plt.legend()

plt.show()

Print(‘Ayush Patel - 53004230035’)

8 –

**Implement Decision Tree classification technique**

**Code**

import pandas as pd

from sklearn.tree import DecisionTreeClassifier

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

from sklearn import metrics

col\_names = ['pregnant', 'glucose', 'bp', 'skin', 'insulin', 'bmi', 'pedigree', 'age', 'label']

pima = pd.read\_csv('diabetes.csv',header=None, names=col\_names)

pima.head()

# Split dataset into features and target variable

feature\_cols = ['pregnant', 'insulin', 'bmi', 'age', 'glucose', 'bp', 'pedigree']

X = pima[feature\_cols]

Y = pima.label

# Split dataset into training set and testing set

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X,Y,test\_size=0.3,random\_state=1)

# 70% training 30% testing

# Create Decision Tree Classifier Object

clf = DecisionTreeClassifier()

# Train Decision Tree Classifier

clf = clf.fit(X\_train,Y\_train)

#Predict the response for test dataset

Y\_pred = clf.predict(X\_test)

# Model Accuracy, how often is the classifier correct?

print('Accuracy:- ',metrics.accuracy\_score(Y\_test,Y\_pred))

print('Ayush Patel - 53004230035')

# Visualizing Decision Trees

from sklearn.tree import export\_graphviz

from six import StringIO

from IPython.display import Image

import pydotplus

import os

os.environ["PATH"] += os.pathsep + r'C:\Users\DELL\anaconda3\Library\bin\graphviz'

dot\_data = StringIO()

export\_graphviz(clf, out\_file=dot\_data, filled=True, rounded=True, special\_characters=True,

feature\_names=feature\_cols, class\_names=['0','1'])

graph = pydotplus.graph\_from\_dot\_data(dot\_data.getvalue())

graph.write\_png('diabetes.png')

Image(graph.create\_png())

9 –

**Naïve Bayes Implementation**

**Code**

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

# Importing the dataset

dataset = pd.read\_csv('Social\_Network\_Ads.csv')

X = dataset.iloc[:, [2, 3]].values

y = dataset.iloc[:, 4].values

# Splitting the dataset into the Training set and Test set

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

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

# Feature Scaling

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)

# Fitting classifier to the Training set

from sklearn.naive\_bayes import GaussianNB

classifier = GaussianNB()

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

# Predicting the Test set results

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

# Making the Confusion Matrix

from sklearn.metrics import confusion\_matrix

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

# Visualising the Training set results

from matplotlib.colors import ListedColormap

X\_set, y\_set = X\_train, y\_train

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01),

np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),

alpha = 0.75, cmap = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],

c = ListedColormap(('red', 'green'))(i), label = j)

plt.title('Naive Bayes (Training set)')

plt.xlabel('Age')

plt.ylabel('Estimated Salary')

plt.legend()

print('Nishi Jain-53004230036')

plt.show()

print(‘Ayush Patel - 53004230035’)

# Visualising the Test set results

from matplotlib.colors import ListedColormap

X\_set, y\_set = X\_test, y\_test

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01),

np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),

alpha = 0.75, cmap = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],

c = ListedColormap(('red', 'green'))(i), label = j)

plt.title('Naive Bayes (Test set)')

plt.xlabel('Age')

plt.ylabel('Estimated Salary')

plt.legend()

plt.show()

print(‘Ayush Patel - 53004230035’)