Lab-1

1) Reading data from URL:

Code:

```
import pandas as pd
url= "https://archive.ics.uci.edu/ml/mt-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()
```

O/p:

	sepal length-in cm	sepal_width-in-cm	petal length in cm
0	5.1	3.5	1.4
1	4.9	3.0	1.4
2	4.7		

petal width-in cm	class
0.2	iris_setosa
0.2	iris setosa

2) Reading data from csv file

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

O/p:

	sepal length in cm	sepal width in cm	petal length in cm	petal width in cm
0	5.1	3.5	1.4	0.2

Lab-2

1) Look at the big picture.

=) Features & columns of dataset must be examined.

=) Frame problem and the questions to be addressed. In the process, regression task is also considered.

=) Performance task

=) Regression problems need root mean squared error. If there are many outliers mean absolute error is used.

2) Get the data

It is preferable to create some util function to automate the process of downloading/extracting web-based data sets

Contents of URL are ~~[illegible]~~ extracted into housing path

Histogram for all numerical data points are plotted against the count

Testing data set is sampled from the above (20%) & remaining 80% goes for training

Lab-3

-> Create data set
80% of the data set is taken as training set and 20% is taken as ~~setting~~ testing set.
Considering test if program is run again, it generates different test set
The test data are always preferred to be hidden. It is scalable & extendable.

-> Discover & Visualize the data
For visualising training set is analysed. It is preferred to make copy of training set.
Examining dataset since we have latitude & longitude info, map visualisations is done
Various visualisations parameters like colour, alpha value, figsize, kind are used to make the visualisation look better.

-> Looking for Correlation.
Various attributes will be inter-related or vary with eachother in a pattern. This is given by correlation. Correlation efficient lies between -1 to 1 when coefficient is close to be 1 it means there is strong (+)ve correlation b/w variable.

-> Prepare data for ML-Algorithms.
Data Cleansing
It is initial step where the missing values, Null values etc are handled.

-> Select and train model
=) Linear regression model can be used to train but it can overfit the data

→ Hence we can use decision tree regressor model is used cause it is capable of finding non-linear relationships within the data but it can be overfitting and can perform worse than linear regression model. To prevent this, Random Forest regression is used.

→ Tuning your Model
→ The model is fine tuned evaluated in test set and launch monitor and maintaining system.
→ Using mean squared error method, evaluate your system on test set.

Launch, Monitor & Maintain
→ Collect fresh data regularly and label it
→ Write script to train model & tune its parameters
→ Write script to evaluate the model

MY PAGES
Date __/__/__

Lab-4

Python's implementation of Linear Regression

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

def estimate_cost(x,y)
    n = np.size(x)
    mx = np.mean(x)
    my = np.mean(y)
    ss_xy = np.sum(y*x) - n*my*mx
    ss_xx = np.sum(x*x) - n*mx*mx
    b_1 = ss_xy/ss_xx
    b_0 = my - b_1*mx
    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_cost(x, y)
    print(b)
    plot_regression_line(x, y, b)
```

O/b = (b_0, b_1) = (1.2363 ... , 1.64 ...)

Multiple Linear Regression.

```
from sklearn model.selection import train test_split
import matplotlib.pyplot
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+", skiprow
        = 21 header = None)

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

x train, x test, y_train, y test = train test_split (x,y,
        test = size = 0.4, randomstate = 1)

reg = linear model. linear regression ()
reg-fit (x_train, y train)

print ("coefficents = ", reg.coef)
print ("Variance score= {3}"
                    format (reg-score (x_test),
                    (y test))

plt.style_use ("fire thirty eight")

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

plt.legend(loc='upper right')
plt.title("residual error")
plt.show()
```

Lab 5

1) Implementation of 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/ml dataset/PlayTennis.csv
```

O/p = Mounted at /content/drive.

	outlook	temp	humidity	windy	play.
0	sunny	hot	high	false	no
1	sunny	hot	high	true	no

```
=) print(f'Rows : {df.shape[0]}, Columns: {df.shape[1]}')
```

O/p = Rows = 14, Columns = 5

```
=) print(df.columns)
```

O/p : index(['outlook', 'temp', 'humidity', 'windy', 'play'],

```
=) df.info() ; df.describe()

def 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])
  return entropy
```

MY PAGES
Date __/__/__

```
def avg_info(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)
```

09.05.24

Lab 6

Logistic Regression

```
import pandas as pd
from matplotlib import pyplot as plt
from sklearn.metrics import accuracy
dt.pd.read_csv(path)

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

plt.scatter(data['Score'], data['Admitted'].
    marker='.', color='purple')
x_train, x_test, y_train, y_test = train_test_split
  (data['SCORE']], data['Admitted'], size=0.8)

from sklearn linear model import logistic regression.
model = LogisticRegression()
model.fit (x_train, y_train)
y_predicted = model.predict (x_test)
model.score(x_test, ytest)
print (y_predicted)
print (x_test)

from sklearn linear model import Linearregression
model = LinearRegression()
model.fit (x_train, y_train)
print ("Co-efficient (m)": model:coet)
print ("Intercept(b):", model: intercept)
```

O/p: Prediction : 0.999

K-NN implementation:

```
import numpy as np
import pandas as pd

from google.colab import drive
drive.mount('drive')
ds = pd.read_csv('iris.csv')
ds.head()

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

feature_columns = ['sepal length', 'sepal width', 'petal length',
                   'petal width']

X = ds[feature_columns].values
Y = ds['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(X, Y, test_size = 0.2,
    random_state = 0)

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

from sklearn.neighbors import KNeighborsClassifier
```

```
from sklearn.metrics import confusion_matrix,
    accuracy_score
from sklearn.model_selection import cross_val_score,

classifier = KNeighbours.Classifier (n_neighbours = 3)
classifier.fit (X_train, Y_train)

Y_pred = classifier.predict (x_test)
accuracy = accuracy_score (y_test, y_pred) * 100
print ("Accuracy" str (round (accuracy, 2)) + '%')
```

O/p = Accuracy = 96.67%

1) SVM

import
import
from s
from sk
from s
from s
from

iris =
X = iris
Y = iris

X_train

scaler
X_train
X_test
svm =
svm.

y_pred
accura
print(

o/p =

Lab-7

1) SVM

•)

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.model selection import train_test_split
from sklearn.preprocessing import Standardscaler
from sklearn.svm from SVC
from sklearn metrics import accuracy_score

iris = datasets.load_iris()
X = iris.data [:, :2]
Y = iris.target

X_train, X test, Y_train, Y_test train_test_split.
                    (X, Y, test_size = 0.3, state=42)

scaler = StandardScaler()
X_train = scaler.fit_transform(X train)
X_test = scaler.transform(X_test)
svm = SVC(kernel='linear', random state=42)
svm.fit(X_train, y_train)

y_pred = svm.predict(X test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

o/p = 0.977 / 97.7%

2) PCA

```
-> import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.model selection import StandardScaler
from sklearn.preprocessing import PCA

iris= dataset.load iris ()
X= iris.data()
Y= iris.target()

X train, Y test, X test, Y train= tt_split (test size = 0.3,
        state = 42)

scalar= StandardScalar()
X train std = scalar.fit_transform(X train)
X test std = scaler.transform (X-test)

pca= PCA(n components = 2)

X train.pca = pca.fit_transform (X train_std)

X test_pca = pca.transform (X test_std)

random forest = randomForestClassifier (n_estimators = 100,
        random_state = 42)

random_forest.fit (x_train pca, y train)

y pred= random forest.predict (x_test_pca)
accuracy= accuracy_score (y_test, y_pred)
```

```
print ('

d/b = A = c

3) K-mean

:) import p
import r
import s
import
import sk
import
import s
import

X, Y = lo
KMean =
kMean -
KMean -
```

MY PAGES
Date __/__/__

print ("Accuracy", accuracy)

o/p = A = 95.5

3) K-mean Cluster

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib
import sklearn.datasets
import load_iris
import sklearn.cluster
import kMeans.

X, Y = load_iris (return_X_Y = True)
KMean = kMeans (n_clusters = 3, random_state = 2)
kMean.fit(x)
KMeans.cluster_center
```

23-05-M

Lab 8

1) Implement Random forest method

->

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

titanic_data = pd.read_csv('url')

print(titanic_data())

titanic_data.drop(['Name', 'Ticket', 'Cabin', 'Embarked'], axis=1,
    inplace=True)
titanic_data['Sex'] = titanic_data['Sex'].map({'male': 0, 'female':
titanic_data.fillna(titanic_data.mean(), inplace=True)
X = titanic_data.drop('Survived', axis=1)
Y = titanic_data('Survived')

X_train, X_test, Y_train, ytest = train_test_split(X, y, test_size=0.2,
    random_state=42)

random_forest = RandomForestClassifier(n_estimators=100, random_state=
    42)
random_forest.fit(X_train, Y_train)

y_pred = random_forest.predict(X_test)

accuracy = accuracy_score(ytest, ypred)
print("Accuracy:", accuracy)
```

o/p: 80%

2) imp
-> imp
imp
fro
fro
fro
fro

data
df =
df
prin
X = d
Y = d
X_te
rand

ada
adaB

accu
prin
o/p = 9

MY PAGE'S
Date __/__/__

2) Implement boosting Method.

=) import numpy as np
import pandas as pd
from sklearn.datasets import load wine
from sklearn.model selection import train test split
from sklearn.ensemble import AdaBoost Classifier
from sklearn.metrics import accuracy score.

data = loadwine()
df = pd.Dataframe(data.data, columns = data.feature_names)
df {'target'] = data.target
print(df.head())
X = df.drop(['target', axis = 1)
Y = df[['target']
x train, x test, y train, y test = traintest_split(x, y, test_size=0.2,
random state = 42)

adaBoost = AdaBoost Classifier(n estimators=50, random_state=42)
adaBoost.predict(xtest)

accuracy = accuracy score(ytest, y pred)
print("Accuracy", accuracy)
o/p = 92%

30.05.2024