INDEX

NAME: Stuti Uniyal STD.: ____ SEC.: D ROLL NO.: ____ SUB.: Machine Learning

S. No.	Date	Title	Page No.	Teacher's Sign / Remarks
1	21/3/24	Reading pandas		dy 4/4/24
2	28/3/24	End to End project		dy 18/4/24
3	4/4/24	Linear Regression		
4	18/4/24	~~Logistic Reg~~ Decision Tree		
5	25/4/24	Logistic Regression		dy 25/4/24
6	9/05/24	KNN classifier		
7	9/05/24	SVM		dy 9/5/24
8	23/5/24	ANN		
9	23/5/24	Random Forest (9a)		dy 23/5/24
9	23/5/24	AdaBoost (9b)		
10	30/5/24	KMeans		
11	30/5/24	Principal Component Analysis		dy 30/5/24

WEEK - 1
PANDAS (Reading CSV)

```
import pandas as pd
df2 = pd.read_csv("/content/sample_data/california_housing_test.csv")
df2.head()
```

OUTPUT:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms
0	-122.05	37.37	27.0	3885.0	661.0

	population	households	median_income	median_house_value
0	1537.0	606.0	6.6085	344700.0

```
df2.to_csv("hi.csv")
```

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

OUTPUT:

	sepal_length_in_cm	sepal_width_in_cm	petal_length_in_cm
0	5.1	3.5	1.4

	petal_width_in_cm	class
	0.2	Iris-setosa

4/4/24

WEEK 2

1. Performance Measure
2. Get the Data

2.1 Download the data

```
import os
import tarfile
import urllib

DOWNLOAD_ROOT = "https://raw.githubusercontent.com/ageron/handson-ml2/master/"
HOUSING_PATH = os.path.join("data", "01")
HOUSING_URL = DOWNLOADS_ROOT + "dataset/housing/housing.tgz"

def fetch_housing_data(housing_url = HOUSING_URL,
                       housing_path = HOUSING_PATH):
    os.makedirs(name = housing_path, exist_ok = TRUE)
    tgz_path = os.path.join(housing_path, "housing.tgz")
    urllib.request.urlretrieve(url = housing_url, filename =
    housing_tgz = tarfile.open(name = tgz_path) tgz_path)
    housing_tgz.extractall(path = housing_path)
    housing_tgz.close()

fetch_housing_data()

import pandas as pd
def load_housing_data(housing_path = HOUSING_PATH):
    data_path = os.path.join(housing_path, "housing.csv")
    return pd.read_csv(data_path)
```

2.2 Take a Quick Look at the Data Structure

```
housing = load_housing_data()
housing.head()
housing.info()
```

- housing["ocean_proximity"].value_counts()
- housing.describe()

2.3 PLOTTING GRAPHS

- import matplotlib.pyplot as plt
- import seaborn as sns
- housing.hist(bins=50, figsize=(20,15))
- plt.show()

2.4 Create a Test set

```
import numpy as np
def split_train_test(data, test_ratio=0.2):
    shuffled_indices = np.random.permutation(len(data))
    test_set_size = int(len(data)* test_ratio)
    test_indices = shuffled_indices[:test_set_size]
    train_indices = shuffled_indices[test_set_size:]
    return data.iloc[train_indices], data.iloc[test_indices]

train_set, test_set = split_train_test(data=housing)
len(train_set), len(test_set)

from zlib import crc32
def test_set_check(identifier, test_ration=2):
    total_size = 2** 32
    hex_repr = crc32(np.int64(identifier)) & 0xffffffff
    in_test = hex_repr < (test_ratio * total_size)
    return in_test

[test_set_check(i) for i in range(10)]
```

2.5 Co-ordinates to Index

```
def from_z_to_N(z):
    if z >= 0:
        n = 2* z
```

```
    else:
        n = -2 * z - 1
    return n

def cantor_pairing(n1, n2):
    n = ((n1 + n2) * (n1 + n2 + 1))/2) + n2
    return n
def lat_lon_to_index(lat, lon):
    lat, lon = int(lat * 100), int(lon * 100)
    lat, lon = from_z_to_N(lat), from_z_to_N(lon)
    index = cantor_pairing(lat, lon)
    return np.int64(index)
housing['id'] = housing.apply(lambda row: lat_lon_to_
                                          index
    (row['latitude'], row['longitude']), axis=1)

housing['id'].value_counts()
```

28/3/24

discover & Visualize the Data to Gain Insights

```
strat_train_set.shape , strat_test_set.shape
((16512,10),(4128,10))
strat_test_set.reset_index(), to_feather(fname='data/
                                    01/strat_test_set_f')

housing = strat_train_set.copy();
housing.shape
```

Looking for correlations

```
corr_matrix = housing.corr()
corr_matrix['median_house_value'].sort_values(ascending=False)
from pandas.plotting import scatter_matrix
attributes = ['median_house_value', 'median_income', 'total_
                rooms',
              'housing_median_age']
scatter_matrix(frame = housing[attributes], figsize=(12,8))
plt.show()
```

WEEK 3 (PART-B)

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from pandas.core.common import random_state
from sklearn.linear_model import LinearRegression

#Get Dataset
df_sal = pd.read_csv('/content/drive/MyDrive/Salary_Data.csv')
df_sal.head()

#Describe data
df_sal.describe()
#Data distribution
plt.title('Salary Distribution Plot')
sns.distplot(df_sal['Salary'])
plt.show()
plt.scatter(df_sal['YearsExperience'], df_sal['Salary'],
            color='lightcoral')

plt.title('Salary vs Experience')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.box(False)
plt.show()
#splitting variables
x = df_sal.iloc[:, :1] # independent
y = df_sal.iloc[:, 1:] # dependent
# splitting dataset into test/train
x_train, x_test, y_train = train_test_split(x, y, test_size=0.2,
                                            random_state=0)
```

```
# Regression model
regressor = LinearRegression()
regressor.fit(X_train, y_train)
# prediction result
y_pred_test = regressor.predict(X_test)
y_pred_train = regressor.predict(X_train)
# prediction on training set
plt.scatter(X_train, y_train, color='lightcoral')
plt.plot(X_train, y_pred_train, color='firebrick')
plt.title('Salary vs Experience (Training set)')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.legend(['X_train/Pred(y_test)', 'X_train/y_Train'],
 title='Sal/Exp', loc='best', facecolor='white')
plt.box(False)
plt.show()
# prediction on test set
plt.scatter(X_test, y_test, color='lightcoral')
plt.plot(X_train, y_pred_train, color='firebrick')
plt.title('Salary vs Experience (Test set)')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.legend(['X_train/Pred(y_test)', 'X_train/y_train'],
title='Sal/Exp', loc='best', facecolor='white')
plt.box(False)
plt.show()
# Regressor coefficients and intercepts.
print(f'Coefficient: {regressor.coef_}')
print(f'Intercept: {regressor.intercept_}')
```

OUTPUT

Coefficient: [[9318.57512673]]

Intercept: [6780.09915063]

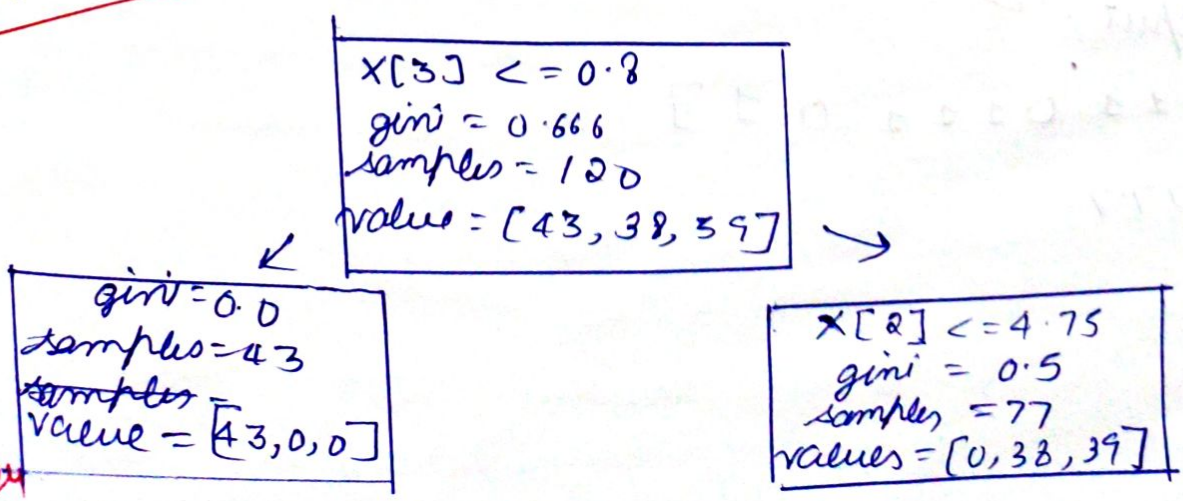
18/4/24

WEEK-4

```
import pandas
import matplotlib
from sklearn.datasets import load_iris
iris = load_iris()
x = iris.data
y = iris.target
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)

from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier(max_depth=3, min_samples_leaf=10,
                             random_state=1)
clf.fit(x_train, y_train)
y_pred = clf.pred(x_test)
from sklearn.metrics import accuracy_score
print(accuracy_score(y_test, y_pred))
from matplotlib.pylab import rcParams
rcParams['figure.figsize'] = 80,50
from sklearn.tree import plot_tree
plot_tree(clf)
```

OUTPUT
0.9666



18/4/24

LAB-5

```
import pandas as pd
from matplotlib import pyplot as plt
%matplotlib inline

df = pd.read_csv("/content/drive/MyDrive/insurance_data.csv")
df.head()
plt.scatter(df.age, df.bought_insurance, marker='+',
            color='red')
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
                    df[['age']], df.bought_insurance,
                    train_size=0.7)

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

output:

```
[0 1 1 0 1 1 1 0 1]
0.777
```

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(x_train, y_train)
print(model.predict(x_test))
model.score(x_test, y_test)
```

OUTPUT:

0.58433

```
import math
def sigmoid(x):
    return 1/(1+math.exp(-x))
def prediction_function(age):
    z = 0.042 * age - 1.53
    y = sigmoid(z)
    return y
age = 35
print(prediction_function(age))
age = 43
print(prediction_function(age))
```

OUTPUT:

0.485004

0.56956

dtj 25/11/24

LAB-6

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.svm import train_test_split
from sklearn.model_selection import confusion_matrix
from sklearn.metrics import accuracy_score, classification_report
from sklearn.neighbors import KNeighborsClassifier

colnames = ["sepal_length_in_cm", "sepal_width_in_cm",
"petal_length_in_cm", "petal_width_in_cm", "class"]
dataset = pd.read_csv("https://archive.ics.uci.edu/ml/
machine-learning-databases/iris/iris.data",
    header = None, names = colnames)

dataset.head()

dataset = dataset.replace({"class": {"Iris-setosa": 1,
    "Iris-versicolor": 2, "Iris-virginica": 3}})

dataset.head()

iris = sns.load_dataset('iris')
sns.set_style("whitegrid")
sns.FacetGrid(iris, hue="species", height=6).
    map(plt.scatter, 'sepal_length', 'petal_length').
        add_legend()

X = dataset.iloc[:, :-1]
y = dataset.iloc[:, -1].values

X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.25, random_state=0)
```

```
classifier = SVC(kernel = 'linear', random_state=0)
classifier.fit(x_train, y_train)
y_pred = classifier.predict(x_test)

cm = confusion_matrix(y_test, y_pred)
print(cm)
[[13  0  0]
 [ 0 15  1]
 [ 0  0  9]]

y_pred = classifier.predict(x_test)
print(y_pred)
print(classification_report(y_test, y_pred, target_
                            names=['1', '2', '3']))
```

OUTPUT

accuracy 0.97

```
classifier_2 = KNeighborsClassifier(n_neighbors = 2)
classifier2.fit(x_train, y_train)
y_pred = classifier2.predict(x_test)

cm = confusion_matrix(y_test, y_pred)
print(cm)
[[13  0  0]
 [ 0 15  1]
 [ 0  0  9]]

y_pred = classifier2.predict(x_test)
print(classification_report(y_test, y_pred, target_names=
                            ['1', '2', '3']))
```

accuracy 0.97

9/5/24

LAB 8

Implementation of ANN using Back propogation for gerven values

```
import numpy as np
X = np.array([[0,9], [1,5], [3,6]), dtype = float))
y = np.array([[92], [86], [89]), dtype = float)
X = X/np.array(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,
                               hiddenlayer_neurons))
bh = np.random.uniform(size = (1, hiddenlayer_neurons))
wout = np.random.uniform(size = hiddenlayer_neurons,
                         output_neurons))
bout = np.random.uniform(size = (1, output_
                                 neurons))

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 = derivatives_sigmoid(output)
d_output = EO * outgrad
EH = d_output . dot(wout.T)

hiddengrad = derivatives_sigmoid(hlayer_act)
d_hidden_layer = EH * hiddengrad

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

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

OUTPUT

```
Input:
[[0.6667   1.
  0.3334   0.556
  1.       0.6667]]

Actual output:
[[0.92]
 [0.86]
 [0.89]]

Predicted output:
[[0.935 ]
 [0.923 ]
 [0.9339]]
```

PROGRAM 9a

RANDOM FOREST ALGORITHM

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
df = pd.read_csv("/content/drive/MyDrive/Iris.csv")
df.head()
y = df["species"]
x = df.drop(["species"], axis = 1)
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split
                    (x, y, test_size = 0.3, random_state = 0)
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(n_estimators = 100)
clf.fit(x_train, y_train)
y_pred = df.predict(x_test)
from sklearn.metrics import accuracy_score
score = accuracy_score(y_pred, y_test)
print(f"Accuracy : {score}")
```

OUTPUT

Accuracy : 1.0

AdaBoost (with default parameters)

```
from sklearn.ensemble import AdaBoostClassifier
adb = AdaBoostClassifier()
adb_model = adb.fit(x_train, y_train)
y_pred = adb_model.predict(x_test)
score = accuracy_score(y_pred, y_test)
print(f"Accuracy: {score}")
```

Accuracy: 0.977

AdaBoost (with Hyper parameters)

```
from sklearn.linear_model import LogisticRegression
lrmodel = LogisticRegression()
adbhp = AdaBoostClassifier(n_estimators=150,
                           estimator=lrmodel,
                           learning_rate=1)
model = adbhp.fit(x_train, y_train)
y_pred = model.predict(x_test)
score = accuracy_score(y_pred, y_test)
print(f"Accuracy: {score}")
```

OUTPUT

Accuracy: 1.0

28/5/24

LAB 10 KMEANS

```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans
import pandas as pd
import numpy as np

iris = datasets.load_iris()
X = pd.DataFrame(iris.data)
X.columns = ['Sepal_Length', 'Sepal_Width', 'Petal_Length',
             'Petal_Width']

y = pd.DataFrame(iris.target)
y.columns = ['Targets']

model = KMeans(n_clusters=3)
model.fit(X)
plt.figure(figsize=(14,14))
colormap = np.array(['red', 'lime', 'black'])
plt.subplot(2,2,1)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[
            y.Targets], s=40)

plt.title('Real Clusters')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')

plt.subplot(2,2,2)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[
            model.labels_], s=40)

plt.title('K-Means Clustering')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
```

PRINCIPAL COMPONENT ANALYSIS

LAB-11

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

from sklearn.datasets import load_breast_cancer
cancer = load_breast_cancer()

cancer.keys()
print(cancer['DESCR'])

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler();
scaler.fit(df)
scaled_data = scaler.transform(df)
from sklearn.decomposition import PCA
pca = PCA(n_components = 2)
pca.fit(scaled_data)
PCA(copy = True, n_components = 2, whiten = False)
x_pca = pca.transform(scaled_data)
scaled_data.shape
x_pca.shape

plt.figure(figsize = (8,6))
plt.scatter(x_pca[:,0], x_pca[:,1], c = cancer['target'],
plt.xlabel('First principal component')
plt.ylabel('Second principal component') cmap = 'plasma')
```

Aditya 30/5/24