ARUN'S

INDEX

Name Shashank. M.S. Subject Machine Learning

Standard 3rd Year BE Section CSE 6D section Roll No. 1BM21CS201

~~School~~ College B.M.S. College of Engineering

S. No.	Date	Title	Page No.	Teacher's Sign
1	21/03/24	Import & Export csv files LAB 1	1-2	10
2	28/03/24	LAB 2	3-7	10
3	04/04/24	LAB 3	8-9	10
4	18/04/24	LAB 4	10-12	10
5	25/04/24	LAB 5	13-14	10
6	09/05/24	LAB 6	15-17	10
7	23/05/24	LAB 7	18-21	10
8	30/05/24	LAB 8	22-24	10

Machine Learning

Lab - 1

1) Write a python program to import and export data using pandas library function.

Reading data from URL

```
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/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()
```

Output:

	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

class

Iris-Setosa

importing & displaying data

```
import pandas as pd
iris_data = pd.read_csv(r"C:\Users\Admin\Downloads\iris.data")
iris_data.head
```

	5.1	3.5	1.4	0.2	Iris setosa
0	4.9	3.0	1.4	0.2	Iris setosa
1	4.7	3.2	1.3	0.2	Iris setosa

Exporting file to current working directory

```
iris_data.to_csv("cleaned_iris_data.csv")
```

24/5/24

Machine Learning

Lab – 2

2> Get 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 = DOWNLOAD_ROOT + "datasets/housing/housing.tgz"
```

```
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)
```

```
housing = load_housing_data()
```

```
housing.head()
housing.info()
```

3) Discover and Visualize the Data to gain Insights

```
strat_train_set.shape, strat_test_set.shape

strat_test_set.reset_index().to_feather(fname = "data/01/strat_test_s

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

Now for visualizing Geographical Data

```
housing.plot(kind = 'scatter', x = 'longitude', y = 'latitude')
    plt.show()

housing.plot(kind = 'scatter', x = 'longitude', y = 'latitude', alpha = 0.
    plt.show()
```

Now for correlation

```
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']

scatter_matrix(frame = housing[attributes], figsize = (12,8))
    plt.show()
```

housing ["rooms_per_household"] = housing ["total rooms"] / housing ["households"]

4) Prepare Data for Machine learning Algorithms

from sklearn.impute import SimpleImputer

imputer = SimpleImputer (strategy = "median")
housing_num = housing.drop ("ocean_proximity", axis = 1)
imputer.fit (housing_num)
imputer.statistics_ housing_num.median().values

from sklearn.preprocessing import OrdinalEncoder
ordinal_encoder = OrdinalEncoder()
ordinal_encoder.categories

attr_adder = CombinedAttributeAdder [add_bedroom], per_room = False)

from sklearn.compose import column transformation
num_attributes = housing_num.columns.to list()

5) Select and Train a Model

- from sklearn.linear_model import LinearRegression

```
lin_reg = LinearRegression()
lin_reg.fit(x=housing_prepared,
            y=housing_labels)
```

```
housing_predictions = lin_reg.predict(housing_prepared)
```

```
lin_mse = mean_squared_error(housing_label,
                             housing_predictions)
```

```
lin_rmse = np.sqrt(lin_mse)
lin_rmse
```

```
tree_reg = DecisionTreeRegressor()
```

```
tree_reg.fit(x= housing_prepared, y= housing_labels)
```

```
forest_reg = RandomForestRegressor()
```

```
forest_reg.fit(x = housing_prepared, y = housing_labels)
```

6Y Fine Tune the model

grid_search.best_params

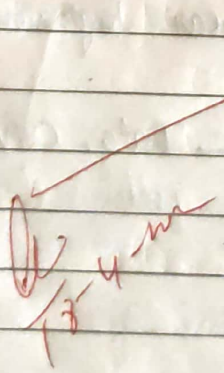
grid_search.best_estimator_
cat_encoder = full_pipeline.named_transformers_["cat"]

final_model = grid_search.best_estimator

X_test_prepared = full_pipeline.transform
(X = X_test)

final_rmse = np.sqrt(final_mse)
final_rmse

squared_errors = (y_test - final_predictions) ** 2



LAB - 02

Simple Linear Regression and Multiple Linear Regression

```
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

df_sal = pd.read_csv(r"c:\Salary_Data.csv")
df_sal.head()
df_sal.describe()

plt.title("Salary Distribution Plot")
sns.distplot(df_sal["Salary"])
plt.show()

x = df_sal.iloc[:, :1]
y = df_sal.iloc[:, 1:]

x_train, x_test, y_train, y_test = train_test_split
                    (x,y , test_size = 0.2, random_state=0)
```

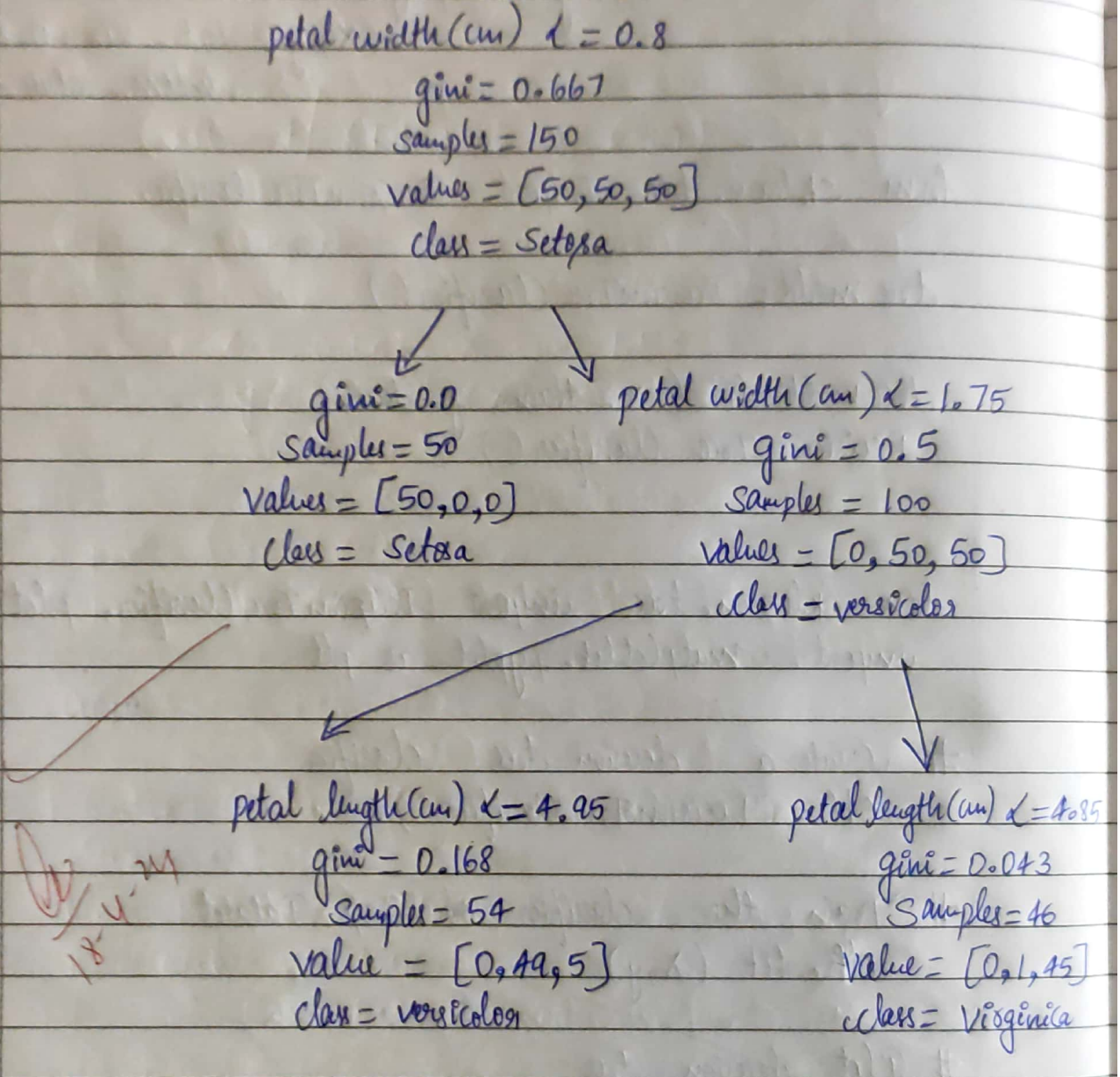
```
regressor = LinearRegression()
regressor.fit(x_train, y_train)

y_pred_test = regressor.predict(x_test)
y_pred_test = regressor.predict(x_train)

print('Coefficient: ', regressor.coef_)
print('Intercept: ', regressor.intercept_)

df_start = pd.read_csv('C:\50_Startups.csv')
df_start.head()
df_start.describe()

sns.distplot(df_start['Profit'])
plt.show()

regressor = LinearRegression()
regressor.fit(x_train, y_train)

y_pred = regressor.predict(x_test)

np.set_printoptions(precision=2)

result = np.concatenate(y_pred.reshape(len(y_pred),1),
                        y_test.reshape(len(y_test),1),1)

result
```

WEEK-4

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
import seaborn as sns

from sklearn.datasets import load_iris

# Load the Iris dataset
iris_data = load_iris()

# Display the dataset description
print(iris_data.DESCR)

# Extract independent features (X) and dependent feature (Y)

X = iris_data.data
y = iris_data.target

# Display the shapes of X and Y

print("Shape of X:", X.shape)
print("Shape of y:", y.shape)
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split
                                   (X, y, test_size=0.33,
                                    random_state=42)

from sklearn.tree import DecisionTreeClassifier

tree_model = DecisionTreeClassifier()

tree_model.fit(X_train, y_train)
DecisionTreeClassifier()

from sklearn.tree import DecisionTreeClassifier, plot_tree
import matplotlib.pyplot as plt

# Create a decesion tree classifier
clf = DecisionTreeClassifier()

# Train the classifier on Iris Dataset
clf.fit(X,y)

# Plot decesion tree

plt.figure(figsize=(12,8))
plot_tree(clf, filled=True, feature_names=iris.data.
          feature_names, class_names=iris.target_names)
plt.show()
```

Output




```
from sklearn.model_selection import cross_val_score

scores = cross_val_score(clf, X, y, cv=5)
accuracy = scores.mean()
print("Mean Accuracy:", accuracy)
```

o/p Mean Accuracy: 0.9666

graphikon
Date 25/01/24
Page

WEEK-5

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

from sklearn.model_selection import
                    traintest_split

from sklearn.linear_model import Logistic Regression

df = pd.read_csv ("insurance_data.csv")
print (df.head ())

X_train, X_test, y_train, y_test = train_test_split
    (df [['age']]), df.bought_insurance, test_size = 0.2

    print ("X_test:")
    print (X_test)

    model = Logistic Regression ()
    model.fit (X_train, y_train)

    y_pred = model.predict (X_test)
       print (y_pred)

    print (model.predict_proba (X_test))
    print (model.score (X_test, y_test))
```

Output:

Accuracy = 0.5

```
import math

def sigmoid(z):
    return 1/(1+math.exp(-z))

def predi(age):
    z = 0.042 * age - 1.53
    y = sigmoid(z)
    return y

print(predi(35))
print(predi(43))
```

Output:

Prediction: array([1, 0, 1, 0, 0, 0, 0, 1, 0])
Score: 0.8888
Linear Reg Score: 0.584321
Predictions: 0.485
0.5685

graphikaa
Date 09/05/24
Page

WEEK-6

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
plt.style.use('ggplot')

df = pd.read_csv('/content/diabetes.csv')

df.head()
df.shape

x = df.drop('Outcome', axis=1).values
y = df['Outcome'].values

from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split
                                   (x, y, test_size = 0.4,
                                    random_state = 42, stratify = y)

from sklearn.neighbors import KNeighborsClassifier

neighbors = np.arange(1,9)

train_accuracy = np.empty (len(neighbors))
test_accuracy = np.empty (len(neighbors))
```

```
for i, k in enumerate(neighbors):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    train_accuracy[i] = knn.score(X_train, y_train)
    test_accuracy[i] = knn.score(X_test, y_test)

plt.title('k-NN Varying number of neighbors')
plt.plot(neighbors, test_accuracy, label='Testing Accuracy')
plt.plot(neighbors, train_accuracy, label='Training Accuracy')
plt.legend()
plt.xlabel('Number of neighbors')
plt.ylabel('Accuracy')
plt.show()

plt.scatter(neighbors, train_accuracy, color='k')
plt.scatter(neighbors, test_accuracy, color='g')
plt.show()

knn = KNeighborsClassifier(n_neighbors=7)
knn.fit(X_train, y_train)
knn.score(X_test, y_test)
```

Accuracy: 0.73051948 (73.05 %)

SVM Model

```
from sklearn import datasets

cancer = datasets.load_breast_cancer()

print (cancer.target)

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split (cancer.data,
                                   cancer.target, test_size = 0.3,
                                   random_state = 109)

from sklearn import svm

    clf = svm.SVC (kernel = 'linear')

    clf.fit (X_train, y_train)

    y_pred = clf.predict (X_test)

  from sklearn import metrics

  print ("Accuracy:" metric.accuracy_score (y_test, y_pred))
```

Output:

Accuracy: 0.9649 (96.49%)

∞

graphykon
Date 23/05/21
Page

WEEK - 7

ANN Program

```
db = np.loadtxt ("../input/ duke ....")
np.random.shuffle (db)

y = db [:, 0]

x = np.delete (db, [0], axis = 1)

X_train, X_test, y_train, y_test = train_test_split
                                   (x, y, test_size = 0.1)

print (np.shape (X_train), np.shape (X_test))

hidden_layer = np.zeros (72)
weights = np.random.random ((len (x[0], 72)))
Output_layers = np.zeros (2)

hidden_weights = np.random.random ((len (x[0], 72)))

def sum_function (weights, index_col, x):
    result = 0

    for i in range (0, len(x)):

    result += x[i] * weights [i][index_locked_col
    return result
```

```
def - activate_layer (layer, weights, x):

    for il in range (0, len (layer)):

    layer [i] = 1.7159 * np. tanh (2.0 * sum_function
                                   (weights, i, x) / 3.0)

def back - propagation (hidden_layer, output_layer,
                        one_hot_encoding, learning_rate, x):

    output_derivative = np. zeroes (2)
    output_gradient = np. zeroes (2)

for i in range (0, len (output_layer)):

        output_derivative [i] = (1.0 - output layer [i]) *
                                output_layer [i])

    hidden_derivative = np. zeroes (72)

    hidden_gradient = np. zeroes (72)

  for i in range (0, len (output_layer)):
                      hidden

      hidden_derivate [i] = (1.0 - hidden_layer [i]) *
                            (1.0 + hidden_layer [i])

  for i in range (0, len (hidden_layer)):

    Sum = 0
```

```
for j in range (0, len (output_gradient)):
        sum += output_gradient [j] * hidden_weights [i][j]
    hidden_gradient [i] = sum * hidden_derivate [i]
recalculate (learning_rate, weights, hidden_gradient, x)
```

Random Boosting and Ada Boosting

```
dataset = pd.read_csv ("Health_data.csv")
corr_matrix = input_x.corr().abs()
upper = corr_matrix where (np.triu (np.one
        (corr_matrix.shape), k=1)
        as type (np.bool))
to_drop = [column for column in
        upper.columns if any
        (upper [column] > 0.95)]
print (to_drop)
X_train, X_test, y_train, y_test = train_test_split
                                   (X, y, test_size=0.1)
n_classes=7
n_estimators= 100
```

RANDOM_SEED = 13

names = ['Random Forest Classifier', 'Ada Boost Classifier']

models = [RandomForestClassifier(n_estimators=n_estimators),
AdaBoostClassifier(DecisionTreeClassifier(max_depth=None),
n_estimators=n_estimators))]

for counter, model in enumerate(models):

model.fit(X_train, y_train)

y_pred = model.predict(X_test)

Accuracy Random Forest Classifier : 0.768439 (appx 76.8%)

Accuracy Ada Boost Classifier : 0.7341337 (appx 73.4%)

WEEK-8

Program 10

K-Means Algorithm

```
iris = pd.read_csv("..... Iris.csv")

x = iris.iloc[:, [0, 1, 2, 3]].values

iris_outcome = pd.crosstab(index = iris["Species"]
                           columns = "count")

iris_outcome

sns.FacetGrid(iris, hue = "Species").map
(sns.displot, "petal_length").add_legend()

sns.FacetGrid(iris, hue = "Species").map
(sns.displot, "petal_width").add_legend()

sns.FacetGrid(iris, hue = "Species").map
(sns.displot, "sepal_length").add_legend()
plt.show()

sns.set_style("whitegrid")
```

```
sns.pairplot(iris, hue="species", size=3);
plt.show()

kmeans = KMeans(n_cluster=3, init='k-means++',
                max_iter=300, n_init=10,
                random_state=0)

y_kmeans = kmeans.fit_predict(X)
```

Program 11

PCA

```
df = pd.read_csv("..... data.csv")

df_features = df.drop(['diagnosis'], axis=1)

from sklearn.preprocessing import StandardScaler

Standardized = StandardScaler()

Standardized.fit(df_features)

Scaled_data = Standardized.transform(df_features)

from sklearn.decomposition import PCA
```

```
pca = PCA (n_components = 3)
pca.fit (scaled_data)

x_pca = pca.transform (scaled_data)
scaled_data.shape

x_pca.shape

def diag (x):

    if x == 'M':
        return 1

    else: return 0

df_diag = df ['diagnosis'].apply (diag)

x_pca [:1]

df_pc = pd.DataFrame (pca.components
                      columns = df_features.columns)

plt.figure (figsize = (15,8))

sns.heatmap (df_pc, cmap = 'viridis')

plt.title ('Principal component correlation
           with the features')
```