INDEX

NAME: Shravani STD.: _____ SEC.: _____ ROLL NO.: _____ SUB.: ML LAB

S. No.	Date	Title	Page No. Mark	Teacher's Sign / Remarks
1	21/03/2024	LAB1	10	
2	28/03/2024	LAB 2	10	
3	04/04/2024	LAB 3	10	
4	18/04/2024	LAB4	10	
5	25/04/2024	LAB5	10	
6	09/05/2024	LAB6	10	
7	09/05/2024	LAB6		
8	23/05/2024	LAB7 - 8	10	
8 9	23/05/2024	LAB7 - 9a 9b	10	
10	30/05/2024	LAB8 - 10	10	
11	30/05/2024	LAB8 - 11	10	

LAB – 1 21/03/2024

Importing and Exporting a dataset

```
import pandas as pd
url = "http s://..."
df = pd.read_csv(url, columns = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'class'])
df.head()
```

	sepal_length	sepal_width	petal_length	petal_width	cla
0	5.1	3.3	1.4	0.2	Iris setos
1	4.9	3.0	1.4	0.2	Iris-setosa
⋮					

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

21/3/24

28/03/2024

LAB - 2
End To end Project

1 Look at the big picture : Performance Measure
2 Get The Data

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

pd.cut

```
housing['income_cat'] = pd.cut(x =
        housing['median_income'], bins =
    [0, 1.5, 3, 4.5, 6, np.inf],
     labels =[1,2, 3, 4,5])
housing['income_cat'].hist()
```

LAB-8 30/05/2024

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.
      distplot, "petal_length").add_legend()
  sns.FacetGrid(iris, hue = "species").map(sns.
      distplot, "petal_width").add_legend()
  sns.FacetGrid(iris, hue = "species").map(
      sns.distplot, "sepal_length").add_legend()
  plt.show()
  sns.boxplot(x = "species", y = "petal_length",
              data = iris)
  plt.show()
  sns.set_style("whitegrid")
  sns.pairplot(iris, hue = "species", size = 3);
  plt.show()
  from kmeans = KMeans(n_clusters = 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("...
df_features = df.drop(
from sklearn.preprocessi
standardized = standa
standardized.fit(df_f
scaled_data = standar

from sklearn.decom
pca = PCA(n_compon
pca.fit(scaled_data
x_pca = pca.transfo
scaled_data.shape
x_pca.shape
def diag(x):
    if x == 'M'
    return
    else: return
df_diag = df['di
x_pca[:,1]
df_pc = pd.DataF
    columns
plt.figure(figsize
sns.heatmap(df_
plt.title('Principa
```

```
imputer.statistics_
housing_num.median().values
```

- ```
  from sklearn.preprocessing import OrdinalEncod
  ordinal_encoder = OrdinalEncoder()
  ordinal_encoder.categories_
  ```
- ```
  attr_adder = CombinedAttributesAdder(add_bedro
                                       per_room = False)
  housing_extra_attribs = attr_adder.transform(
                                       housing.values)
  ```
- ```
  from sklearn.compose import ColumnTransform
  num_attribs = housing_num.columns.tolist()
  ```

5 Select and Train a Model:

- ```
  from sklearn.linear_model import LinearRegres
  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_labe
                                       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)
  forest_reg.fi
  ```
  ```

6 Fine - Tune Your 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_predictio
  **2

7 Launch, Monitor and Maintain your system

4/4/24

LAB-3 04/04/2024

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
regressor = LinearRegression()
regressor.fit(x_train, y_train)
y_pred_test = regressor.predict(x_test)
y_pred_train = regressor.predict(x_train)
```

```
print(f'coefficient : {regressor.coef_}')
print(f'Intercept: {regressor.intercept_}')

df_start = pd.read_csv(r'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
```

LAB-4 18/04/2024

Use an appropriate data set for building The decision tree (ID3) and apply this knowledge To classify a new sample

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

import sklearn.datasets as datasets
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score,
    roc_auc_score, roc_curve
from sklearn.tree import plot_tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report

url = "....."
df = pd.read_csv(url, names = ['sepal length(cm)',
                               ....])

df.head()

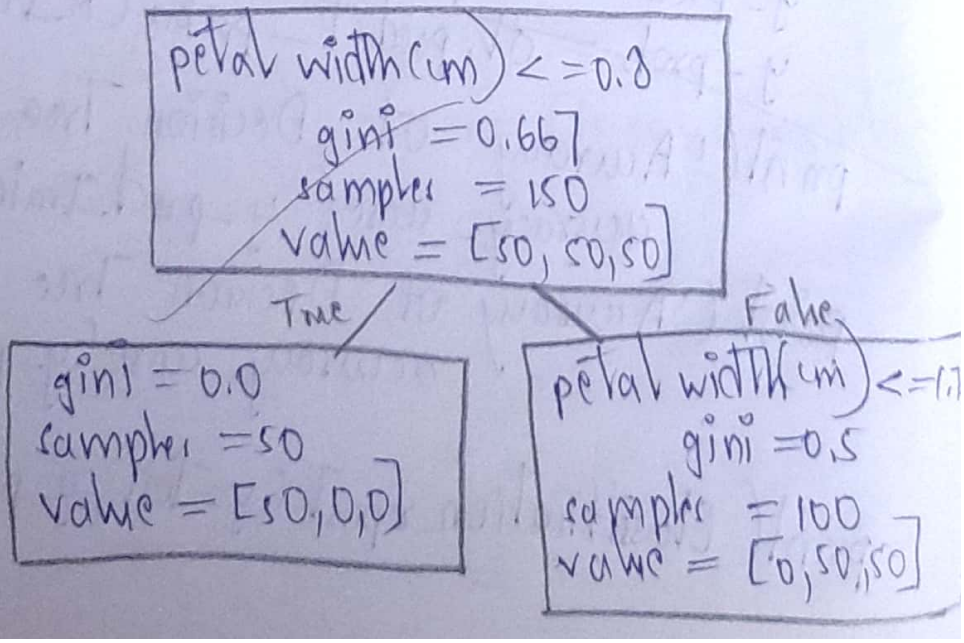
x = df.drop("species", axis =1)
y = df["species"]
x_train, x_test, y_train, y_test = train_test_
```

```
split(x, y, test_size = 0.3, random_state = 1)
dt = DecisionTreeClassifier(max_depth = 3,
                            min_samples_leaf = 10, random_state
                            = 1)
dt.fit(x, y)
from IPython.display import Image
from sklearn.tree import export_graphviz
!pip install pydotplus
import pydotplus
features = x.columns
dot_data = export_graphviz(dt, out_file=
    None, feature_names = features)
graph = pydotplus.graph_from_dot_data(
                    dot_data)
Image(graph.create_png())
dt = DecisionTreeClassifier(random_state = 1)
dt.fit(x_train, y_train)
y_pred_train = dt.predict(x_train)
y_pred = dt.predict(x_test)
y_prob = dt.predict_proba(x_test)
print("Auracy of Decision Tree - Train:"
      accuracy_score(y_pred_train, y_train))
print("Accuracy of Decision Tree - Test:",
      accuracy_score(y_pred, y_test))
print(Classification_report(y_test, y_pred))
```

```
dt2 = DecisionTreeClassifier(random_state=1)
params = {'max_depth': [2,3,4,5]
          'min_samples_split': [2,3,4,5]
          'min_samples_leaf': [1,2,3,4,5]}
gsearch = GridSearchCV(dt2, param_grid=
                       params, cv=3)
gsearch.fit(x,y)
gsearch.best_params_

dt2 = DecisionTreeClassifier(**gsearch.best_
      params_, random_state=1)
dt2.fit(x_train, y_train)
y_pred_train = dt2.predict(x_train)
y_prob_train = dt2.predict_proba(x_train)
                [:,1]
y_pred = dt2.predict(x_test)
y_prob = dt2.predict_proba(x_test)[:,1]
```

Output:



random_state=1

[2,3,4,5]

[2,3,4,5]

[1,2,3,4,5]]

, param_grid=

, cv=3)

**gsearch.best_

=1)

t(X_train)

t_proba(X_train)

Test)

ba(X_Test)[:,1]

Accuracy of Decision Tree-Train : 1.0

Accuracy of Decision Tree-Test : 0.955556

Date

18-4-2024

<=0.8

.667

150

0, 50, 50]

False

petal width(cm) <=1.75

gini = 0.5

samples = 100

value = [0, 50, 50]

LAB-5 25/04/2024

```
import pandas as pd
from matplotlib import pyplot as plt
from sklearn.model_selection import
    train_test_split
from sklearn.linear_model import LogisticRegre

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 = LogisticRegression()
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

LAB 6 09/05/2024

KNN Classifier Model

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

df = pd.read_csv('/cont')
df.head()

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)

neighbours = np.arange(1,9)
train_accuracy = np.empty(len(neighbours))
test_accuracy = np.empty(len(neighbours))

for i,k in enumerate(neighbours):
    knn = KNeighboursClassifier(n_neighbours = 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.scatter(neighbours, train_accuracy, color=
plt.show()
```

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

Output

0.730519

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:", metrics.accuracy_score(y_test,
                                    y_pred))
```

Output:

Accuracy : 0.9649

LAB 7 23/05/2024

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_layer = np.zeros(2)
hidden_weights = np.random.random((72,2))

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 i 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_rat
        x):
    output_derivative = np.zeros(2)
    output_gradient = np.zeros(2)
    for i in range(0, len(output_layer)):
        output_derivative[i] = (1.0 - output
            layer[i]) * output_layer[i]
    for i in range(0, len(output_layer)):
        output_gradient[i] = output_derivati
            [i] * (one_hot_encoding[i] -
                output_layer[i])
    hidden_derivative = np.zeros(72)
    hidden_gradient = np.zeros(72)
    for i in range(0, len(hidden_layer)):
        hidden_derivative[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_
                gradint)):
            sum_ += output_gradient[j] *
                hidden_weights[i][j]
        hidden_gradient[i] = sum_ *
            hidden_derivative[i]
recalculate(learning_rate, weight, hidden_
                gradient, x)
```

Random Boosting and Ada Boosting

```
dataset = pd.read_csv("...")

corr_matrix = input_x.corr().abs()
upper = corr_matrix.where(np.triu(
    np.ones(corr_matrix.shape),
    k=1).astype(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 =
n_estimators = 100
RANDOM_SEED = 13
names = ["Random Forest Classifier",
    "AdaBoost 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)
```

Acuracy RandomForestClassifier: 0.768439

Acuracy AdaBoostClassifier : 0.7341337

23-5-24

Random Boosting and Ada Boosting

```
dataset = pd.read_csv("...")

corr_matrix = input_x.corr().abs()
upper = corr_matrix.where(np.triu(
    np.ones(corr_matrix.shape),
    k=1).astype(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
RANDOME_SEED = 13
names = ["Random Forest Classifier",
         "AdaBoost 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)
```

Program 11:

PCA

```
df = pd.read_csv("... data.csv")
df_features = df.drop(['diagnosis'], axis=1)
from sklearn.preprocessing import StandardScaler
standardized = StandarScaler()
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 Components correlation with the features')
```