INDEX

Name: Saikumar Police Patil Std: 8C Sec:

Roll No: (8) Subject: ML - Lab School/College:

School/College Tel. No: Parents Tel. No:

Sl. No.	Date	Title	Page No.	Teacher's Sign. Remarks
1	21-3-24	import & export using pandas	1-2	
2	4-4-24	End-End ML project	3-5	
3	11-4-24	visualizing graphical data	6-7	
4	18-4-24	Simple linear regression	8-9	
5	25-4-24	Decision TREE algorithm	10-11	
6	2-5-24	KNN classifier	12-13	
7	9-5-24	logistic regression Model	14-14	
8	23-5-24	(a) SVM	15-16	
	23-5-24	(b) Dimensionality reduction	17	
	23-5-24	(c) K-means clustering	18-19.	
9.	30-5-24	(a) Random forest ensemble method	20-21	
	30-5-24	(b) Boosting algorithm.	22-23	

S.T.

Lab -1

=> Write a Python Program to Import and export data using Pandas, library functions.

```
import pandas as pd
airbnb data = pd.read_csv("datalistings_austin.csv")
airbnb_data.head()
```

Output:

id	name	host_id	host_name
2265	Ken-retreat	2466	Paddy
5245	Cocolorful	2466	Paddy
3265	neighborhood george	7892	Private room
1234	latitude	3234	Executive art
5982	Longitude	7892	Private room

Reading data from URL

```
Url = "https://archive.ics.uci.edu/ml/machine learning-
      database/iris/iris.data"

Url_names = ["sepal.length-in-cm", "stepal-width-in-cm", "sepal-length-in-cm", "class"]

iris_data = url.read_csv(url, names=col_names)

iris_data.read()
```

Output :

	Sepal-length-in-cm	Sepal-width-in-cm
0	5.1	3.5
1	4.9	3

Sepal-length-in-cm	Sepal-width-in-cm	Class
4.4	0.2	Iris-setosa
1.4	0.2	Iris-setosa

Lab-2

End-End-ML-Project

1. Look at the big picture.

Our first task is to use the California census data to build a model of the housing prices in the state. The data includes features such as.

- Population
- Median Income
- Media housing price for each ~~block~~ block group in California.

A block is the smallest geographical unit for which cen -sus data is Published.

Frame the problem

The problem is a Supervised learning problem, and because we're predicting the median housing price for a distri -ct, then this is a Regression task.

Performance measure : A typical measure for regression problem is Root Mean Squared Error.

$$RMSE(X,h) = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(h(x^{(i)}) - y^{(i)})^2}$$

Assumption Checking

Lastly, it is good practise to list & verify the ass of assumptions made by us or others, this can help us catch serious mistakes early one.

(2). Get the data.

-> Download the data : It is preferable to create some util functions to automate the process of downloading / extracting web-based data sets. So, we need to import some of the python libraries such as os, tarfile, urllib.

Then we download the data using fetch_housing_data. Pandas library is used to import the data from url's abstracted.

-> Take a Quick look at the data structure.

command: housing.head()

Each row represents one district & has the 10 attributes.

-> The info() method is useful to take a quick look at the data. It gives the info such as: The number of rows, Nan's per column, data type of each columns in the dataset.

The matplotlib.pyplot and seaborn libraries are imported in order to plot the graphs to gain the most insightful view into the data.

-> The Histograms shows the distribution of a particular column.

=> Create a Test set : creating a test set is easy. randomly select a subset of the data, typically 20% (or less if the dataset is large)

housing['ocean_proximity'].value_counts()

housing.groupby(by=['longitude', 'latitude']).count()

from sklearn.model_selection import train_test_split

train_set, test_set = train_test_split(housing, test_size=0.2, random_state=42)

Sept
14/4/24

Lab-2

28-3-24

(3) Discover & visualizing the data to gain insight

visualizing the graphical Data.

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

housing.plot(kind = 'scatter', x = 'longitude', y = 'latitude',
alpha = 0.4, s = housing['population']/100, label =
population, figsize = (10,4), c = 'median_house_value',
cmap = plt.get_cmap(name = 'jet'), colorbar = True)
plt.legend()

Looking For Correlations.

corr_matrix = housing.corr()

Prepare the data for Machine learning algorithm

housing = start_train_set.drop("median_house_value")
housing_labels = start_train.set("median_house_value").
copy()

* Data Cleaning

housing.dropna(subset = 'total_bedrooms')
computer = simpleImputer(strategy = 'median')
housing_num = housing.drop("ocean_proximity")
inputs.fit(housing_num)

- Handling Text and categorical Attributes

housing_cat = housing[['ocean_proximity']]
housing_cat.head(10)

* Select & train a Model

from sklearn.linear_model import LinearRegression

lin_reg = LinearRegression()
tree_reg = DecisionTreeRegressor
forest_regr = RandomForestRegressor

* Fine-Tune Your Model

forest_reg = RandomForestRegressor()
grid_search = GridSearch(CV(estimator = forest_reg,))
param_grid = param_grid, scoring = 'neg_mean_square_error', CV=5, return_train_score = True, n_jobs=1).

* Launch Monitor & Maintain your System

We can hot-load the model with in web application ~~or~~ alternatively, wrap the model around its own API end-point & design web component separately.

Week-4 (LAB) 18-4-23

Simple Linear Regression

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

Import data

```
df_sal = pd.read_csv('/content/Salary_Data.csv')
df.head()
```

Analyze data

```
df_sal.describe()  # mean, count, std, min, max
```

Distribution :

```
plt.title("Salary Distribution plot")
sns.distplot(df_sal['Salary'])
plt.show()
```

Scatter Plot [Salary Vs Expirience]

```
plt.scatter(df_sal['Years Experience'], df_sal['Salary'],
            color = 'mediumseagreen')
plt.title('Salary Vs experience')
plt.xlabel('Years of experience')
plt.ylabel("Salary")
plt.box(False)
```

split Data

```
x = df_sal.iloc[:, :1] # independent
y = df_sal.iloc[:, 1:] # dependent

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_train = regressor.predict(x_train)
```

Prediction on training set

```
plt.scatter(x_train, y_train)
plt.plot(x_train, y_pred_train)
plt.title('Salary vs experience')
plt.xlabel('Years of experience')
plt.ylabel('Salary')

plt.show()
```

Prediction on test set

```
plt.scatter(x_test, y_test)
plt.plot(x_train, y_pred_train)
plt.title('Salary vs Experience')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
```

Lab-5

Decision TDT Algorithm Implementation.

⇒ Reading the dataset

```
import numpy as np
import pandas as pd
from numpy import log2 as log
df = pd.read_csv("dataset.csv")
```

Calculating entropy

$$Entropy = \frac{-p}{p+n}\log_2\left(\frac{p}{p+n}\right) - \frac{n}{(p+n)}\log_2\left(\frac{n}{p+n}\right)$$

```
def find_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])
        entropy += -fraction * np.log2(fraction)
    return entropy
```

Average information gain

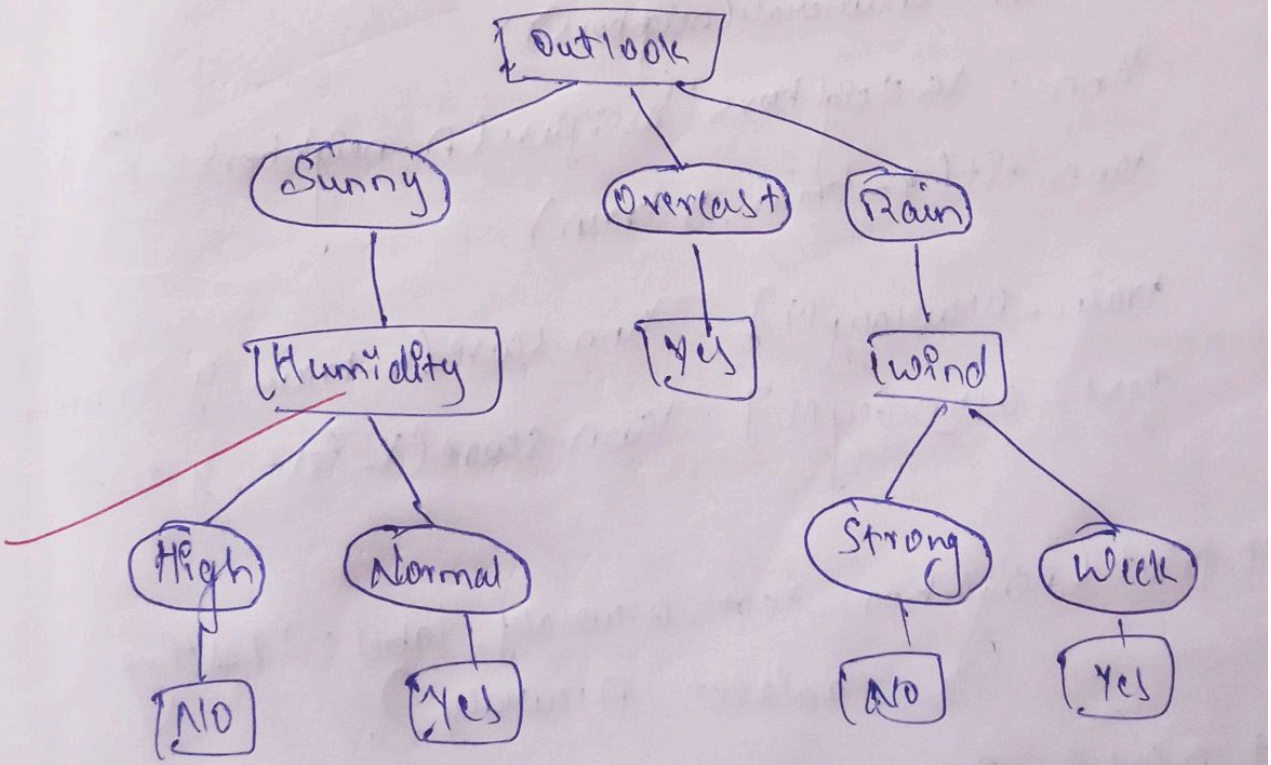
```
def average_information(df, attribute):
    target = df.keys()[-1]
    target_variables = df[target].unique()
    variables = df[attribute].unique()
    entropy2 = 0
for variable in variables
        entropy = 0
for target_variable in target_variables:
```

```
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)
```

Information gain

```
def find_min(df):
    Ig = []
for key in df.keys()[:-1]:
    Ig.append(find_entropy(df) - average_information
                                (df, key))
return df.keys()[:-1][np.argmax(Ig)]
```

Decision Tree



Lab-6

9-5-2024

=> KNN classifier.

```
From sklearn.neighbors import kNeighborsClassifier
From sklearn.model_selection import train_test_split
From sklearn.datasets import load_iris
import numpy as np
import matplotlib.pyplot as plt

irisData = load_iris()

X = irisData.data
y = irisData.target

X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, random_state=42)

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.plot(neighbors, test_accuracy, label='Testing
                dataset Accuracy')

plt.plot(neighbors, train_accuracy, label='Training
```

plt.legend()
plt.xlabel('n_neighbors')
plt.ylabel('Accuracy')
plt.show()

Output

	Sepal-length	Sepal-width	petal-len	petal-wid	Species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	4.6	3.6	1.4	0.2	Iris-setosa

9-5-24

⑤. Build Logistic Regression Model

```
From sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
From sklearn.metrics import accuracy_score
x,y = load_breast_cancer(return_X_y = True)
x_train, x_test, y_train, y_test = train_test_split
    (x, y, test_size = 0.20, random_state = 23)
clf = LogisticRegression(random_state = 0)
clf.fit(x_train, y_train)
y_pred = clf.predict(x_test)
acc = accuracy_score(y_test, y_pred)
print("Logistic Regression model accuracy (in %):"
                acc * 100)
```

Output:

Logistic Regression model accuracy : 95.61

Sri 7/5/24.

Q. Build Support vector machine model for a given dataset

```
=> From sklearn.datasets import load_iris
   From sklearn.model_selection import train_test_split
   From sklearn.preprocessing import StandardScaler
   From sklearn.svm import SVC
   From sklearn.metrics import accuracy_score,
                               classification_report

   iris = load_iris()
   x, y = iris.data, iris.target

x_train, x_test, y_train, y_test = train_test_split(x, y,
                    test_size = 0.2, random_state = 42)

scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)

svm_model = SVC(kernel = 'rbf', C=1.0, gamma = 'scale',
                    random_state = 42)

svm_model.fit(x_train_scaled, y_train)

=>  SVC
    SVC(random_state = 42).

y_pred = svm_model.predict(x_test_scaled).

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

print("\n Classification Report:")
print(classification_report(y_test, y_pred, target_names=iris.target_names))

Classification Report:

	Precision	recall	f1-score	support
Setosa	1.00	1.00	1.00	10
versicolor	1.00	1.00	1.00	9
virginica	1.00	1.00	1.00	11
accuracy			1.00	30
macro avg	1.00	1.00	1.00	30
weighted avg	1.00	1.00	1.00	30

3. Implement dimensionality reduction using principal component Analysis (PCA) method

=> Import the libraries
=> Load the Iris dataset
=> split the dataset into training & testing sets.
=> Preprocess the data.
=> Apply PCA for dimensionality reduction.

```
pca = PCA(n_components) # Reduce to 2 principal components
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)

svm_model = SVC(kernel='rbf', C=1.0, gamma='scale', random_state=42)

y_pred = svm_model.predict(X_test_pca)

accuracy = accuracy_score(y_test, y_pred)
print("Accuracy with pca:", accuracy)
print("Classification Report with PCA:")
print(classification_report(y_test, y_pred, target_names=iris.target_names))
```

=> Accuracy with PCA: 0.9

Classification Report with PCA:

	Precision	recall	f1-score	support
accuracy			0.90	30
macro avg	0.90	0.90	0.90	30
weighted avg	0.90	0.90	0.90	30

(5). Build k-means algorithm to cluster a set of data stored in a .csv file.

```
=> import numpy as np
class KMeans:
    def __init__(self, n_clusters, max_iter=300):
        self.n_clusters = n_clusters
        self.max_iter = max_iter

    def fit(self, X):
        centroids_indices = np.random.choice(len(X),
        size=self.n_clusters, replace=False)
        self.centroids = X[centroids_indices]

        for _ in range(self.max_iter):
            labels = self.assign_clusters(X)
            new_centroids = np.array([X[labels == k].
            mean(axis=0) for k in range(self.n_clu
            if np.all(self.centroids == new_centroids
            break

            self.centroids = new_centroids.

np.random.seed(42)
X = np.random.rand(100,2)
kmeans = KMeans(n_clusters=3)
kmeans.fit(X).
```

```
labels = kmeans.fit_predict(X)
Print("Cluster labels:", labels)
```

Output:

Cluster labels: [1 0 2 1 0 1 0 1 0 2 1 2 2 2 1 1 1 2
0 2 2 0 2 2 0 0 1 0 0 2 2 0 0 1
2 0 1 0 0 1 1 0 1 2 1 2 2 2].

Sri
23/5/24

Lab-8

(8)-(a) Implement Random Forest ensemble method.

```
=> import numpy as np
From sklearn.datasets import load_iris
From sklearn.model_selection import train_test_split
From sklearn.ensemble import RandomForestClassifier
From sklearn.metrics import accuracy_score, classification_
    report.

#load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

#split the dataset into training & testing sets.
X_train, X_test, y_train, y_test = train_test_split
    (X, y, test_size = 0.3, random_state = 42)

#Initialize Random Forest classifier
rf_classifier = RandomForestClassifier(n_estimators = 100,
                random_state = 42).

#Train the classifier.
rf_classifier.fit(X_train, y_train).

y_pred = rf_classifier.predict(X_test).

accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred).

print('Accuracy: {accuracy}').
print('Classification Report:')
print(report).
```

Output:

Accuracy : 1.0

Classification Report:

	Precision	recall	f1-score	support
0	1.00	1.00	1.00	14
1	1.00	1.00	1.00	13
2	1.00	1.00	1.00	13
accuracy			1.00	45
macro avg	1.00	1.00	1.00	45
weighted avg	1.00	1.00	1.00	45.

8. Implement Boosting ensemble method.

```
From sklearn.datasets import load_breast_cancer.
x,y = load_breast_cancer(return_x_y = True).

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 = 23)

From sklearn.tree import DecisionTreeClassifier
From sklearn.metrics import accuracy_score

dtree = DecisionTreeClassifier (max_depth =3, random_state =23)
dtree.fit(x_train, y_train)
dt_pred = dtree.predict( x_test)
dt_acc = round ( accuracy_score(y_test, dt_pred),3)
Print ("Decision Tree classifier Accuracy score:", dt_
    acc).
```

=) Decision Tree classifier Accuracy : 0.953

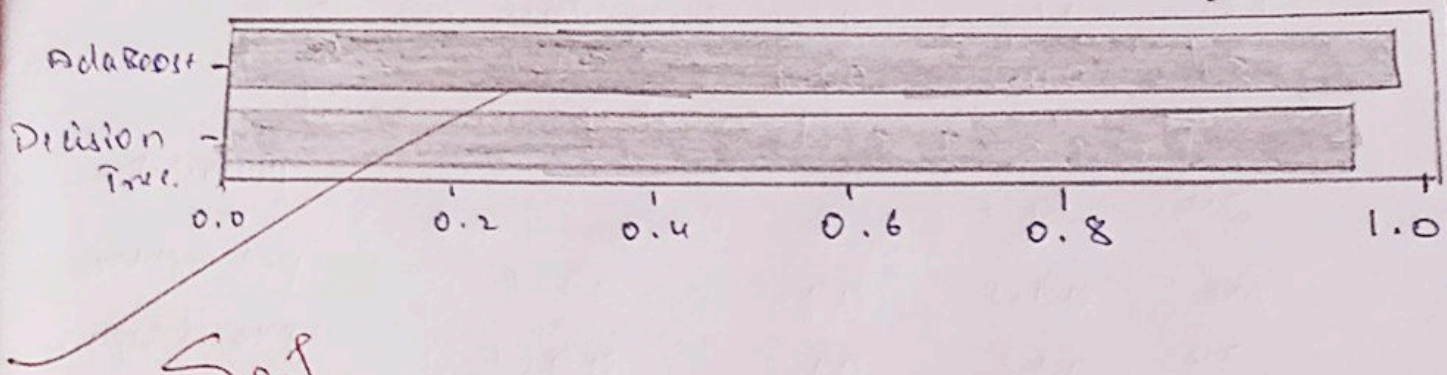
```
From sklearn.ensemble import AdaBoostClassifier

ada = AdaBoostClassifier(n_estimators =50, learning_rate =0.6)
ada.fit(x_train, y_train)
ada_pred = ada.predict (x_test).

ada_acc = round (accuracy_score( y_test, ada_pred),3)
Print("Decision tree AdaBoost Model Accuracy
    Score:", ada_acc)
```

=) Decision Tree AdaBoost Model Accuracy score : 0.982.

```
import numpy as np
import matplotlib.pyplot as plt
plt.figure(figsize=(10,2))
plt.barh(np.arange(2), [dt_acc, ada_acc],
        tick_label = ['Decision Tree', 'AdaBoost'])
```



Sol.1
30/7/24