INDEX

Name SANKETH BM Std 6th Sem Sec C

Roll No. Subject ML Lab School/College

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

Sl. No.	Date	Title	Page No.	Teacher Sign / Remarks
1	21/03/24	LAB 1		
2	4/4/24	lab(2 & 3) end-end project		
3	18/4/24	Lab - 4 (Regressions)		
4.	25/4/24	Lab-5 (Decision_tree) - ID3		
5.	9/5/24	LAB-6 (Logistic Regression, KNN- Classification)		
6.	23/5/24	SVM, K-clusters, PCA		
7	30/5/24	Random forest Model <Ensemble learning (Boosting)		

25/4/2024

30/5/2024

LAB-1

Q) Write a python program to import and export data using pandas library functions.

→ import pandas as pd.
airbub data = pd.read_csv("datalistings_austin.csv")
airbub_data.head()

output

id	name	host_id	host_name
2265	Ken_retreat	2466	Poddy
5245	Ecocolonful	2466	Poddy
3265	Neighbourd	7892	Private room
1234	lattitude	3234	Executive art
5298	Longtitude	7892	Private room

Reading data from URL

url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"

url_names = ["sepal_length_in_cm", "sepal_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	6.9	3

	Sepal_length_in_cm	sepal_width_in_cm	class
	1.4	0.2	iris_setosa
	1.4	0.2	iris_setosa

LAB-2

→ End-to-End project:

① Look at the big picture:

Dataset consists of california census data to build a model for housing prices in the state.

→ features: * population
* Median Income
* Median housing price for each block group in California.

Block group is smallest geographical area.

→ The problem is a supervised learning problem, and because we're predicting median housing price for a district, then we can use regression task.

Performance measure: (Root mean square method).

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

② Get the data:

→ importing the data using pandas

→ .head() [loads the dataset into the file representing all the columns & rows]

→ Attribute

(10 columns, 20,640 rows)

- Longitude
- Latitude
- housing_median_age
- total_rooms
- total_bedrooms
- population
- households
- median_income
- median_house_value
- ocean_proximity.

→ describe() function shows all numerical values of dataset for each attributes
Eg: count, mean, std, min, max, IQR
(ignores null values)

→ plotting histograms for each attributes which shows the frequency of distribution over a continuous range of values.

```
housing.hist(bins=50, figsize=(20,15))
plt.show()
```

→ dataset is devided to 80-20 for training & testing data: [testing data: 20%]

→ housing['id'].value_counts()
→ housing.group-by(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)
```

4/9/2024

(3) Discover & visualize the data to gain insights

Visualizing the geographical data

```
housing.plot(kind='scatter', x='longitude', y='latitude')
plt.show()
housing.plot(kind='scatter', x='longitude', y='latitude')
plt.show()
alpha=0.4, s=housing['population']/100, label='
population', figsize=(10,7), c='median_house_
value,
cmap=plt.get_cmap(name='jet'), colorbar
=True)
plt.legend()
```

④ Looking for correlations!

corr_matrix = housing.corr()

Prepare the data for Machine learning Algorithms

housing = strat_train_set.drop("median_house_value")
housing_labels = strat_train_set["median_house_value"].copy()

⑤ Data cleaning:-

housing.dropna(subset=['total_bedrooms'])
imputer = SimpleImputer(strategy='median')
housing_num = housing.drop("ocean_proximity")
imputer.fit(housing_num).

⑥ Handling text & categorical Attributes.

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

* Select & train model.

from sklearn.linear_model import LinearRegression.

lin_reg = LinearRegression()
tree_reg = DecisionTreeRegressor
forest_rmse = RandomForestRegression()

Find - Tune your model.

* forest_reg = RandomForestRegression().

grid_search = GridSearch(estimator = forest_reg,
param_grid = param_grid, scoring = neg_mean_
square_

- Launch, monitor & Maintain your System.

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

WEEK-4 (CAA 4) 18/4/2024

Q 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, IQR.
```

Distribution:

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

Scatter plot [Salary vs Experience]

```
plt.scatter(df_sal['Years Experience'], df_sal['Salary'],
            color = 'lightcoral')
plt.title('Salary vs Experience')
plt.xlabel('Years of Experience')
plt.ylabel("Salary")
plt.box(False)
plt.show()
```

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 test(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')
plt.show()
```

Lab-4

Simple Linear Regression

```
import numpy as np
import matplotlib.pyplot as plt
def estimate_coeff(x,y)
    n = np.size(x)
    #mean of x,y vector
    # calculating cross-deviation& deviation about x
    #calculating regression coefficient

def plot_regression_line(x,y,b):
    plt.scatter(x,y,color='m',marker='o',s=20)
# plotting the reg line
    plt.show()

def main():
# observation / data
# estimating coefficient
plot reg line

if __name__=="__main__":
    main()
```

Multiple linear Regression

```
import sklearn model_selection import
    train_test_split
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd.
from sklearn import dataset, linear_model
    metrics
```

* Import data from url
* reg = linear_model.LinearRegression()
* calculate coefficient
* calculate variance
* plot the Residual error

LAB - 5

Decision tree ID3 algorithm Implementation

Steps

* Reading the data sets
* Importing packages numpy, pandas
* Reading the data set and set the indexes.

Decision tree

ID3 is simple decision tree algorithm.

Implement used functions

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

Average Information

$I = \sum \frac{p_i+n_i}{p+n}$ Entropy (attribute)

Information gain

Gain = Entropy (S) - I (A#).

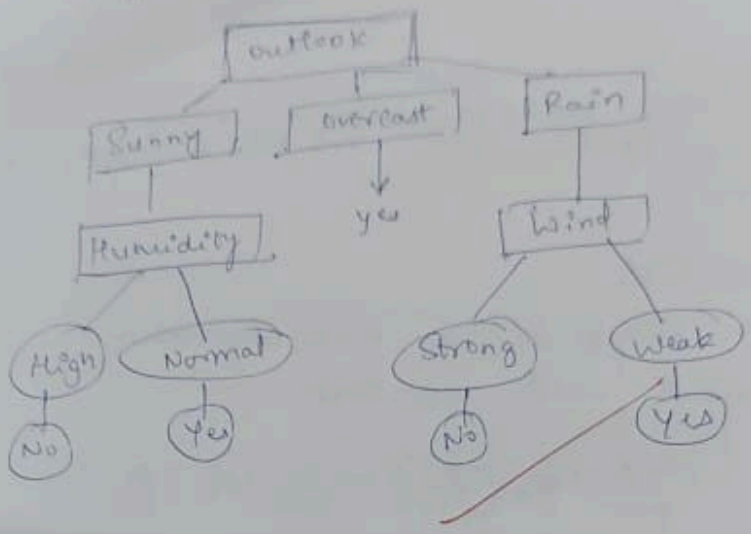
Build Decision tree

Build Tree function construct a decision tree recursively.

Select the attribute with high information gain

process continue till subsets are pure.

Play tha tree



LAB-6 (KNN)

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
// loading datasets

import numpy as np
import matplotlib.pyplot as plt

irisData = load_iris()

# create feature and target arrays.
X = irisData.data
y = irisData.target

# split into training and test set
X_train, X_test, y_train, y_test = train_test_split(
                X, y, test_size=0.2, random_state=42)

neighbors = np.arrage(1,9)
test_acc = np.empty(len(neighbors))
test_ace = np.empty(len(neighbors))

# loop over k values
for i, k in enumerate(neighbors):
        knn = KNeighborsClassifier(n_neighbors=k)
        knn.fit(X_train, y_train)

# compute training and test data accuracy
train_accuracy[i] = knn.score(X_train, y_tra
test_accuracy[i] = knn.score(X_test, y_test)
```

```
# generate plot
plt.plot(neighbors, test_acc)
plt.plot(neighbors, train_acc)

plt.legend()
plt.xlabel("n_neighbors")
plt.ylabel("Accuracy")
plt.show()
```

o/p	Sepal length	Sepal width	Petal length	width	species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	"
2	4.7	3.2	1.3	0.2	"
3	4.6	3.1	1.5	0.2	"
4	5.0	3.6	1.4	0.2	"

LAB-7

→ SVM (Support vector Machine Model)

(Linear)

→ Importing necessary libraries.

```
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)
```

Accuracy : 1.0.

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

Classification Report:

	Precision	recall	ff-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
			1.00	30
accuracy			1.00	30
macro avg	1.00	1.00	1.00	30.
weighted avg	1.00	1.00		

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=2)  # Reduce to 2 principal components
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.fit_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(y_test, y_pred, target_names
                            =iris.target_names))
```

Accuracy with PCA = 0.9

Classification Report with PCA:

	Precision	recall	f1-score	Support
uracy			0.90	30
ro avg	0.90	0.90	0.90	30
ht d avg	0.90	0.90	0.90	30

Q) 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_clusters)])

            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.assign_clusters(X)
print("Cluster labels:", labels)
```

LAB-08

1) Random Forest classifier

```
# Importing necessary libraries
# Load the Iris dataset
iris = load_iris()
x = iris.data
y = iris.target

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

rf_model = RandomForestClassifier(n_estimators=100,
                                  random_state = 42)

rf_model.fit(X_train, y_train)
y_pred = rf_model.predict(X_test)

# evaluate the model

accuracy = accuracy_score(y_test, y_pred)
print("Accuracy of Random forest model:",
      accuracy)
```

2. Build Logistic Regression Model

```
from sklearn.datasets import load_diabetes
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

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)

Accuracy = 95.61

5/2024

———X———

Boosting Implementation

```
#import necessary libraries
# loading 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 = 0.2, random_state = 42)

base_estimator = DecisionTreeClassifier(max_depth=1, random_state
                                        = 42)

ada_model = AdaBoostClassifier(base_estimator = base_
                               estimator,
                    n_estimators = 50, random_state = 42)

# train the model
ada_model.fit(X_train, y_train)

y_pred = ada_model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print("Accuracy of AdaBoost model:", accuracy)
```

30/5/2024