Dataset-1

Lab-0

Global Blood Group Distribution worldwide Dataset.

attributes:

1) Country : Where the person having blood group is from.
2) Population : Number of people
3) Blood Groups:
   [O+, A+, B+, AB+, O-, A-, B-, AB-]

- Missing values in blood groups:
  O-, A-, B-, AB-
- Total unique values - 126

Dataset-2

Metro Transit Data

This dataset was created for analyzing metro data across indian cities

no. of rows : 226
no. of columns = 10
city → name of the city
country → Name of the country
Name → Name of the metro system
Service opened → opening date of metro :
Last-expanded - : Year when the metro system was last expanded.
Lines → Total number of metro lines
System Length → length of the metro length in K.M.

Dataset-3

Startup growth & Funding Trends.

No. of columns : 12

No. of rows : 500.

Attributes :

StartupName : The name of the startup

Industry - The sector in which the start-up operates.

Funding Rounds - The total number of funding rounds

valuation - The startup's post-money

Revenue - The estimated revenue

Employee - The number of employees working with start-ups.

Market share - The percentage of market share that the start-up has captured.

profitable - A binary indicator (1 = profitable, 0 = Not profitable)

Dataset - 4

Titanic Dataset - EDA & Logistic regressions.

No. of rows : 891

No. of columns : 10 to 12 columns.

1. Survival prediction: To build a logical regression
2. Data cleaning and preprocessing: To perform data cleaning
3. EDA: visualizing patterns.

Name - person's name

Sex → either → Female
↳ Male

Aage : age of a person.

Ticket → which type of ticket.

Fare → Price of the fair

Dataset-5

Mars rover datasets

id → unique identifier for each image

sol → Martian sol (day) when the image was captured.

camera_name → rover's camera

img_scr → url link to the image

earth_date → The earth date corresponding to the martian sol.

rover_status : curren operational status → active
↳ complete

landing_date → Date when it was landed.

launch_date → Date when launched from earth.

No. of rows: 856 values

No. of columns: 10 columns.

Lab-1

Linear Regression in Machine Learning

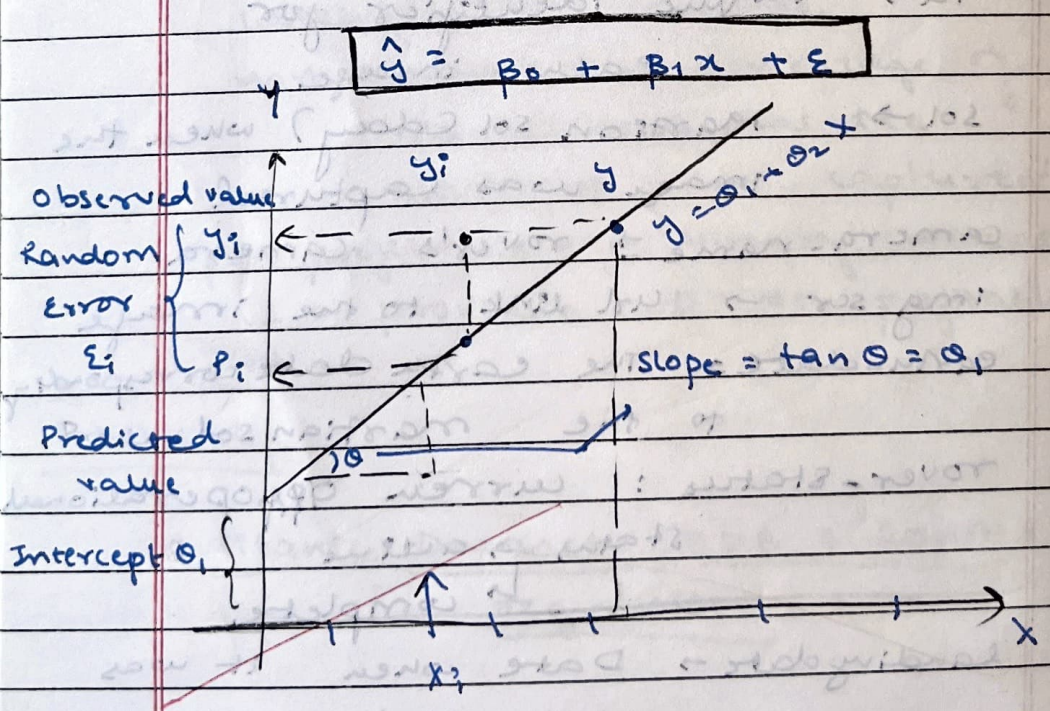
Linear Regression is a statistical method used to model the relationship between a dependent variable and one independent variable.

- It is a model-based learning and also a type of supervised machine learning

Regression: try to find the relationship between variables.

Equation for Linear expression:

$$\hat{y} = \beta_0 + \beta_1 x + \varepsilon$$



Slope = $\frac{\text{change in } x}{\text{change in } y} = \frac{\Delta x}{\Delta y}$

- Linear regression learns from labelled datasets.

code for linear regression:

```
import pandas as pd
import numpy as np
// load dataset
df = pd.read_csv("data_for_lr.csv")
df = df.iloc[:50] // first 50 points
// first row & second column
x = df.iloc[:, 0].values
y = df.iloc[:, 1].values
// least square formula
x_mean, y_mean = np.mean(x),
                 np.mean(y)
// slope - m & intercept is (b)
m = np.sum(x - x_mean) * (y - y_mean) /
    np.sum(x - x_mean) ** 2

b = y_mean - m * x_mean

// Predict values
y_pred = m * x + b

mse = np.mean(y - y_pred) ** 2

print(f"Slope (m): {m},
      Intercept (b): {b},
      MSE: {mse}")

// Plotting
plt.scatter(x, y, color='blue',
            label='Datapoints')
```

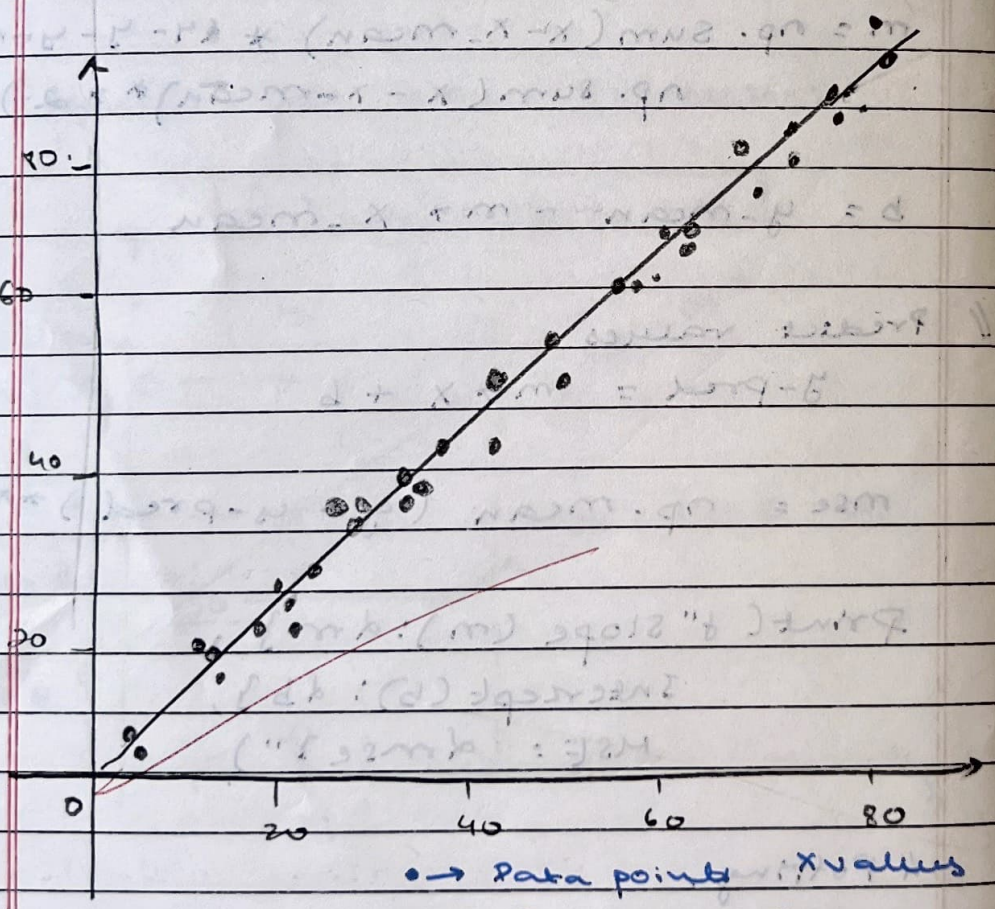
```
plt.plot(X, y_pred, color='red',
         label='Regression Line')

plt.xlabel("X values")
plt.ylabel("Y label")
plt.legend()
plt.title("Linear Regression")
plt.show()
```

output:

Slope (m): 0.9926528446814438

Intercept (b): -0.07380229436

MSE: 6.421138985746 97



formula:

$$B_1 = \frac{\overline{XY} - \overline{X} \cdot \overline{Y}}{\overline{X^2} - (\overline{X})^2}$$

$$B_0 = \overline{Y} - B_1 \overline{X}$$

10/3/25

Bafna Gold
Date: Page:

Lab - 2

End-to-End Machine Learning Project.

Step - 1.

Download the dataset in CSV format
ie. housing.tgz → housing.csv

attributes: 1) longitude
2) latitude
indexes: 20640 3) housing_median_age
4) total_rooms
5) total_bedrooms
6) population
7) households
8) median_income
9) median_house_value
10) ocean_proximity.

numerical data ↓ can use regression

string/text value

can only be categorized. (object-type)

Step-2

Load the dataset, for quick glimse:

housing.head() → displays first 5 rows with attributes.

housing.info() → Gives info about whole dataset

Step-3

categorize based on ocean-proximity

housing.describe() → gives mean, count, std, max, min, 25%, 75%, 50% of all attributes. [null values are ignored]

or

use histogram for each attributes.
bins => 50 } divides the data into 50-equal width.

Step-4

create a Test set

Data Snooping: It occurs when model is trained or tested on data that was used in preprocessing / decision making process.

Test_data: test size
Training data = Totaldataset - Test Data [16512]

iloc[] → specific rows.

np. random. permutation()
↳ using unique ID's & immutable identifer. or use hash function.

random_state & using Split_trainTest()

Visualization Methods:

1) Histogram:
- hist() → visualize the data based on histograms

2) using scatter plots
using scatter plots between different types of attributes.

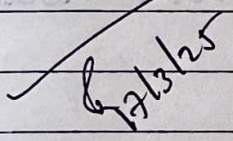
(alpha = 0.1) → density based scatter plotting for visualization

3) correlation:
using: corr() for attributes.

Step-5
Data cleaning:
1) remove corresponding districts
2) Get rid of whole attribute
3) set values to some value

use dropna(), fillna(), drop()

Step-6
Select and train a model.
use regression models.
for validation use :: train_test_split()



24/03/25

Lab-04

Decision Tree ID3 classification

Psuedocode.

```
Function ID3 (Data, features, Target):
    If all instances in Data have the
        same class:
        Return the class label

    If features is empty:
        Return the most common class
            in Data

    BestFeature ← feature with highest
    Information Gain in Features
    Tree ← create a root node labeled
        BestFeature

    For each value v in BestFeature:
        subset ← subset of Data where
            BestFeature = v
        If subset is empty:
            Add a leaf node with the
            most common class in Data

        Else:
            childNode ← ID3(subset,
            Features - {BestFeature}, Target)
            Add childNode to Tree

    Return Tree
```

Entropy for each feature:

Outlook : 1.5774

Temperature : 1.5567

Humidity : 1.000

Windy : 0.9852

Information Gain for each feature:

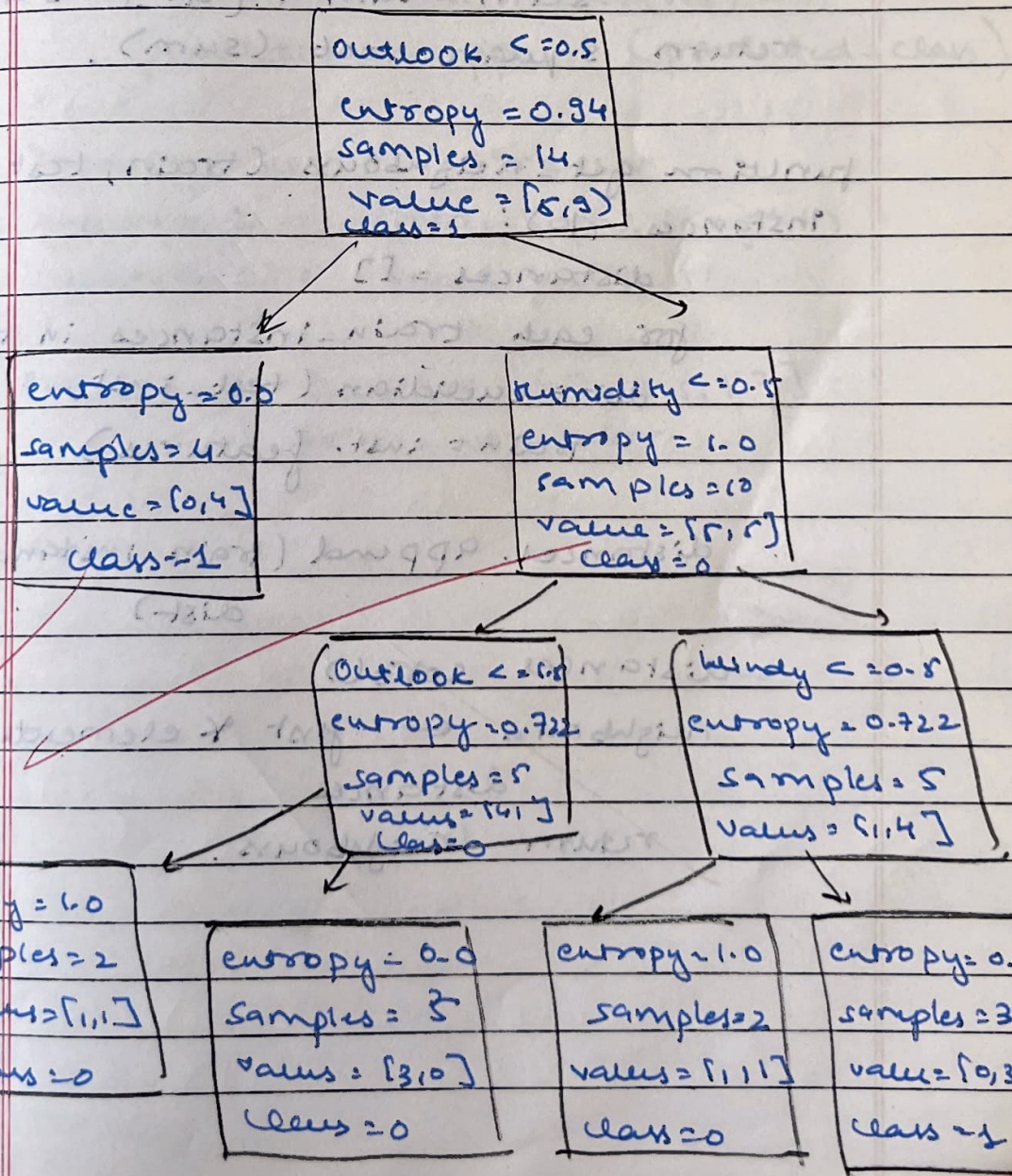
Outlook : 0.2467

Temperature : 0.0292

Humidity : 0.1518

Windy : 0.0481

Root Node (Best split) : Outlook.



Lab-5

KNN Algorithm.

```
Function split (data, 0.2):
    split_index ← length(data) * (1 - 0.2)
    train = first split_index elements of data
    test = remaining elements of data
    return train, test.

function euclidian (a, b):
        sum = 0
        for i = 1 to length(a):
            sum = sum + [a[i] - b[i]]^2
    return square-root (sum).

function get_neighbours (train, test_
    instances, k):
        distances = []
        for each train_instances in train
            dist = euclidian (test_inst,
                train = inst. features)
            distances. append (train_instances,
                dist)
        distances. sort()
        neighbours ← first k elements of
                distances
        return neighbours.
```

function predict (train, test-instance, k):
    neighbours ← get-neighbours
    (train, test-instance, 3)
    count occurances of each class in
    neighbours.
    return class with max (count).

function KNN()
    train-test = split. dataset (dataset=0.7)
    k = 3
    predictions = []
    for each test_instance in test_set:
        predicted-class = product (train,
        test-instance. features, k)
        predictions: append (predicted-class)

Dataset: use 'iris.csv'
feature 1 = spread. sepal. length()
feature 2 = Sepal width

Predicted class for [6.7, 3.5, 1.7]:
    Iris-ver. color.

SVM Algorithm:

```
function load_dataset (filename):
    dataset = []
    open (file)
    skip header
    for each row in file:
        features = first 2 feature values
        label = 1 if class is 'Iris-setosa',
        else 0
    dataset.append ([features, label])
    return dataset

function train_test_split (dataset, test_
                ratio):
    split_under = length (dataset) x
                (1 - test_ratio)
    train_set = dataset [:split_index]
    test_set = dataset [split_under]
    return trainset, test_set.

function dot_product (a, b):
    return sum (a[i] x b[i]) for
                all

function train_svm (train, learning_
    rate, lamda_param, epochs)
        wt [0,0]
        b ← 0
    for each epoch in epoch:
        for each data point (x, y) in
                transit:
```

```
if (y x (dot-product) (w, x) + b ≥ 1 :
    update w = w - learning-
        rate x 2x lamda-paramxw
else:
    update w = w - learning-rate (2x
        lamda_param - w - yxx)
    update b = b + learing_ratexy
    end if
return w, b.

function predict (x, w, b):
    return 1 if dot-product (w, x)+
        b>0 else -1

filename = 'content & iris.csv"
dataset = load-dataset (filename)
train, test = train-test-split (dataset)
w, b = train-sum (train)

plot-sum (train, w, b)

for each point in test:
    prediction = predict (point.features, w, b)
    print ( Actual | Predicted).
```

Lab-6

Random forest classification

```
RandomForestClassifier(X_train, y_train,
T, max_depth, min_samples_split):

  n_samples, n_features = X_train.shape

for t = 1 to T:
    bootstrap_idx = random.sample(
    range(n_samples), n_samples)
    X_bootstrap = X_train[bootstrap_idx]
    y_bootstrap = y_train[bootstrap_idx]

#Train decision Tree
  tree = DecisionTree(max_depth=max_depth,
  min_samples_split=min_samples_split)
  trees.append(tree)
return trees

Predict(X_test):
    prediction = []
    for tree in trees:
        tree_predictions = tree.predict(X_test)
        predictions.append(tree_predictions)

final_predictions = majority_vote
                    (predictions)
return final_predictions

majority_vote(predictions):
return mode(predictions)
```

Bafna Gold
Date: Page:

Lab-7

Boosting Ensamble

1. Initialize weights for each training sample:

2. For each iteration (t = 1 to T):

a. Train a weak classifier (e.g. a decision stump) using the weighted sampl

b. calculate the classifier
- alpha = 0.5 * log [(1 - error) / error]

if error is high, alpha will be low, meaning less importance for this classifier.

c. calculate the classifier's error:
error = (sum of weights of misclassified samples) / (sum of all sample weights)

d. update the sample weight
- increase the weight of misclassified sample
- Decrease the weight of correctly classified samples.
- Normalize the weight to make sure they can sum to 1.

3. After all iterations, combine the weak classifiers:
- each classifier contributes based on its importance (alpha).

- Final prediction =
sign(sum of all classifiers)

4. Use the combined classifier to make predictions on new data.

Lab-8

K-means clustering.

1. Initialize K clusters centroids [randomly selected data points].

2. Repeat until convergence:
   a. Assign step:
      - For each data point, calculate the distance to each centroid.
      - Assign each data points to the the closest centroid [cluster].
   b. Update step:
      - For each cluster, compute the mean of all data points assigned to that cluster.
      - Update the centroid to this mean.

3. Repeat the process will be until the centroids do not change significantly (convergence)

4. The final centroids represent the cluster centers, and each data point belong to one of the clusters.

Lab-9

Input:

Data matrix X with shape (n-samples, n-features)

Desired number of components k

step 1- center the data

For each feature column in X:

Subtract the mean of that column for each value

step 2: compute the covariance matrix

cov = (1 / (n-samples - 1)) * X_transposed x X

step 3: compute eigen values and eigenvectors of the covariance matrix [eigenvalues, eigenvectors] = eig(cov)

step 4: sort eigenvectors by descending eigenvalues

sort eigenvectors so that those with highest eigenvalues come first

step 5: select top k eigenvectors

select the first k eigenvectors to form the projection matrix W

step 6: Transform the original data

X_reduced = X * W

output: X_reduced (data in k dimension)