Date __/__/__
Page _____

1) Obesity Prediction Dataset:
Understanding lifestyle factors that contribute to obesity can help with early intervention, health recommendations, and machine learning applications in healthcare.

Attributes:

1) Gender — No. of columns: 14
2) Age — No. of rows: 2111
3) Weight
4) Weight
5) family history with overweight
6) FAVC - Frequent high-calorie food consumption
7) FCVC - Vegetable consumption
8) NCP - No. of meals/day
9) CAEC - Frequency of consuming food b/w meals
10) Smoke
11) H_2O - daily water intake
12) SCC - If person monitors their calorie intake
13) FAF - Physical activity
14) CALC - Alcohol consumption

2) Road Accident Survival Dataset
Analyzing the impact of Age, Speed, and safety measures on accident survival.

Attributes:

1) Age — No. of Rows: 76
2) Gender — No. of columns: 6
3) Speed of impact
4) Helmet Used
5) Seatbelt Used
6) Survived

Date / /
Page

~~3) Indian Pole Groups~~

3) Diabetes dataset

Includes various health parameters, lifestyle habits that contribute to diabetes risk.

Attributes :

1) Age
2) Pregnancies
3) BMI
4) Glucose
5) Blood Pressure
6) LDL
7) HDL
8) HbA1c
9) Triglycerides
10) Waist circumference
11) Hip circumference
12) WHR
13) Family History
14) Diet Type
15) Hypertension
16) Medication Use
17) Outcome

No. of rows : 9538
No. of columns : 17

4) Titanic Dataset

1) PassengerID
2) Survived
3) Pclass
4) Name
5) Sex
6) Age
7) SibSp - Siblings / spouse
8) Parch - Parents child
9) Ticket
10) Fare

No. of columns : 10
No. of rows : 2224

Date __/__/__
Page _____

5) Customer Personality Analysis

Helps business to better understand its customers and makes it easier for them to specify products according to the specific needs, behaviours.

Attributes

1) ID
2) Year_Birth
3) Education
4) Marital_Status
5) Income
6) Kidhome
7) Teenhome
8) Dt_Customer
9) Recency
10) Complain

No. of rows : 2240
No. of columns : 10

3/3/25

LAB-1

Date __/__/__
Page _____

LINEAR REGRESSION

Linear regression is a supervised machine learning algorithm that is used to model the relationship between two variables by fitting a straight line to the data. The goal is to find the line that best describes how one variable (the independent variable, or "x") influences another value (the dependent variable, or "y").

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

($\hat{y}$: output; $\beta_0 + \beta_1$: coefficients; x: input; ε: error)

$\hat{y}$ - is the dependent variable
x - is the independent variable
β_0 - is the intercept
β_1 - is the slope.

Linear Regression Code

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

np.random.seed(42)
x_values = np.random.uniform(0, 50, 50)
y_values = 3 * x_values + 5 + np.random.normal(0, 10, 50)
def LR_formula(x: list[int], y: list[int]):
    x_values = x_values.reshape(-1)
    N = len(x_values)
    sum_x = np.sum(x_values)
```

Date __/__/__
Page _____

```
sum_x = np.sum(x_values)
sum_y = np.sum(y_values)
sum_xy = np.sum(x_values * y_values)
sum_x_squared = np.sum(x_values ** 2)

xy_mean = sum_xy / len(x)
x_mean = sum_x / len(x)
y_mean = sum_y / len(x)
x2_mean = sum_x2 / len(x)

b1 = (xy_mean - x_mean * y_mean) / (x2_mean - x_mean ** 2)

b0 = y_mean - b1 * x_mean
return b0, b1

def predict(b0: int, b1: int, x: int):
    return b0 + b1 * x

x = x_coor = [x for x in range(1, 31)]
y = y_coor = [round(2 * x + 10 + random.uniform(-5, 5), 2)
              for x in x_coor]

print(x_coor)
print(y_coor)

b0, b1 = LR_formula(x_coor, y_coor)
y = predict(b0, b1, 32)
print(round(y))
plt.scatter(x_coor, y_coor, color='blue', label='Data
    points')
```

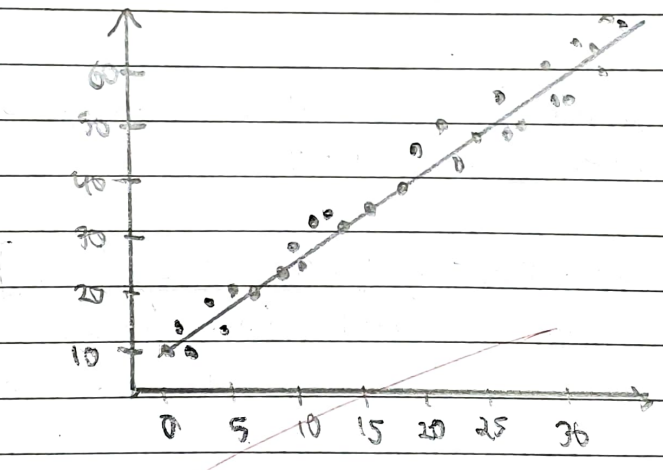
Date / /
Page

```
plt.plot (x_test, [predict(b0,b1,x) for x in
                   x_test], color = 'red', label
                   = 'Regression line')

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

OUTPUT:

[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,
20,21,22,23,24,25,26,27,28,29,30]

[12.36, 15.09, 13.28 ...]



LAB-2

Date __/__/__
Page _____

END-TO-END MACHINE LEARNING PROJECT

The dataset contains 20640 rows and 10 columns.
The data is split 80-20 with 80% training and 20 test data

1) Data Loading and Exploration:
   Dataset: housing.csv was loaded using pandas
   Exploration: .head(), .info(), .describe() used to inspect dataset structure.
   Visualization: Histograms and scatter plots were created to understand feature distributions

2) Data Splitting: Implemented random sampling and stratified sampling based on income categories to ensure representative test and training sets.

3) Data cleaning and preprocessing
   * Handling missing values using Simple Imputer to fill missing values with median values
   * Feature Engineering: Created new features such as rooms_per_household.
   * Categorical Encoding: Applied Ordinal Encoder and One Hot Encoder for categorical data transformation

4) Data Transformation Pipeline
   * Used Scikit-Learn Pipelines to automate preprocessing:
     Numerical Features: Imputation, feature scaling
     Categorical Features: One-Hot encoding

Date __/__/__
Page

5) Model Training and Evaluation:

Linear Regression :-

i) Model Training using Scikit-learn
housing_prepared: Preprocessed feature matrix after applying transformations
housing_labels: Target variable
fit(): Trains the model by finding the best-fitting line that minimizes the error.

ii) Making Predictions
The models predictions were compared with actual values to check performance

iii) Evaluating the Model
Root Mean Squared Error (RMSE) a common metric for regression models is us

LAB-3

Date __/__/__
Page _____

DECISION TREE - ID3 CLASSIFICATION

Dataset Used: PlayTennis.csv from Kaggle Datasets

Number of Columns: 5
Number of rows : 200

ID3 Algorithm :

function ID3 (Data, Target, Attributes):
1) If all instances in Data have the same class
   a) Return the class.

2) If Attributes is empty
   a) Return the most common class in Data

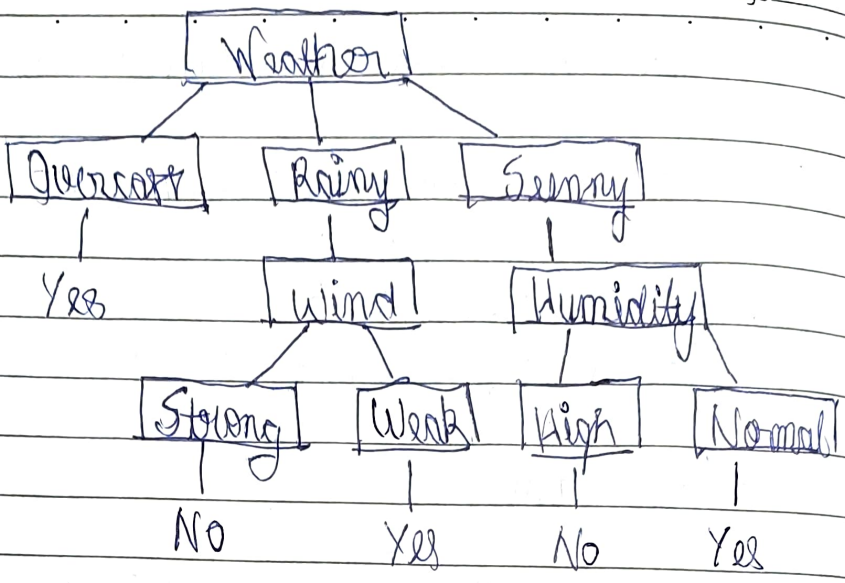
3) Select Best_Attribute with the highest information gain

4) Create a decision node with Best_Attribute

5) for each value v in Best_Attribute:
   a) Split Data where Best_Attribute = v
   b) Add a branch to the decision node:
      i) If Subset is empty:
         - Add a leaf with the most common class
      ii) Else:
         - Add the result of ID3 (Subset, Target, Attributes - Best_Attributes)

6) Return the decision tree

Date __/__/__
Page



One Prediction that was made :-

'Weather': 'Sunny'
'Temperature': 'Cool'
'Humidity': 'High'
'Wind': 'Strong'

Prediction : No

Decision Trees explanation in this case:-

1) If the weather is Overcast, the decision is Yes (Direct classification)
2) If the weather is rainy, the next check is Wind.
   * If the wind is Strong, decision is No
   * If the wind is Weak, decision is Yes
3) If the weather is Sunny, the next check is humidity
   * If the humidity is high, the decision is No
   * If the humidity is Normal, the decision is Yes.

Date __/__/__
Page _____

Information gain Formula:

$$S(T) - \sum_{v \in \text{Values}(X)} \frac{|S_v|}{S} S(T_v)$$

$S(T)$: Entropy of the whole dataset
v = Possible values of attribute X
S_v = Subset of S where attribute $X = v$
$|S_v|$ = Number of instances in subset S_v
$|S|$ = Total number of instances in S.

LAB-4

Date __/__/__
Page

KNN Algorithm:

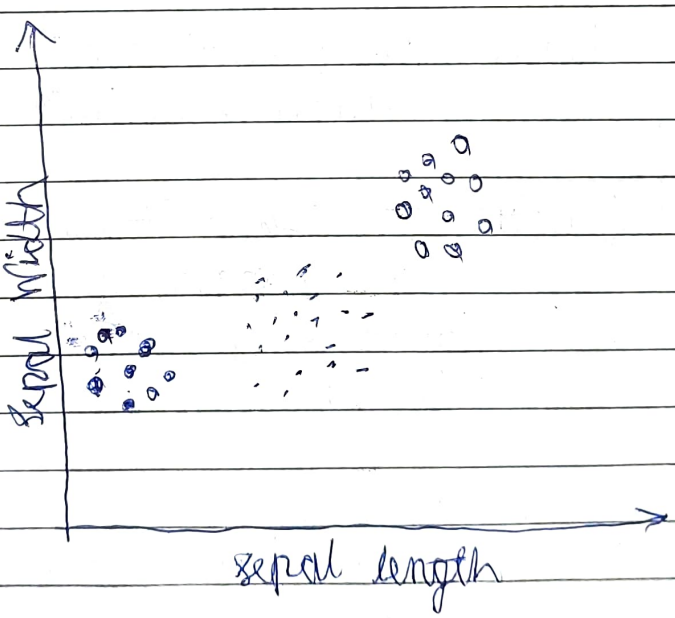
```
function split(data, 0.2):
    split_index ← length(data) × (1-0.2)
    train = first split_index elements of data
    test = remaining elements of data
    return train, test

function euclidean(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_instance, k):
    distances = []
    for each train_instance in train:
        dist = euclidean(test_inst, train_inst, features)
        distances.append(train_instance, dist)

    distances.sort()
    neighbours ← first K elements of distances
    return neighbours

function predict(train, test_instance, k):
    neighbours ← get_neighbours(train, test_instances, 3)
    count occurrences of each class in neighbours

    return class with max(count)
```

Date __/__/__
Page _____

function knn():
    train, test = split_dataset(dataset, 0.2)
    K = 3
    predictions = []

for each test instance in test_set:
    predicted_class = predict(train, test_instance, features, k)
    predictions.append(predicted_class)

Dataset used is iris.csv :-
Consists of 2 features = sepal length
sepal width
Multiclass target variable = Iris setosa
Iris versicolor
Iris virginica



SVM Algorithm

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

function train_test_split(dataset, test_ratio):
    split_index = length(dataset) * (1 - test_ratio)
    train_set = dataset[:split_index]
    test_set = dataset[split_index]
    return train_set, test_set

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

function train_svm(train, learning_rate, lambda_param, epochs)
    w ← [0, 0]
    b ← 0

for epoch in epochs:
    for each data point (x, y) in train:
        if (y * (dot_product(w, x) + b) >= 1:
            update w = w - learning_rate * 2 * lambda_param * w
```

Date __/__/__
Page _____

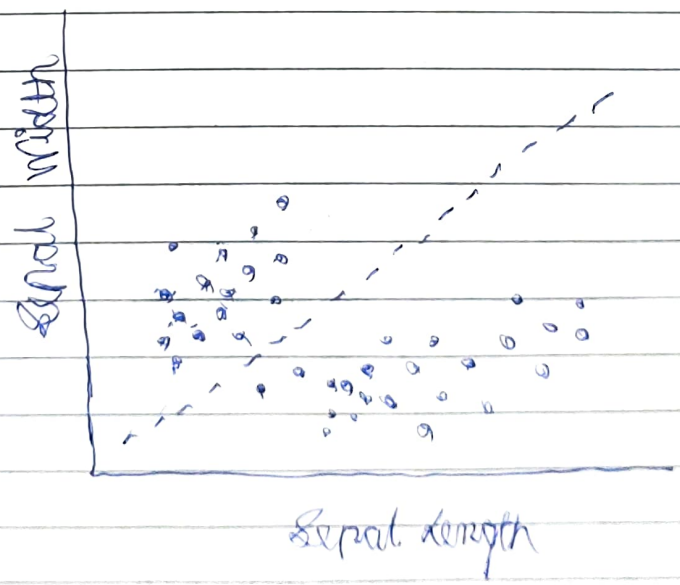
else:
update w = w - learning-rate * (2 * lambda - param * w - y * x)
update b = b + learning-rate * y
end if
return w, b

function predict(x, w, b):
return 1 if dot_product(w, x) + b >= 0 else 0

filename = "content/iris.csv"
dataset = load_dataset(filename)
train, test = train_test_split(dataset)
w, b = train_svm(train)

plot_svm(train, w, b)

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



Date / /
Page

LAB - 5

RANDOM FOREST :-

1) Load the dataset
2) Split the dataset into training and testing

function train_test_split (data, test_size):
test_len = int (len(data) * test_size)
data_copy = shuffle(data)
return training and test set.

3) GINI_INDEX (groups, class_labels)
Initialize total instances as sum of the lengths of all groups.
Initialize gini = 0
For each group in groups:
if the group skip
Calculate proportion of each class in the group

4) Test_split (index, value, dataset)
- Initialize empty left and right groups

5) get_split (dataset)
get the unique class labels from the dataset

6) To_terminal (group)
- Return the most common class label from the group.

Date __/__/__
Page _____

7) SPLIT (node, max_depth, min_size, current_depth)
Extract left and right groups from node groups

8) BUILD TREE (train_data, max_depth, min_size)
Initialize root node using GET_SPLIT on train_data.

9) PREDICT (tree, row)
If row[tree, index] < tree.value
if tree.left is a dictionary (subtree)
recurse left
Else, return tree left
Else:
if tree.right is a dictionary, recurse right
Else return tree.right.

Date __/__/__
Page

LAB- 6

Boosting Pseudocode

function AdaBoost:
    Input : X (Training features), y (Training labels), n_clf (Number of classifiers)
    Output : Trained AdaBoost model

Initialize an empty list 'clfs' to store the decision stumps

Initialize sample weights:
    weights = [1/number of samples]

for each classifier in the range of n_clf:
    Initialize best_classifier to store the best decision stump
    Initialize min_error to a large value

    For each feature in the dataset
        for each unique threshold in that feature
            for each polarity (1 or -1)
                Make predictions based on the feature

    Calculate classifier weight (alpha):
        alpha = 0.5 * log((1-min_error)/min_error)

    Make predictions using best_classifier and update sample weights

Date __/__/__
Page _____

Function predict:
Input: X, Trained AdaBoost model
Initialize clf_preds to store prediction of each classifier
Get classifier prediction and multiply by classifier weights.

Main Program:
Call AdaBoost.fit() on the training set
Call AdaBoost.predict() on the test set

Calculate accuracy: period accuracy
Plot accuracy versus no of classifiers.

KMEANS PSEUDOCODE
function KMeans(X, K, max_iters=100, tol=1e-4)
Inputs: X (data points, matrix of size (n_samples, n_features))
K: Number of clusters

Step 1: Randomly initialize the centroids
Step 2: Repeat for max_iters times or until convergence.

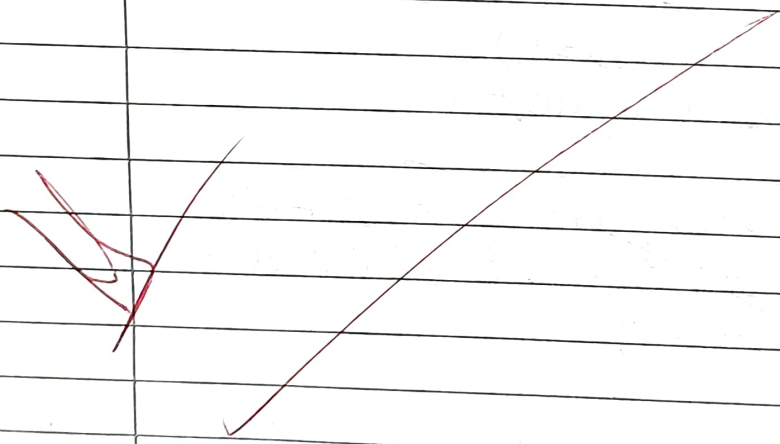
Step 3a: Assign each point to nearest centroid
Calculate the distances from each data point to each centroid
Step 3b: Update the centroids

Step 3c: Check for convergence
If the centroids don't change significantly break the loop.

Date
Page

Step 3d: If convergence is not reached update centroids and repeat.

Return the centroids and the label of the data points.



Date __/__/__
Page _____

PCA (Principal Component Analysis)

1) Standardize the dataset (mean = 0, std = 1 for each feature in X):
   subtract mean and divide by standard deviation

2) Compute the covariance matrix of the standardized data.
   $cov = (X.T \cdot X) / (n\text{-samples} - 1)$

3) Compute eigenvalues and eigen vectors of the covariance matrix.
   eig_vals, eig_vecs = eig(cov)

4) Sort eigenvectors by decreasing eigen values (variance) sort eig_vecs according to sorted eig_vals in decreasing order.

5) Select the top k eigen vectors.
   $w = \text{eig\_vecs}[:, :k]$

6) Project the data onto the new subspace
   $X\text{-reduced} = X \cdot w$