8/8/25

ML - LAB-1

To-Do

1) import pandas as pd
data = { 'Name': ['U', 'V', 'W', 'X', 'Y'],
'USN': ['111', '222', '333', '444', '555'],
'Marks': [64, 30, 80, 90, 50]
}

df = pd.DataFrame(data)
print("Sampledata")
print(df.head())

2) from sklearn.datasets import load_diabetes

diabetes = load_diabetes()
df = pd.DataFrame(diab.data, cols=diab.feature_names)

df['target'] = diab.target
print("Sample")
print('df.head())

3) file_path = '/content/sample_data/Californiahousing.csv'

df = pd.read_csv(file_path)
print("Sample")
print("df.head())
Print("\n")

4) import pandas as pd
df = pd.read_csv('/content/Dataset of Diabetes.csv.zip')
print(df.head(1))

df.to_csv('output.csv', index=False)
print("Data saved to output.csv")

5) import yfinance as yf
import pandas as pd
import matplotlib.pyplot as plt

tickers = ["HDFCBANK.NS", "ICICIBANK.NS", "KOTAKBANK.NS"]

data = yf.download(tickers, start="2020-01-01", end="2024-12-31", group_by='ticker')

print("First 5 rows of the dataset:")
print(data.head())

6) plt.figure(figsize=(12,6));
plt.subplot(2,1,1)
HDFCBANK.data['close'].plot
(title="HDFCBANK - Closing price")
plt.subplot(2,1,2)
HDFCBANK.data['Daily Return'].plot
(title="HDFCBANK - Daily Returns", color='orange')
plt.tight_layout()
plt.show()

Date __/__/__
Page _____

LAB-2

ID3 Algorithm

ID3 algorithm is a popular decision tree algorithm used in machine learning. It aims to build decision Tree by iteratively selecting the best attribute to split the data based on information gain.

Each node represents a test on an attribute and each branch represent a possible outcome of the test.

$$H(S) = \sum -(p_j * \log_2(p_j)) \rightarrow \text{Entropy}$$

$$IG(A,D) = H(S) - \sum_v \frac{|S_v|}{|S|} \times H(S_v)] \rightarrow \text{Information Gain}$$

Pseudocode of ID3

```
def ID3(D,A):
    If D is pure or A is empty:
        Return a leaf node with
        the majority class in D
    else:
        A_best = argmax (Information
                         gain (D, A)
        root = Node (A_best)
        for v in values (A_best):
            D_v = subset (D, A_best,
                          v)
            child = ID3 (D_v, A-S
                         A_best)
            root.add - child (v, child)
    return root
```

outlook <= 10.5
entropy = 0.94
samples = 14
value = [5, a]
class = 1

True

entropy = 0.0
samples = 4
value = [0, 4]
class = 1

False

outlook <= 1.5
entropy = 1.0
samples = 10
value = [5, 5]
class = 0

entropy = 0.971
samples = 5
value = [2, 3]
class = 1

entropy = 0.971
samples = 5
value = [3, 2]
class = 0

17/03/25 LAB-3

Date __/__/__
Page

ID 3 - Code Implementation without using sklearn library

```
import pandas as pd
import numpy as np

data = pd.read_csv("weather.csv")
df = pd.DataFrame(data)

def entropy(data):
    class_counts = data.value_counts()
    prob = class_counts / len(data)
    return -np.sum(prob * np.log2(prob))

def information_gain(df, feature, target):
    total_entropy = entropy(df[target])
    feature_value = df[feature].unique
    weighted_entropy = 0
    for value in feature_values:
        subset = df[df[feature] ==
                    value]
        weighted_entropy += (len
                (subset) /
            len(df))
            * entropy
        (subset[target])
    return total_entropy - weighted_entropy

def id3(df, features, target):
    if len(df[target].unique()) == 1:
        return df[target].iloc[0]
```

```
    if not features:
        return df[target].mode()[0]
    gains = { feature: information_gain
                (df, feature, target) for
                feature in features }
    best_feature = max(gains, key = gains.
    tree = { best_feature : { } }                get)

    for value in df[best_feature].unique():
        subset = df[df[best_feature] == value]
        tree[best_feature][value] = id3(subset,
                    [f for f in features if
                        f != best_feature],
                    target)
        return value tree

features = ['outlook', 'temperature',
            'Humidity', 'wind']
target = 'Playtennis'
desc_tree = id3(df, features, target)
print(desc_tree)
```

OUTPUT

```
{'Outlook' : { 'Sunny' :
    { Humidity : {'high' : 'No', 'how': 'yes'}}
    'overcast': 'Yes', Rain' : { 'wind' :
                {'weak' : 'Yes',
                'strong' : 'No' }}}}
```

Date __/__/__
Page _____

End to End ML Project

① Get Data

```
from sklearn.datasets import fetch_california_housing
cal_house = fetch_california_housing(as_frame=True)
print(cal_house.DESCR)
cal_house.head()
```

② Discover & Visualize the data to gain Insights

```
import matplotlib.pyplot as plt
import seaborn as sns

print( cal_house.frame.isnull().sum())

plt.figure(figsize=(8,6))
sns.histplot(cal_house.frame['MedHouseVal'],
             kde=True)
plt.title('Dist')
plt.show()
```

③ Prepare the data for ML Algo

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

X = cal_house.frame.drop('MedHouseVal', axis)
```

Date / /

Page

```
y = cal_house.frame['MedHouseVal']

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

Date __/__/__
Page _____

```
scaler = StandardScaler()
X_train_select = scaler.fit_transform(
                                X_train)
X_test_scaler = scaler.transform(X_test)
```

④ Select a model and train it

```
from sklearn.linear_model import Linear
                                Regression
from sklearn.metrics import mean_
  r2_score                  squared_error,
linear_model = LinearRegression()
linear_model = fit(X_train_scaled,
                                y_train)
y_pred_linear = linear_model.predict
                (X_test_mse_linear =
                 mean_square_error
                  (y_test, y_pred_
                                linear)
r2_linear = r2_score(y_test, y_pred_
                                linear)
print(f mes_linear), {r2_linear})
```

⑤ Fine Tune Your Model

```
from sklearn.metrics import root_
                        mean_squared
import numpy as np                -error
y_pred = model.predict(X_test)
rmse = root_mean_squared_error(y_test,
print(f rmse})                  y_pred)
```

LAB-4

Simple Linear Regression

```
function linear_reg(x, y, a, num)

    m = 0
    b = 0
    n = length(x)

    for i ← 1 to num
        pred = []
        for i ← 0 to n-1
            pred = m[i] + b
            append pre to pred

        errors = []
        for j ← 0 to n-1
            error = pred[j] - y[j]
            error.append(error)

    cost = (1/n) * Sum(error^2 for error in errors)

    m_gradient = 2/n * sum(error[j] * x[j] for j ← 0 to n-1)

    b_gradient = 2/n * sum(error)

    m = m - m_gradient * a
    b = b - b_gradient * a

    return m, b
```

Date __/__/__
Page _____

Logistic Linear Regression

```
function logistic_regression(x, y, num)

    m = length(x[0])
    n = length(x)
    weights = array of 0 to length in
    bias = 0

    for i 1 to num:
        pred = []
        for i <- 0 to n-1:
            lin = dot_product(weights, x[i])
                  + bias
            pred = sigmoid(lin)
            pred.append(pred)

    errors = []
    for j <- 0 to n-1
        error = pred[j] - y[j]
        error.append(error)

    cost = -(1/n) * sum(y[j] * log(pred
                                   [j])
           + (1 - y[j]) * log(1 - pred[j])
           for j from 0 to n-1)

    weight_grad = (1/n) * dot_product
                          (errors, x)
    bias_grad = (1/n) x sum(errors)

    weights = weights - weight_grad
    bias = bias - l x bias_grad
return weight, bias
```

Date __/__/__

Page _____

Algo

Linear Regression

1. Initialise m & b = 0
2. make prediction mX+b
3. Find error by y - pred
4. update m & b using gradient descent
5. Repeat 4 steps for num-iteration

Algo logistics Regression

1. m be the length of features initialized to 0 ;
2. n = no of data points
3. Initialize weight & bias to 0
4. Make predictions weight X + bias
5. Use sigmoid function for pred.
6. Find errors using pred - y[i]
7. Update weight & bias using gradient descent
8. Repeat steps 4 Return weight & bias

Algo multilinear Regression

Same as linear but with multiple features.

Date __/__/__
Page ____

Multilinear Regression

function multilinear_reg(x, y, l, num):

m = length(x[0])
weight = array of 0 of length n
bias = 0
n = length(x)

for i ← 1 to num:
    pred = []
    for j ← 0 to n-1
        pl = dot_product(weight, x[j]) + bias
        pred.append(pl)

errors = []
for j ← 0 to n-1
    error = pred[j] - y[j]
    errors.append(error)

cost = 1/n * sum((1 - y[j])).error° in error in error)

weight_grad = 1/n x dot_prod(error, x)

bias_grad = 1/n x dot sum(error)

Return weight & bias

7/4/25

LAB-5

KNN valgorithm

1. choose the numbes of neighbors(k)

   eg :- K = 3

2. calculate distance from the test points to all training points

   use a distance metric like.
   Euclidean distance:

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + \cdots}$$

3. Sort the distances in ~~ord~~ ascending order of the Euclideon's distance

4. Select the k ~~num~~ nearest neighbors
   Pick the top k points with the smallest distance.

5. Do a majority vote (for classification)
   → Count the labels of the k neighbors.
   → The label with the most votes is the predicted class.

   (for Regression) : average of the k nearest neighbors

Date / /
Page

SVM algorithm

1. Data loading and preprocessing
2. Split the Dataset
3. Initialize the SVM classifier
4. One-vs-Rest Training Strategy
5. Binary SVM Training:
6. Prediction Phase
7. Evaluation

→ • Store the score
• Predict the class with maximum decision score

→ Compare predicted labels ypred with true labels ytest
Calculate accuracy:-
Accuracy = Total test samples Number of correct predictions × 100

inputs:

D: Training dataset $D = \{(x_1, y_2) \ldots (x_n, y_n)\}$

where $y_i \in \{+1, -1\}$

C: Regularization parameter (for soft margin SVM)

Date / /
Page

CAB - 6

i) Random forest Ensemble Method

1. Load dataset D with features X and target y

2. preprocess the data:
   - Handle missing value
   - Encode categorical variables
   - split dataset into training set D-train and test set D-test

3. Initialise RandomForest with params
   - n_estimators (no. of trees)
   - max_depth (max depth of trees)
   - RandomState (seed for replicating)

4. For each tree t = 1 to n_estimators
   - Randomly select a bootstrap sample of data points from D-train set

5. After all trees are built, make predictions:
   a) for each test instance X-test, aggregate predictions from all trees using majority voting
   b) prediction = mode (prediction from all trees)

6. Evaluate the model:
   calculate accuracy, precision, recall, F1 score

7) Return Model Performance

Date __/__/__
Page _____

LAB-7

Boosting Ensemble Method

1. Load dataset D with features X and target Y
2. preprocess the data
   a. Handle missing values
   b. Encode categorical variables
   c. Split dataset into training set D-train and test set D-test
3. Initialise the weights of all training sample equally
   $w_1 = w_2 = \ldots = w_N = 1/N$
   where N is the no of training sample
4. For each interaction $t = 1$ to n-estimation:
   a. Train a weak classifier h_t using the weighted dataset D-train with sample weights w_i
   b. calculate the error of the classifier
   $$\varepsilon_t = \frac{\sum w_i \times I(h_t(x_i) = y_i)}{\sum w_i}$$
   where I is the indicator function
   c) Compute the classifier weight (α_t):
   $$\alpha_t = 0.5 * \log\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$$
   d) Update the weights of misclassified points

c) normalize the weight so that they sum to 1:

$$w_i = w_i / \sum w_i$$

5. After all iterations, combine weak classifiers:
   a) Final prediction for test point
   $$x = \text{sign}(\sum \alpha_t h_t(x))$$
6. Evaluate the model:
   a) Calculate accuracy, Precision, Recall, F1-Score
7. Return model performance

Date __/__/__

Page _____

LAB-8

K-means clustering Algorithm

1. Load dataset D with datapoints x_i, where $x_i \in R^d$ (d-dimensional space)

2. Preprocess the data:
   a. Handling the missing values
   b. Standardize the data

3. Initialize k centroids randomly from the dataset:
   $C_1, C_2, \ldots C_k$

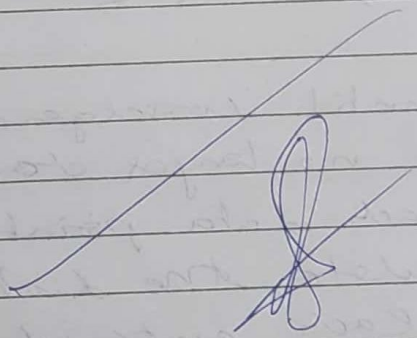
4. Repeat until convergence (centroids no longer change):
   a. For each data point x_i:
      i) Calculate the distance to each centroid C_j:
      $distance(x_i, C_j) = ||x_i - C_j||_2$
      ii) Assign point x_i to the closest centroid
      $cluster(x_i) = \arg\min_j distance(x_i, C_j)$
   b. Update centroids by calculating the mean of points in each cluster:

5. Convergence: If centroid do not change significantly b/w iterations, stop

Date __/__/__
Page _____

6. output :
   a. Find centroids C-1, C-2 ... C-k
   b. cluster assignments for each data point

7. Optionally calculate clustering metrics (eg. Silhouttes, Sse, method)



Date __/__/__

Page _____

LAB-9

Principle Component Analysis

1. Load data D with data points x_i where $x_i \in R^d$ (d - dimensional space)
2. Pre process the data:
   a. Handle missing values
   b. Standardize the data by substracting mean of each feature
   $x_j = (x_j - \mu_j) / \sigma_j$
   where μ_j is the mean of feature
   j & σ_j is the standard deviation
3. Compute the covariance matrix Σ of standardize data:
   $\Sigma = (1/N) * X^T * X$
   where X is the data matrix (each row is a data point)
4. Perform eigenvalue decomposition on the covariance matrix Σ:
   $\Sigma * V = \lambda * V$
   where λ are the eigenvalue, and V are the eigenvectors.
5. Sort the eigenvalues in descending order and select the top K eigenvectors.

6. project the data onto the k eigenvectors (PCA):

$X\_new = X * V$

where V is the matrix of the top k eigenvectors

output:

Transformed dataset X_new (reduced dimensions)

optionally, visualize the reduced data (if K=2 or K=3)

16/5