# **Practical – 3**

**Aim:** Implement and demonstrate the FIND-S Algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file

* **Code:**

import pandas as pd

df = pd.read\_csv("/ws.csv")

print("Columns in ws.csv:", df.columns)

features = df.columns[:-1]

target = df.columns[-1]

df[features] = df[features].astype(str)

df[target] = df[target].astype(str)

positive\_example = df[df[target] == 'Yes'][features].values

def find\_s(example):

if len(example) == 0:

return "No positive examples found."

hypothesis = example[0].copy()

print("Initial hypothesis:", hypothesis)

for i, instance in enumerate(example[1: ]):

for j in range(len(hypothesis)):

if hypothesis[j] != instance[j]:

hypothesis[j] = '?'

print(f"Hypothesis after example {i+2}:", hypothesis)

return hypothesis

hypothesis = find\_s(positive\_example)

print("Most specific hypothesis:", hypothesis)

* A close up of a text

  AI-generated content may be incorrect.**Output:**

# **Practical – 4**

**Aim:** Write a Python program to implement Simple Linear Regression

* How many total observations in data?
* How many independent variables?
* Which is a dependent variable?
* Quantify the goodness of your model and discuss steps taken for improvement (RMSE, SSE, R2Score).
* **Code:**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import train\_test\_split

from pandas.core.common import random\_state

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, r2\_score

df = pd.read\_csv("/Salary\_Data.csv")

df.head()

df.describe()

X = df.iloc[:, :1] # independent

y = df.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)

rmse = np.sqrt(mean\_squared\_error(y\_test, y\_pred\_test))

sse = np.sum((y\_test - y\_pred\_test) \*\* 2)

r2 = r2\_score(y\_test, y\_pred\_test)

print(f"\nModel Evaluation:")

print(f"RMSE: {rmse:.4f}")

print(f"SSE: {sse.iloc[0]:.4f}")

print(f"R² Score: {r2:.4f}")

plt.scatter(X\_train, y\_train, color = 'green')

plt.plot(X\_train, y\_pred\_train, color = 'firebrick')

plt.title('Salary vs Experience (Training Set)')

plt.xlabel('Years of Experience')

plt.ylabel('Salary')

plt.legend(['X\_train/Pred(y\_test)', 'X\_train/y\_train'], title = 'Sal/Exp', loc='best', facecolor='lightblue')

plt.box(False)

plt.show()

* A graph of a salary vs experience

  AI-generated content may be incorrect.**A black text on a white background

  AI-generated content may be incorrect.Output:**

# **Practical – 5**

**Aim:** Implementation of Multiple Linear Regression for House Price Prediction using sklearn.

* **Code:**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.linear\_model import LinearRegression

from sklearn.datasets import fetch\_california\_housing

housing = fetch\_california\_housing()

df = pd.DataFrame(housing.data, columns=housing.feature\_names)

df['Target'] = housing.target

df = df.sample(300, random\_state=42).reset\_index(drop=True)

df['Group'] = np.random.choice(['A', 'B', 'C'], size=len(df))

x\_feature = 'MedInc'

y\_feature = 'Target'

plt.figure(figsize=(10, 6))

colors = {'A': 'red', 'B': 'blue', 'C': 'green'}

for group in df['Group'].unique():

subset = df[df['Group'] == group]

X = pd.DataFrame(subset[[x\_feature]])

y = subset[y\_feature]

model = LinearRegression()

model.fit(X, y)

x\_line = np.linspace(X[x\_feature].min(), X[x\_feature].max(), 100).reshape(-1, 1)

y\_line = model.predict(pd.DataFrame(x\_line, columns=[x\_feature]))

plt.scatter(X, y, alpha=0.5, label=f'Group {group}', color=colors[group])

plt.plot(x\_line, y\_line, color=colors[group], linewidth=2)

plt.xlabel(x\_feature)

plt.ylabel(y\_feature)

plt.title(f"Linear Regression Lines for {x\_feature} by Group")

plt.legend()

plt.grid(True)

plt.tight\_layout()

plt.show()

* **Output:**

# **Practical – 6**

**Aim: Two Class Classification (Logistic Regression)**

* How many total observations in data?
* How many independent variables?
* Which is a dependent variable?
* Implement logistic function.
* Implement Log-loss function.
* Quantify the goodness of your model and discuss steps taken for improvement (Accuracy, Confusion matrices, F-measure).
* **Code:**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.datasets import fetch\_california\_housing

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import (

accuracy\_score, confusion\_matrix, f1\_score, log\_loss, classification\_report

)

housing = fetch\_california\_housing()

df = pd.DataFrame(housing.data, columns=housing.feature\_names)

df['Target'] = housing.target

median\_value = df['Target'].median()

df['Target\_Binary'] = (df['Target'] > median\_value).astype(int)

X = df.drop(columns=['Target', 'Target\_Binary'])

y = df['Target\_Binary']

print("Total observations:", len(df))

print("Number of independent variables:", X.shape[1])

print("Dependent variable: Target\_Binary")

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

model = LogisticRegression(max\_iter=1000)

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

y\_pred\_prob = model.predict\_proba(X\_test)[:, 1]

accuracy = accuracy\_score(y\_test, y\_pred)

f1 = f1\_score(y\_test, y\_pred)

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

logloss = log\_loss(y\_test, y\_pred\_prob)

print("\nAccuracy:", accuracy)

print("F1 Score:", f1)

print("Log Loss (sklearn):", logloss)

print("\nClassification Report:\n", classification\_report(y\_test, y\_pred))

def logistic(z):

return 1 / (1 + np.exp(-z))

def log\_loss\_manual(y\_true, y\_prob):

eps = 1e-15

y\_prob = np.clip(y\_prob, eps, 1 - eps)

return -np.mean(y\_true \* np.log(y\_prob) + (1 - y\_true) \* np.log(1 - y\_prob))

manual\_logloss = log\_loss\_manual(y\_test.values, y\_pred\_prob)

print("Manual Log Loss:", manual\_logloss)

plt.figure(figsize=(6, 4))

sns.heatmap(conf\_matrix, annot=True, fmt="d", cmap="Blues")

plt.title("Confusion Matrix")

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.tight\_layout()

plt.show()

* A screenshot of a computer

  AI-generated content may be incorrect.A blue squares with white text

  AI-generated content may be incorrect.**Output:**

# **Practical – 7**

**Aim: Implementation of Decision tree using sklearn and its parameter tuning.**

* **Code:**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.datasets import fetch\_california\_housing

from sklearn.model\_selection import train\_test\_split, GridSearchCV

from sklearn.tree import DecisionTreeClassifier, plot\_tree

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report

housing = fetch\_california\_housing()

df = pd.DataFrame(housing.data, columns=housing.feature\_names)

df['Target'] = housing.target

df = df.sample(n=200, random\_state=42).reset\_index(drop=True)

df['Target\_Binary'] = (df['Target'] > df['Target'].median()).astype(int)

X = df.drop(columns=['Target', 'Target\_Binary'])

y = df['Target\_Binary']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

clf = DecisionTreeClassifier(random\_state=42)

param\_grid = {

'max\_depth': [3, 5, None],

'min\_samples\_split': [2, 10],

'criterion': ['gini', 'entropy']

}

grid = GridSearchCV(clf, param\_grid, cv=5, scoring='accuracy')

grid.fit(X\_train, y\_train)

best\_tree = grid.best\_estimator\_

print("Best Parameters:", grid.best\_params\_)

y\_pred = best\_tree.predict(X\_test)

acc = accuracy\_score(y\_test, y\_pred)

cm = confusion\_matrix(y\_test, y\_pred)

print("\nAccuracy:", acc)

print("\nConfusion Matrix:\n", cm)

print("\nClassification Report:\n", classification\_report(y\_test, y\_pred))

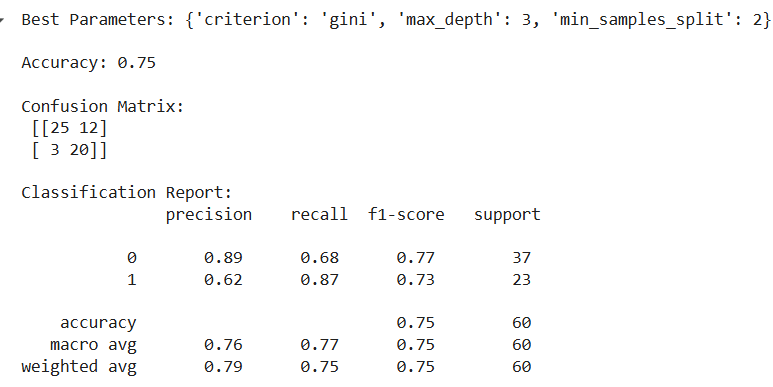
plt.figure(figsize=(16, 8))

plot\_tree(best\_tree, feature\_names=X.columns, class\_names=["Low", "High"], filled=True, rounded=True)

plt.title("Decision Tree Visualization")

plt.show()

* A diagram of a tree

  AI-generated content may be incorrect.**Output:**

# **Practical – 9**

**Aim: Write a program to implement Random Forest Algorithm**

* **Code:**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.datasets import fetch\_california\_housing

from sklearn.ensemble import RandomForestClassifier

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report

housing = fetch\_california\_housing()

df = pd.DataFrame(housing.data, columns=housing.feature\_names)

df['Target'] = housing.target

df = df.sample(n=300, random\_state=42).reset\_index(drop=True)

median\_value = df['Target'].median()

df['Target\_Binary'] = (df['Target'] > median\_value).astype(int)

X = df.drop(columns=['Target', 'Target\_Binary'])

y = df['Target\_Binary']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

rf = RandomForestClassifier(

n\_estimators=100,

max\_depth=None,

random\_state=42

)

rf.fit(X\_train, y\_train)

y\_pred = rf.predict(X\_test)

acc = accuracy\_score(y\_test, y\_pred)

cm = confusion\_matrix(y\_test, y\_pred)

print("Accuracy:", acc)

print("\nConfusion Matrix:\n", cm)

print("\nClassification Report:\n", classification\_report(y\_test, y\_pred))

plt.figure(figsize=(6, 4))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')

plt.title("Confusion Matrix")

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.tight\_layout()

plt.show()

importances = rf.feature\_importances\_

feature\_names = X.columns

sorted\_idx = np.argsort(importances)[::-1]

plt.figure(figsize=(10, 6))

sns.barplot(x=importances[sorted\_idx], y=feature\_names[sorted\_idx])

plt.title("Feature Importances in Random Forest")

plt.tight\_layout()

plt.show()

* A screenshot of a computer

  AI-generated content may be incorrect.**Output:**

A blue squares with white text

AI-generated content may be incorrect.

A graph with blue bars

AI-generated content may be incorrect.

# **Practical – 10**

**Aim: Write a program to implement Random Forest Algorithm**

* **Code:**

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.naive\_bayes import GaussianNB

from sklearn.preprocessing import LabelEncoder

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report

import seaborn as sns

import matplotlib.pyplot as plt

df = pd.read\_csv("/content/drive/MyDrive/temp/1\_Housing.csv")

# converting data into binaries 1 for high and 0 for low

median\_price = df['price'].median()

df['price\_category'] = (df['price'] > median\_price).astype(int)

df.drop(columns=['price'], inplace=True)

categorical\_cols = df.select\_dtypes(include='object').columns

le = LabelEncoder()

for col in categorical\_cols:

df[col] = le.fit\_transform(df[col])

X = df.drop(columns=['price\_category'])

y = df['price\_category']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

model = GaussianNB()

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

cm = confusion\_matrix(y\_test, y\_pred)

print("Accuracy:", accuracy)

print("\nConfusion Matrix:\n", cm)

print("\nClassification Report:\n", classification\_report(y\_test, y\_pred))

plt.figure(figsize=(6, 4))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')

plt.title("Naïve Bayes Confusion Matrix")

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.tight\_layout()

plt.show()

* A blue squares with white text

  AI-generated content may be incorrect.**A screenshot of a computer

  AI-generated content may be incorrect.Output:**

# **Practical – 11**

**Aim: Write a program to implement OR, AND gate using Perceptron with learning rules.**

* **Code:**

import numpy as np

def step\_function(x):

return 1 if x >= 0 else 0

def train\_perceptron(inputs, labels, learning\_rate=0.1, iterations=5):

weights = np.zeros(inputs.shape[1])

bias = 0

for iteration in range(iterations):

print(f"\nIteration no: {iteration+1}")

for x, label in zip(inputs, labels):

z = np.dot(x, weights) + bias

y\_pred = step\_function(z)

error = label - y\_pred

weights += learning\_rate \* error \* x

bias += learning\_rate \* error

print(f"Input: {x}, Pred: {y\_pred}, Error: {error}, Updated Weights: {weights}, Bias: {bias}")

return weights, bias

def predict(inputs, weights, bias):

return [step\_function(np.dot(x, weights) + bias) for x in inputs]

inputs = np.array([[0,0],[0,1],[1,0],[1,1]])

or\_labels = np.array([0,1,1,1])

and\_labels = np.array([0,0,0,1])

# OR Gate

print("Training Perceptron for OR Gate")

or\_weights, or\_bias = train\_perceptron(inputs, or\_labels)

or\_outputs = predict(inputs, or\_weights, or\_bias)

print("\nFinal OR Predictions:")

for i, o in zip(inputs, or\_outputs):

print(f"Input: {i}, Output: {o}")

# AND Gate

print("\n\nTraining Perceptron for AND Gate")

and\_weights, and\_bias = train\_perceptron(inputs, and\_labels)

and\_outputs = predict(inputs, and\_weights, and\_bias)

print("\nFinal AND Predictions:")

for i, o in zip(inputs, and\_outputs):

print(f"Input: {i}, Output: {o}")

* A screenshot of a computer program

  AI-generated content may be incorrect.**A screenshot of a computer program

  AI-generated content may be incorrect.Output:**

# **Practical – 12**

**Aim:  Build an Artificial Neural Network by implementing the Backpropagation Algorithm.**

* **Code:**

import numpy as np

def sigmoid(x):

return 1 / (1 + np.exp(-x))

def sigmoid\_derivative(x):

return x \* (1 - x)

def mse(y\_true, y\_pred):

return np.mean((y\_true - y\_pred) \*\* 2)

X = np.array([[0,0], [0,1], [1,0], [1,1]])

y = np.array([[0], [1], [1], [1]])

np.random.seed(42)

input\_size = 2

hidden\_size = 4

output\_size = 1

W1 = np.random.randn(input\_size, hidden\_size)

b1 = np.zeros((1, hidden\_size))

W2 = np.random.randn(hidden\_size, output\_size)

b2 = np.zeros((1, output\_size))

learning\_rate = 0.1

epochs = 10000

for epoch in range(epochs):

# Forward pass

z1 = np.dot(X, W1) + b1

a1 = sigmoid(z1)

z2 = np.dot(a1, W2) + b2

a2 = sigmoid(z2)

# Backward pass

error = y - a2

d\_a2 = error \* sigmoid\_derivative(a2)

d\_W2 = np.dot(a1.T, d\_a2)

d\_b2 = np.sum(d\_a2, axis=0, keepdims=True)

d\_a1 = np.dot(d\_a2, W2.T) \* sigmoid\_derivative(a1)

d\_W1 = np.dot(X.T, d\_a1)

d\_b1 = np.sum(d\_a1, axis=0, keepdims=True)

# update weights

W2 += learning\_rate \* d\_W2

b2 += learning\_rate \* d\_b2

W1 += learning\_rate \* d\_W1

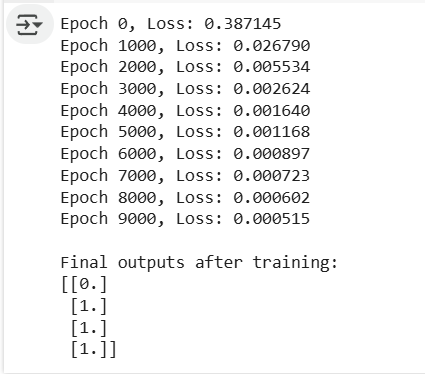
b1 += learning\_rate \* d\_b1

if epoch % 1000 == 0:

print(f"Epoch {epoch}, Loss: {mse(y, a2):.6f}")

print("\nFinal outputs after training: ")

print(np.round(a2))

* **Output:**